IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 8, 2020, accepted May 21, 2020, date of publication June 1, 2020, date of current version June 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2999085

Mapreduce-Based Distributed Clustering Method Using CF$^+$ Tree

HYEONG-CHEOL RYU AND SUNGWON JUNG, (Member, IEEE)
Department of Computer Science and Engineering, Sogang University, Seoul 121-742, South Korea

Corresponding author: Sungwon Jung (jungsung@sogang.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2015R1D1A1A01058001, and in part by the Ministry of Science and ICT (MSIT), South Korea, through the Information Technology Research Center (ITRC) support program, supervised by the Institute for Information & Communications Technology Promotion (IITP) under Grant IITP-2020-2017-0-01628.

ABSTRACT Clustering exceptionally large data sets is becoming a major challenge in data analytics with the continuous increase in their size. Summary-based clustering methods and distributed computing frameworks such as MapReduce can efficiently handle this challenge. These methods include BIRCH and its extension CF$^+$-ERC. CF$^+$-ERC can reduce the clustering time of large data sets by utilizing the structure of a CF$^+$ tree. However, CF$^+$-ERC is a sequential clustering method, so it cannot be used with multiple machines to reduce the clustering time. In this study, we propose a novel MapReduce-based distributed clustering method called CF$^+$-ERC on MapReduce (CF$^+$ERC_MR). It builds a CF$^+$ tree for clustering an exceptionally large data set with a given threshold and finds the final clusters using MapReduce, which significantly reduces the clustering time. Further, our method is scalable with respect to the number of machines. The efficacy of this method is validated through not only its theoretical analysis but also in-depth experimental analysis of exceptionally large synthetic and real data sets. The experimental results demonstrate that the clustering speed of our approach is far superior to that of the existing clustering methods.

INDEX TERMS Clustering, BIRCH, CF$^+$ tree, range query, very large data sets, MapReduce.

I. INTRODUCTION

Owing to the rapid advancement of the Internet and online technologies, exceptionally large collections of data containing digital traces from users and devices can be generated. Such large data sets can be generated by Twitter, Google, Yahoo, and Facebook [1]. Processing and analyzing such large data can provide valuable insights on the activity patterns and preferences of users or customers, which can strongly impact decision making in data-driven businesses. Clustering is a popular technique for analyzing large data sets in which the data set is divided into clusters based on some measure of similarity or distance.

Clustering methods are widely used in a variety of fields such as data science, machine learning, pattern recognition, and artificial intelligence [1]–[4]. These methods can be classified into three main categories: sampling-based, dimensionality reduction-based, and summary-based methods [5]. However, it is difficult to derive appropriate samples with sampling-based methods. Further, dimensionality reduction-based methods cannot extract the precise dimensions that maintain the key properties of the data sets. Therefore, summary-based clustering methods are more suitable for the analysis of large data sets.

BIRCH [6] is a popular summary-based clustering method that is used to solve real-world problems owing to its wide applicability and high-level scalability with respect to the data size [7]–[10]. In BIRCH, a clustering feature (CF) tree is built by using a threshold value, which is used to obtain a suitable data summary. When the threshold value is not given, BIRCH uses the computer's memory to estimate the threshold value, following which the existing clustering methods are used as the global clustering method. In some real-world problems, a threshold value is often given as a criterion of clustering [11]–[17]. This criterion is similar to t in a distance-t stopping condition [18], where two clusters within t are grouped into the same cluster.

Recently, CF$^+$-ERC [5] has been proposed as an extension of BIRCH by incorporating the global clustering method ERC (effective multiple range queries-based clustering).

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

ERC reduces the number of computations by using range queries based on the threshold value, which cluster rapidly. However, CF$^+$-ERC is a sequential clustering method, which consumes a significant amount of computation time to cluster exceptionally large data sets. Further, it cannot be implemented in a distributed computing environment, which implies that it cannot be used with multiple machines for rapid clustering.

Distributed computing algorithms such as MapReduce are an efficient alternative for handling large data sets [19]. Distributed clustering methods based on MapReduce have been useful in analyzing data sets. PKMeans [20] and *mrk*-means [21] can implement *K*-means in parallel using MapReduce. Such distributed clustering methods can be used as the global clustering method of BIRCH. However, these methods do not use the threshold value, which makes them unsuitable for clustering large data sets in which the threshold values are estimated. In addition, BIRCH that uses distributed clustering methods as the global clustering method cannot build the CF tree in a distributed parallel manner.

In this study, we propose a novel distributed clustering method CF$^+$ERC_MR, which is an extension of CF$^+$-ERC based on MapReduce. The CF$^+$ tree is built by using a given threshold value of a data set in the distributed computing environment. Note that our proposed method assumes data sets with given threshold values. Subsequently, ERC is invoked to cluster the data summary in a parallel way. The two key advantages of CF$^+$ERC_MR are the reduction in the clustering time and the utilization of the threshold value.

There are three main challenges for extending CF$^+$-ERC to the MapReduce programming paradigm as follows.

- The workload should be evenly distributed to multiple reduce tasks.
- Each reduce task should receive similar objects.
- The results of the reduce tasks should be refined.

The region centroid set is used to avoid unbalanced data distribution and collect similar data with the same reduce task. Because the clustering of a divided data set might not find the cluster based on the threshold, the refinement step merges the results of the multiple reduce tasks.

CF$^+$ERC_MR rapidly clusters the data sets compared to the existing clustering methods because it runs a MapReduce job using the data summary. Since CF$^+$ERC_MR uses the threshold value, it appropriately clusters the data sets of some domains that use their corresponding threshold values. We have compared the performance of our method with that of PKMeans, *mrk*-means, and BIRCH-based distributed clustering methods on MapReduce. Our method is scalable, i.e., it can work across many machines in a distributed computing environment. Further, it can analyze a large data set more rapidly and accurately than the existing clustering methods.

The remainder of the paper is organized as follows. CF$^+$-ERC and MapReduce are briefly described in Section II. The extension of CF$^+$-ERC to the MapReduce programming paradigm is discussed in Section III. Section IV discusses the theoretical and experimental performance analyses of our proposed method. Finally, the study is concluded in Section V.

II. RELATED WORKS

CF$^+$-ERC [5] is a state-of-the-art clustering method for analyzing large data sets. MapReduce [19] is a suitable programming paradigm to handle large data in a distributed computing environment. In this section, we have focused on a detailed description of CF$^+$-ERC because CF$^+$ERC_MR is an extension of CF$^+$-ERC to deal with data sets in a distributed parallel manner.

A. CF$^+$-ERC

CF$^+$-ERC builds a CF$^+$ tree and conducts multiple range queries on this tree, which drastically reduces the clustering time. A node is split if it overflows. The first step in the node split is to choose the farthest entry pair of the overflow node as the seeds. Two new entry sets are created, and the seeds are separated into these sets. The remaining entries are redistributed in the two new entry sets by considering the centroids of the current entry sets that contain several entries during the node split.

CF$^+$-ERC shares the definitions of the *CF* vector and *CF* tree along with the *CF additivity* theorem of BIRCH [6].

Definition 1: Given N d-dimensional objects in a cluster $\mathcal{C}$: $\{\vec{X}_i\}$, where $i = 1, 2, \ldots, N$, the CF vector of $\mathcal{C}$ is defined as $CF = (N, \vec{LS}, SS)$, where N is the number of objects in $\mathcal{C}$, $\vec{LS}$ is the linear sum of the N data, that is, $\vec{LS} = \sum_{i=1}^{N} \vec{X}_i$, and SS is the sum of the N squared data, that is, $SS = \sum_{i=1}^{N} \vec{X}_i^{\,2}$.

Definition 2: The CF$^+$ tree exhibits the following properties:

1) Each non-leaf node contains at most C entries in the form $[CF_i, child_i, r_i]$, where $i = 1, 2, \ldots, C$. CF_i is the CF vector of the i-th child entry, $child_i$ is a pointer to the i-th child node, and r_i is the radius covering all the descendant microclusters. The i-th entry of the non-leaf node is called a subcluster $\mathcal{SC}_i$.
2) Each leaf node contains at most L entries in the form $[CF_i]$, where $i = 1, 2 \ldots, L$ and CF_i is the CF vector of the i-th child entry. The i-th entry of the leaf node is called a microcluster $\mathcal{MC}_i$.
3) For efficient sequential scanning, all leaf nodes are linearly chained by the "*prev*" and "*next*" pointers.

Figure 1 shows the CF$^+$ tree constructed by using 50 objects. The CF$^+$ tree is built starting from the root node. Each data instance p recursively descends through the CF$^+$ tree by choosing the closest child node in accordance with the distance metric. When p reaches the leaf node, it is absorbed into the microcluster of the node if the threshold requirement is satisfied. Otherwise, a new entry is generated and p is included in that entry. If the node has room for the new entry, the new entry is inserted in the node. Otherwise, the node splits. Splitting of the leaf node causes the insertion of a new non-leaf entry into the parent node. The parent node splits if

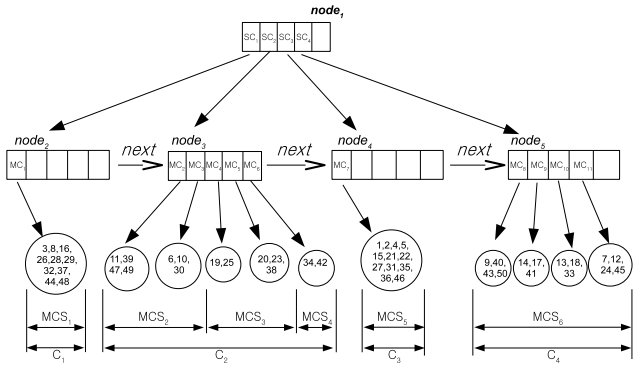


FIGURE 1. CF+ tree constructed by using 50 objects.

it overflows. The node split can be propagated up to the root node. When the root node splits, the height of the CF tree increases by one. If the propagation of the node split ceases at a non-leaf node, merging refinement is attempted to reduce the number of nodes.

Subsequently, the threshold value is used as the criterion for global clustering. The inter-microcluster distance (*IMD*) is used to compute the distance between two microclusters. *IMD* between two microclusters is the Euclidean distance between their centroids minus the sum of their radii. If *IMD* between two microclusters is less than the threshold value, they reside in the same final cluster. Note that the term ''final cluster'' refers to the cluster obtained by the clustering methods.

ERC is conducted by utilizing the structure of the CF+ tree and the threshold value T. It can be divided into two steps: partition and refinement steps. In the partition step, the linearly adjacent microclusters within T are grouped into a segment called microcluster segment ($\mathcal{MCS}$) using the *microcluster linearization* property of CF+ tree. For instance, in Figure 1, the CF+ tree has eleven microclusters ($\mathcal{MC}_1, \mathcal{MC}_2, \cdots, \mathcal{MC}_{11}$) that are linearly adjacent. Because *IMD* ($\mathcal{MC}_2$, $\mathcal{MC}_3$) (*IMD* between $\mathcal{MC}_2$ and $\mathcal{MC}_3$) is less than T, $\mathcal{MC}_2$ and $\mathcal{MC}_3$ are grouped into $\mathcal{MCS}_2$. Similarly, the partition step finds six microcluster segments $\mathcal{MCS}_1, \mathcal{MCS}_2, \cdots, \mathcal{MCS}_6$.

In the refinement step, the **e**ffective **r**efinement **s**tep (*ERS*) of *ERC* is employed to gather a set of $\mathcal{MCS}s$ into the set of final clusters using multiple range queries. For each $\mathcal{MCS}$, *ERS* employs an **e**ffective **r**ange **q**uery (*ERQ*) to find the *connections* between that $\mathcal{MCS}$ and other $\mathcal{MCS}s$ without having redundant query computation by using the structure of the CF+ tree. A *connection* between two microcluster segments $\mathcal{MCS}_i$ and $\mathcal{MCS}_j$ implies that at least one *IMD* between $\mathcal{MC}_x$ and $\mathcal{MC}_y$ is less than T, where $\mathcal{MC}_x \in \mathcal{MCS}_i$ and $\mathcal{MC}_y \in \mathcal{MCS}_j$. This indicates that $\mathcal{MCS}_i$ and $\mathcal{MCS}_j$ are in the same final cluster.

In Figure 1, *IMD* ($\mathcal{MC}_2$, $\mathcal{MC}_4$) and *IMD* ($\mathcal{MC}_2$, $\mathcal{MC}_6$) are less than T. However, both microcluster pairs are not linearly adjacent, so they are not grouped into the same microcluster segment in the partition step. In the refinement step, *ERS* conducts *ERQ* for each $\mathcal{MCS}$ to find the *connections* to other $\mathcal{MCS}s$. *ERQ* for $\mathcal{MCS}_2$ checks whether *IMD* ($\mathcal{MC}_2$, $\mathcal{MC}_4$) and *IMD* ($\mathcal{MC}_2$, $\mathcal{MC}_6$) are less than T, and then finds two *connections* $\langle \mathcal{MCS}_2, \mathcal{MCS}_3 \rangle$ and $\langle \mathcal{MCS}_2, \mathcal{MCS}_4 \rangle$, respectively. Subsequently, *ERS* merges $\mathcal{MCS}_2$, $\mathcal{MCS}_3$, and $\mathcal{MCS}_4$ into the final cluster $\mathcal{C}_2$. *ERQs* for other $\mathcal{MCS}s$ (*i.e.*, $\mathcal{MCS}_1$, $\mathcal{MCS}_3$, and $\mathcal{MCS}_4$) find no *connection* between them. Thus, ERC returns the four final clusters $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, and $\mathcal{C}_4$ shown at the bottom of Figure 1. The four final clusters are as follows: $\mathcal{C}_1 = \{\mathcal{MC}_1\}$, $\mathcal{C}_2 = \{\mathcal{MC}_2, \mathcal{MC}_3, \mathcal{MC}_4, \mathcal{MC}_5, \mathcal{MC}_6\}$, $\mathcal{C}_3 = \{\mathcal{MC}_7\}$, and $\mathcal{C}_4 = \{\mathcal{MC}_8, \mathcal{MC}_9, \mathcal{MC}_{10}, \mathcal{MC}_{11}\}$.

B. MapReduce

MapReduce is a programming paradigm for distributed and scalable computation of large data. For example, Facebook employs MapReduce running on Hadoop to deal with a large part of their data-processing applications [22]. Hadoop [23] is a widely used open-source implementation of MapReduce. Figure 2 shows the overall procedure of a MapReduce job. While conducting a MapReduce job, the inputs and outputs are treated as a form of ⟨key, value⟩-pair. The machine performing a map task is called a mapper, while the machine performing a reduce task is called a reducer. The number of mappers and reducers depends on the number of machines in the system and the user configuration.

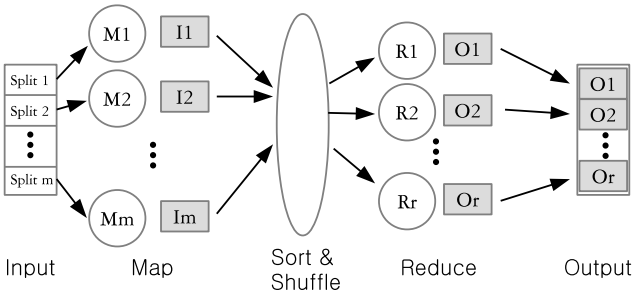


FIGURE 2. Overall implementation of a MapReduce job.

A MapReduce job consists of the five phases: the input, map, sort and shuffle, reduce, and output phases. In the input phase, the input data set is divided into m `splits` and all `splits` are distributed to all the mappers. In the map phase, each mapper receives several `splits`. For each `split`, the map task is performed to generate an intermediate result, which is then utilized in the sort and shuffle phase. In the sort and shuffle phase, the intermediate results are partitioned based on their key and sent to the reducer that manages the key. In the reduce phase, each reducer receives the ⟨key, list of values⟩-pair and performs the reduce task to deal with this pair. Finally, the results of the reduce task are gathered and written in the distributed file system, which is the result of the MapReduce job.

The user only defines the map and reduce tasks to deal with input data set in a parallel way on MapReduce. Meanwhile, the user must consider the characteristics of MapReduce. In particular, since MapReduce exhibits a shared-nothing

architecture, the map and reduce tasks cannot read the data sent to other map and reduce tasks. Thus, all map and reduce tasks can only use their input data to generate the results.

III. CF+ERC_MR: CF+-ERC ON MapReduce

In this section, we discuss the problems encountered while conducting CF+-ERC on MapReduce, which includes building the CF+ tree and performing ERC in a parallel manner, and our solutions for these problems.

A. PROCESS FLOW OF CF+ERC_MR

The proposed method, i.e., CF+ERC_MR, can be broadly divided into three steps: space-partitioning, clustering, and refining. Figure 3 shows the overall process flow of this method. CF+ERC_MR must perform an additional task of merging the results of the reduce tasks to obtain the final result on the refining step at the refinement phase (see Section III-D). In the space-partitioning step, a region centroid set $\mathbb{V}$ for the map tasks is obtained in a sequential manner. $\mathbb{V}$ is used to guide the intermediate result of the map task to its proper reduce task. This is discussed in detail in Section III-B.

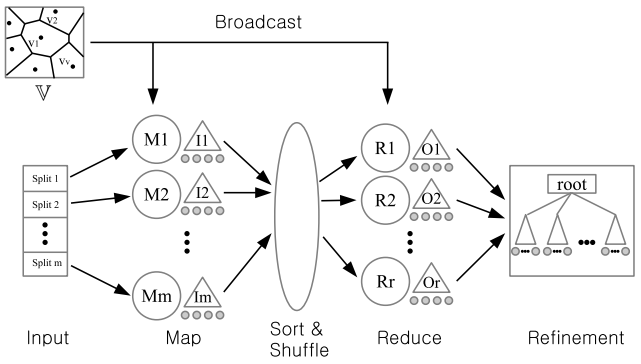


FIGURE 3. Process flow of CF+ERC_MR.

In the clustering step, the local final clusters are determined using MapReduce in parallel. Note that the term ''local final clusters'' refers to the final clusters obtained by ERC in the reduce phase. Each map task receives its corresponding `split` data and $\mathbb{V}$. In each map task, a CF+ tree is first built by using a given threshold value and then a set of microclusters of this tree is determined. All the microclusters are sent to their proper reduce tasks through the sort and shuffle phase in accordance with $\mathbb{V}$. The reduce task receives the ⟨key, (list of values)⟩-pair and then assembles the local final clusters.

For example, the map task `M1` in Figure 3 only reads the ⟨*key*, *value*⟩-pairs of `split` 1 and uses it to build a CF+ tree. A set of microclusters of the CF+ tree is the intermediate result `I1`, which is sent to reduce tasks `R`$i(1 \leq i \leq r)$ through sort and shuffle phase in the form of the $\langle i, \mathcal{MC}\rangle$-pair. The reduce task `R`$i(1 \leq i \leq r)$ receives the $\langle i$, list of $\mathcal{MC}s\rangle$-pair and then builds a CF+ tree using the list of $\mathcal{MC}s$. After completely building the tree, `R`$i(1 \leq i \leq r)$ finds the local final clusters. A set of local final clusters represents the result `O`i, which is discussed in Section III-C.

In the refining step, the local final clusters are sequentially merged into the global final clusters. Note that the term ''global final cluster'' refers to the final clusters merged from the local final clusters at the refinement phase. Because a reduce task finds the local final clusters consisting of the microclusters in that reduce task only, the global final clusters based on those local final clusters must be determined; this is explained in Section III-D.

B. SPACE PARTITIONING STEP OF CF+ERC_MR

All the reduce tasks are simultaneously conducted in isolation in the clustering step. The reducer performing the ''straggler'' reduce task may continue to run even after the other reducers have already finished their reduce tasks. In such a case, MapReduce cannot finish the reduce phase, which is a major factor that contributes to the computation time of a MapReduce job [24]. Thus, it is necessary to send an equal workload to every reduce task.

If the intermediate results of map tasks are randomly and evenly distributed to the reduce tasks, every reduce task receives the intermediate results that are spread in the entire data space. In each reduce task, the local final clusters in the entire data space are determined. This implies that the local final clusters obtained from different reduce tasks overlap with each other. Thus, assembling the global final clusters by combining all the local final clusters can be time-consuming.

However, collecting similar objects corresponding to the same reduce task may be useful for multiple reduce tasks. This is because if the entire data space is divided into exclusive regions and each reduce task oversees microcluster in the respective region, the local final clusters of different reduce tasks do not overlap. In addition, each reduce task can find the local final clusters that might become the global final clusters. This, in turn, leads to a decrease in the workload of the refining step. In other words, the intermediate results of the map tasks should be evenly distributed to every reduce task and the input microclusters of each reduce task should be similar. We now compute the region centroid sets to satisfy these conditions.

A map task cannot read the *split* data sent to other map tasks. The map task must receive the region centroid set $\mathbb{V}$ in advance, which is then used to evenly and similarly distribute the microclusters. The distribution of all the objects should be known for distributing the microclusters to the r reduce tasks. Sampling facilitates the determination of the entire object distribution, so only the samples obtained by applying the reservoir sampling technique [25] are used.

Algorithm 1 describes the space-partitioning function for generating a region centroid set $\mathbb{V}$. The K-means++ method is conducted using the sample set $\mathbb{D}$, where K is set to r. In the K-means++ approach, the centroid of each cluster is considered as a region centroid because it represents the center of that cluster. A set $\mathbb{V}$ of r region centroids is broadcast to all the map and reduce tasks.

Algorithm 1 Space-Partitioning($\mathbb{D}$, r)

INPUT $\mathbb{D}$: A set of samples,
r: Number of reduce tasks
OUTPUT $\mathbb{V}$: A set of region centroids
1: $K \leftarrow r, \mathbb{S} \leftarrow \emptyset, \mathbb{V} \leftarrow \emptyset$
2: $\mathbb{S} \leftarrow K$-means++($\mathbb{D}$)
3: **for** each $C_j \in \mathbb{S}$ **do**
4: Compute the centroid v_j of C_j
5: $\mathbb{V} \leftarrow \mathbb{V} \cup v_j$

C. CLUSTERING STEP OF CF+ERC_MR

The map and reduce tasks are described in this subsection. Algorithm 2 shows the map task of the clustering step. Here, $ED(\mathcal{MC}, v_j)$ is the Euclidean distance between a microcluster $\mathcal{MC}$ and a region centroid v_j. Each map task receives an input split $\mathbb{S}$, a set $\mathbb{V}$ of region centroids, and a given threshold value T of the data set. Note that each map function uses the same threshold value T, which is assumed to be given for data sets of application domains. It first creates a CF$^+$ tree (*tree*) with T. Subsequently, all the objects of $\mathbb{S}$ are inserted into *tree*. Then, for each microcluster $\mathcal{MC}$ of *tree*, $\mathcal{MC}$, whose closest region centroid is v_i, is sent to the reduce task Ri.

Algorithm 2 Map($\mathbb{S}$, $\mathbb{V}$, T)

INPUT $\mathbb{S}$: A *split* of input data,
$\mathbb{V}$: A set of region centroids,
T: Threshold value
OUTPUT i: Reduce task index,
$\mathcal{MC}$: A microcluster
1: Build CF$^+$ tree *tree* using $\mathbb{S}$ and T
2: $\mathbb{M} \leftarrow$ A set of microclusters of *tree*
3: **for** $\mathcal{MC} \in \mathbb{M}$ **do**
4: min $\leftarrow \infty, i \leftarrow 0$
5: **for** $v_j \in \mathbb{V}$ **do** ▷ v_j is a region centroid
6: **if** min $> ED(\mathcal{MC}, v_j)$ **then**
7: min $\leftarrow ED(\mathcal{MC}, v_j), i \leftarrow j$
8: **return** $\langle i, \mathcal{MC}\rangle$-pair

However, occasionally these microclusters do not satisfy the threshold requirement. In Figure 3, the entire input data is divided into m splits, which are processed by the corresponding m map tasks. Figure 4 shows an example of the intermediate results obtained from four map tasks M1, M2, M3, and M4. All the map tasks handle the objects in the same data space, so that the intermediate results may be overlapped. If the distance between the centroids of any two microclusters (i.e., o_1 and o_2 in Figure 4) is less than the threshold value T, they must be merged into the same microcluster.

The validity of the threshold requirement among the microclusters must be rechecked in the reduce task by building a tree. While building the tree, each microcluster $\mathcal{MC}$ is inserted into the tree and then routed to its proper leaf entry e_l in the tree based on the *closest* criteria. If $\mathcal{MC}$ is closer to e_l than T, $\mathcal{MC}$ is absorbed in e_l. Thus, the threshold requirement

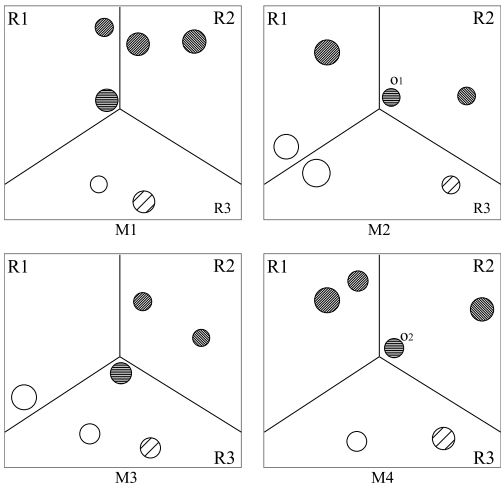


FIGURE 4. Example of intermediate results obtained from four map tasks.

of all the microclusters can be rechecked when the tree is built by using these microclusters. After completely building the tree, the reduce task implements ERC for finding the local final clusters.

Figure 5 illustrates the example of the microclusters of the CF$^+$ trees in three reduce tasks R1, R2, and R3. R1-3 have built their CF$^+$ trees after sending the intermediate results of four map tasks to the all reduce tasks based on the region centroids v_1-v_3. Here, each of the smallest circles represents a microcluster. The pattern of each of the smallest circles indicates the label of the global final clusters ($\mathcal{C}_1$, $\mathcal{C}_2$, $\cdots$, $\mathcal{C}_5$). The microclusters in the same global final cluster are covered by a thick solid circle. A set of microclusters that are connected via the dotted line indicates the local final clusters determined by ERC during the reduce task. The local final cluster is covered by a dashed circle in this figure. Each region separated by solid lines in the data space indicates the local region managed by reduce task such as $\mathcal{R}1$, $\mathcal{R}2$, and $\mathcal{R}3$. For the reduce task $\mathcal{R}1$, four microclusters are generated and three local final clusters are obtained. Thus, there are five

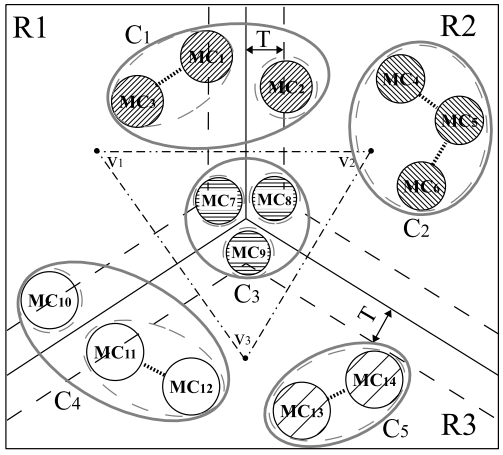


FIGURE 5. Partitioning of the data space.

global final clusters, nine local final clusters, and fourteen microclusters in this figure.

This figure demonstrates why the local final clusters must be refined. Because of dividing the data space based on $\mathbb{V}$, some global final clusters (*i.e.*, $\mathcal{C}_2$ and $\mathcal{C}_5$) are the same as the local final clusters. Contrarily, other global final clusters (*i.e.*, $\mathcal{C}_1$, $\mathcal{C}_3$, and $\mathcal{C}_4$) cannot be established by the reduce tasks which cannot read the microclusters sent to other reduce tasks. Here, the local final clusters consisting of all objects in each region are the subsets of the global final clusters separated by the solid line, as shown in Figure 5. Therefore, the additional *connections* between the local final clusters of the different regions must be obtained to assemble the three global final clusters $\mathcal{C}_1$, $\mathcal{C}_3$, and $\mathcal{C}_4$ after the clustering step.

The border between the regions handled by the two reduce tasks is used as the baseline to discern microclusters that are likely to merge into the same global final cluster. If a microcluster is closer to the border than T, it can be merged with the microclusters in another reduce task. The region that is closer than T from the border is called the border region *BR*. For example, the solid line in Figure 5 represents the border, and the region between the dashed lines including the solid line represents *BR*.

Let $\mathbb{B}$ be a set of microclusters overlapped with *BR*. A scalar projection is used to determine $\mathbb{B}$. Let Ri and Rj be the two reduce tasks. To check whether a microcluster $\mathcal{MC}$ of Ri is included in $\mathbb{B}$, the following approach must be used. Let $v_i, v_j \in \mathbb{V}$ be the two region centroids in the two regions managed by Ri and Rj, respectively. d_c is the sum of the average radius of $\mathcal{MC}$ and the scalar projection of the centroid of $\mathcal{MC}$ onto the line between v_i and v_j, while d_h is the distance between v_i and v_j divided by two. If the difference between d_c and d_h is less than T, $\mathcal{MC}$ is included in $\mathbb{B}$. We name all $\mathcal{MC}s$ in $\mathbb{B}$ as *border microclusters*.

Figure 6 shows an example of the scalar projection of $\mathcal{MC}_1$ onto the line between v_1 and v_2. If the difference between d_c and d_h is less than T, there is a possibility that $\mathcal{MC}_1$ may be connected to one or more microclusters in R2 ($\mathcal{MC}_2$ in this figure). So $\mathcal{MC}_1$ becomes the *border microcluster* and is added to $\mathbb{B}$.

The function *isOverlap* $(i, \mathbb{V}, \mathcal{MC}, T)$ returns true if $\mathcal{MC}$ of Ri is overlapped with *BR*. If it returns true, $\mathcal{MC}$ is added to $\mathbb{B}$. After finding all *border microclusters*, $\mathbb{B}$ is stored into the distributed file system (*DFS*) such as HDFS [26]. Subsequently, $\mathbb{B}$ is rechecked in the refining step to obtain the global final clusters.

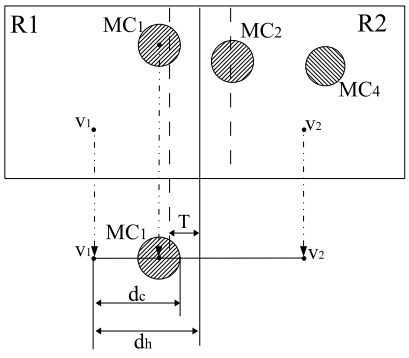


FIGURE 6. An example of the scalar projection of $\mathcal{MC}_1$.

Algorithm 3 shows the reduce task of the clustering step. The reduce task receives the $\langle i, \mathbb{M}\rangle$-pair generated during the shuffle phase, the threshold value T, and a set $\mathbb{V}$ of region centroids. It builds a CF$^+$ tree, called *tree*, using the microclusters of $\mathbb{M}$ to recheck the validity of the threshold requirement. After completely building *tree*, the reduce task Ri performs *ERC*(*tree*, T) for finding the local final cluster set $\mathbb{L}^i$ by utilizing the structure of *tree* in a parallel manner. Assume that $\mathbb{L}^i$ consists of n local final clusters $\{\mathbb{L}^i_1, \mathbb{L}^i_2, \cdots \mathbb{L}^i_n\}$ and each local final cluster $\mathbb{L}^i_l$ is a set of microclusters.

Algorithm 3 Reduce(i, $\mathbb{M}$, $\mathbb{V}$, T)

INPUT i: Reduce task index,
$\mathbb{M}$: A list of microclusters,
$\mathbb{V}$: A set of region centroids,
T: Threshold value

OUTPUT $\mathbb{L}^i$: A set of local final clusters of Ri,
$\mathbb{I}^i$: A set of indices of the local final clusters in $\mathbb{L}^i$,
$\mathbb{B}^i$: A set of *border microclusters* of Ri

1: Build CF$^+$ tree *tree* using $\mathbb{M}$ and T
▷ $\mathbb{L}^i = \{\mathbb{L}^i_1, \mathbb{L}^i_2, \cdots \mathbb{L}^i_n\}$
▷ $\mathbb{L}^i_l$: l-th local final cluster of $\mathbb{L}^i$
2: $\mathbb{L}^i \leftarrow ERC(tree, T)$
▷ $\mathbb{B}^i$: A set of *border microclusters* of Ri
3: $\mathbb{B}^i \leftarrow \emptyset$, $\mathbb{I}^i \leftarrow \emptyset$
4: **for** $\mathbb{L}^i_l \in \mathbb{L}^i$ **do**
5: **for** $\mathcal{MC} \in \mathbb{L}^i_l$ **do** ▷ $\mathcal{MC}$: Microcluster
▷ Check if $\mathcal{MC}$ is overlapped with *BR*
6: **if** *isOverlap* $(i, \mathbb{V}, \mathcal{MC}, T)$ **then**
7: $\mathbb{B}^i \leftarrow \mathbb{B}^i \cup (l, \mathcal{MC})$
8: $\mathbb{I}^i \leftarrow \mathbb{I}^i \cup l$
▷ $\mathbb{B}^i$ and $\mathbb{I}^i$ will be used to obtain global final clusters
9: Store $\mathbb{B}^i$ and $\mathbb{I}^i$ to *DFS*
10: **return** $\mathbb{L}^i$

For each $\mathcal{MC}$ in every $\mathbb{L}^i_l$ in $\mathbb{L}^i$, if $\mathcal{MC}$ is overlapped with *BR*, $(l, \mathcal{MC})$ is added to a *border microcluster* set $\mathbb{B}^i$, where l is the local final cluster index. In the refining step, the local final clusters resulting from different reduce tasks are merged using the *border microclusters* maintaining both the indices of the local final clusters and the reduce tasks. An index set $\mathbb{I}^i$ consists of the indices of the local final clusters of the reduce task Ri. Then, $\mathbb{B}^i$ and $\mathbb{I}^i$ are stored into *DFS*. Finally, $\mathbb{L}^i$ is returned as the results of Ri. $\mathbb{B}^i$ and $\mathbb{I}^i$ are used in the refining step to obtain the global final clusters.

For example, in Figure 5, R1 obtains three local final clusters $\mathbb{L}^1_1 = \{\mathcal{MC}_1, \mathcal{MC}_3\}$, $\mathbb{L}^1_2 = \{\mathcal{MC}_7\}$, and $\mathbb{L}^1_3 = \{\mathcal{MC}_{10}\}$. It also obtains an index set of the local final clusters $\mathbb{I}^1 = \{1, 2, 3\}$. Similarly, R2 and R3 obtain the six local final clusters $\mathbb{L}^2_1 = \{\mathcal{MC}_2\}$, $\mathbb{L}^2_2 = \{\mathcal{MC}_4, \mathcal{MC}_5, \mathcal{MC}_6\}$, $\mathbb{L}^2_3 = \{\mathcal{MC}_8\}$, $\mathbb{L}^3_1 = \{\mathcal{MC}_9\}$, $\mathbb{L}^3_2 = \{\mathcal{MC}_{11}, \mathcal{MC}_{12}\}$,

and $\mathbb{L}_3^3 = \{\mathcal{MC}_{13}, \mathcal{MC}_{14}\}$ as the result of all reduce tasks. They obtain the two index sets of the local final clusters $\mathbb{I}^2 = \{1, 2, 3\}$ and $\mathbb{I}^3 = \{1, 2, 3\}$.

The clustering step also finds the three *border microcluster* sets $\mathbb{B}^1$, $\mathbb{B}^2$, and $\mathbb{B}^3$, including the seven *border microclusters* as follows $\mathbb{B}^1 = \{(1, \mathcal{MC}_1), (2, \mathcal{MC}_7), (3, \mathcal{MC}_{10})\}$, $\mathbb{B}^2 = \{(1, \mathcal{MC}_2), (3, \mathcal{MC}_8)\}$, and $\mathbb{B}^3 = \{(1, \mathcal{MC}_9), (2, \mathcal{MC}_{11})\}$. Finally, $\{\mathbb{B}^1, \mathbb{B}^2, \mathbb{B}^3\}$ and $\{\mathbb{I}^1, \mathbb{I}^2, \mathbb{I}^3\}$ are stored in *DFS*.

D. REFINING STEP OF CF+ERC_MR

All the reduce tasks stored the *border microcluster* sets into *DFS*. The local final cluster sets resulting from all the reduce tasks were written to *DFS*. After the reduce phase of MapReduce, the refining step reads these *border microcluster* sets and local final cluster sets during the refinement phase, as shown in Figure 3. It merges the local final cluster sets into the global final clusters using the *border microcluster* sets.

Let $\mathbb{L}_l^i$ be the l-th local final cluster resulting from the reduce task Ri. Two *border microclusters* $\mathcal{MC}_x \in \mathbb{L}_l^i$ and $\mathcal{MC}_y \in \mathbb{L}_m^j$ are given. If *IMD* $(\mathcal{MC}_x, \mathcal{MC}_y)$ is less than T, $\mathcal{MC}_x$ and $\mathcal{MC}_y$ are *connected* and then two local final clusters $\mathbb{L}_l^i$ and $\mathbb{L}_m^j$ are also *connected*.

Then, the global final clusters are assembled by sequentially combining the connected local final clusters. The *connected* local final clusters can be efficiently obtained by calling ERC if the CF+ tree is composed of the *border microclusters*. However, one *border microcluster* $\mathcal{MC}_x$ can be absorbed into another by the threshold requirement when building the CF+ tree. This indicates that the *connected* local final clusters by using $\mathcal{MC}_x$ cannot be found.

We propose a refining CF+ tree consisting of the *border microclusters* where each *border microcluster* maintains both the reduce task index and the local final cluster index. While building the refining CF+ tree, we do not allow all *border microclusters* to be absorbed into the existing leaf node entries. Thus, each *border microcluster* becomes a new entry of a leaf node in the tree. After building the refining CF+ tree, the execution of ERC over the refining CF+ tree gives a set of *connected* local final clusters. Thus, the *connected* local final clusters are merged into the same global final cluster. In contrast, the local final cluster that is not merged into another local final cluster becomes the global final cluster.

Algorithm 4 is used to determine the global final clusters by using a set $\mathbb{B}$ of *border microclusters*. Here, the list of local final cluster index sets $\mathbb{I} = \{\mathbb{I}^1, \mathbb{I}^2, \cdots, \mathbb{I}^r\}$ and the list of *border microcluster* sets $\mathbb{B} = \{\mathbb{B}^1, \mathbb{B}^2, \cdots, \mathbb{B}^r\}$ are read from *DFS*. The refining CF+ tree *tree* is built by inserting the *border microclusters* of $\mathbb{B}$. The *connection* set of the local final clusters $\mathbb{P} = \{\mathbb{P}_1, \mathbb{P}_2, \cdots \mathbb{P}_p\}$ is obtained by performing ERC in a sequential way.

Since $\mathbb{P}_k (1 \leq k \leq n)$ consists of the *connected border microclusters*, the local final clusters containing the *border microclusters* of $\mathbb{P}_k$ are also *connected*. Thus, these local final clusters are merged into the same global final cluster. Subsequently, the remaining local final clusters are moved into the global final clusters because they are not *connected* to others. Consequently, Algorithm 4 refines a set of local final clusters to obtain the desired global final clusters.

Algorithm 4 Refine(T)

INPUT T: Threshold value
OUTPUT $\mathbb{G}$: A list of global final cluster index sets

```
Read 𝕀 = {𝕀^1, 𝕀^2, ··· , 𝕀^r} from DFS
Read 𝔹 = {𝔹^1, 𝔹^2, ··· , 𝔹^r} from DFS
▷ Build a refining CF+ tree using 𝔹 incrementally
tree ← ∅                                   ▷ tree: Refining the CF+ tree
for 𝔹^i ∈ 𝔹 do
    for (l, MC) ∈ 𝔹^i do                   ▷ MC: Microcluster
        Insert (i, l, MC) into tree
▷ ℙ = {ℙ_1, ℙ_2, ··· ℙ_p}
▷ ℙ_k consists of the connected border microclusters
ℙ ← ERC(tree, T)
List 𝔾 ← ∅
▷ Merge the indices of the connected local final clusters
for ℙ_k ∈ ℙ do
    ▷ Find indices of the local final clusters using ℙ_k
    𝔼 ← ∅
    for (i, l, MC) ∈ ℙ_k do
        𝕀^i ∈ 𝕀
        ▷ l: Index of the l-th local final cluster of Ri
        if l ∈ 𝕀^i then
            ▷ Remove l from 𝕀^i and insert (i, l) to 𝔼
            𝕀^i ← 𝕀^i \ l
            𝔼 ← 𝔼 ∪ (i, l)
    Add 𝔼 to 𝔾
▷ Add the index of remaining local final clusters to 𝔾
for 𝕀^i ∈ 𝕀 do
    for 𝕀^i_l ∈ 𝕀^i do
        Add {(i, l)} to 𝔾
return 𝔾
```

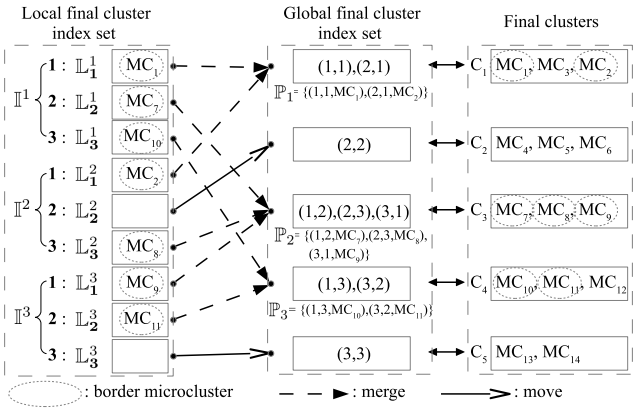


FIGURE 7. Process flow of the refining step using the example in Figure 5.

Figure 7 shows the process flow of the refining step using the fourteen microclusters in Figure 5. In this figure, Algorithm 4 reads the list of the local final cluster index sets $\mathbb{I} = \{\mathbb{I}^1, \mathbb{I}^2, \mathbb{I}^3\}$ and the three *border microcluster* sets

$\mathbb{B}^1 = \{(1, \mathcal{MC}_1), (2, \mathcal{MC}_7), (3, \mathcal{MC}_{10})\}$, $\mathbb{B}^2 = \{(1, \mathcal{MC}_2), (3, \mathcal{MC}_8)\}$, and $\mathbb{B}^3 = \{(1, \mathcal{MC}_9), (2, \mathcal{MC}_{11})\}$. The microcluster covered by the dotted circle in the dashed rectangle of "Local final cluster index set" is referred to as the *border microcluster*. Each local final cluster of every local final cluster set is merged with other local final clusters to assemble the global final cluster (represented by the dashed arrow) or becomes a global final cluster (represented by the solid arrow).

Algorithm 4 then builds a refining CF$^+$ tree by using $\mathbb{B} = \{(1, 1, \mathcal{MC}_1), (1, 2, \mathcal{MC}_7), (1, 3, \mathcal{MC}_{10}), (2, 1, \mathcal{MC}_2), (2, 3, \mathcal{MC}_8), (3, 1, \mathcal{MC}_9), (3, 2, \mathcal{MC}_{11})\}$. Next, ERC finds three *connection* sets $\mathbb{P}_1 = \{(1, 1, \mathcal{MC}_1), (2, 1, \mathcal{MC}_2)\}$, $\mathbb{P}_2 = \{(1, 2, \mathcal{MC}_7), (2, 3, \mathcal{MC}_8), (3, 1, \mathcal{MC}_9)\}$, and $\mathbb{P}_3 = \{(1, 3, \mathcal{MC}_{10}), (3, 2, \mathcal{MC}_{11})\}$ because *IMD* $(\mathcal{MC}_1, \mathcal{MC}_2)$, *IMD* $(\mathcal{MC}_7, \mathcal{MC}_8)$, *IMD* $(\mathcal{MC}_7, \mathcal{MC}_9)$, and *IMD* $(\mathcal{MC}_{10}, \mathcal{MC}_{11})$ are less than T.

By using $\mathbb{P}_1$, (1, 1) and (2, 1) are first merged thus giving $\mathbb{E} = \{(1, 1), (2, 1)\}$, which requires that $\mathbb{I}_1^1 = \{\mathcal{MC}_1, \mathcal{MC}_3\}$ and $\mathbb{I}_1^2 = \{\mathcal{MC}_2\}$ should also be merged into the global final cluster $(\{\mathbb{I}_1^1, \mathbb{I}_1^2\} = \{\mathcal{MC}_1, \mathcal{MC}_3, \mathcal{MC}_2\})$. This global final cluster is the same as $\mathcal{C}_1$. Thus, Algorithm 4 finds the final cluster $\mathcal{C}_1$. Next, $\mathbb{E}$ is added to the global final cluster set $\mathbb{G}$ giving $\mathbb{G} = \{\{(1, 1), (2, 1)\}\}$. Similarly, Algorithm 4 obtains the four final clusters $\mathcal{C}_2 = \{\mathcal{MC}_4, \mathcal{MC}_5, \mathcal{MC}_6\}$, $\mathcal{C}_3 = \{\mathcal{MC}_7, \mathcal{MC}_8, \mathcal{MC}_9\}$, $\mathcal{C}_4 = \{\mathcal{MC}_{10}, \mathcal{MC}_{11}, \mathcal{MC}_{12}\}$, and $\mathcal{C}_5 = \{\mathcal{MC}_{13}, \mathcal{MC}_{14}\}$. Consequently, the refining step returns $\mathbb{G} = \{\{(1, 1), (2, 1)\}, \{(2, 2)\}, \{(1, 2), (2, 3), (3, 1)\}, \{(1, 3), (3, 2)\}, \{(3, 3)\}\}$ that is the same as a set of the five final clusters $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4$, and $\mathcal{C}_5$.

IV. PERFORMANCE ANALYSIS

The effectiveness of the proposed clustering method, CF$^+$ERC_MR, was validated through theoretical and experimental analyses.

A. THEORETICAL ANALYSIS

The time complexities of our proposed clustering method are analyzed. CF$^+$ERC_MR is divided into three steps: space partitioning, clustering, and refining. The clustering step is also divided into two tasks: map and reduce. MapReduce runs on n nodes (one NameNode and $(n-1)$ workers). Since each worker normally performs either the map tasks or the reduce tasks, $(n-1)$ workers consist of μ mappers and ν reducers, thus giving $n = \mu + \nu + 1$.

Table 1 summarizes the symbolic notations frequently used in this section. We discuss the time complexities of the construction CF$^+$ tree and ERC. We assume that each leaf entry of the CF$^+$ tree absorbs a objects, on average. The time complexity of the construction CF$^+$ tree is $O(N \cdot L \cdot d \cdot \log_L \frac{N}{a})$ [5].

Let W be the number of microcluster segments and w be the average number of microclusters resulting from the multiple range queries. The time complexity of ERC is $O(d \cdot (L \cdot \frac{L^h-1}{L-1} + \frac{N}{a} + W \cdot w \cdot L \cdot \log_L \frac{N}{a}) + W \cdot p)$ where h is the height of the CF$^+$ tree and p is the average number of indices in *po* lists [5]. $O(L \cdot d \cdot \frac{L^h-1}{L-1})$ and $O(\frac{N}{a} \cdot d)$ are the time complexities of all radii computations of the CF$^+$ tree and the partition step of ERC, respectively. Since all radii computations and the partition step are performed only once, both are negligible. Further, p is usually very small. Therefore, ERC takes $O(W \cdot w \cdot L \cdot d \cdot \log_L \frac{N}{a})$.

TABLE 1. Summary of symbolic notations.

Notation	Description
N	Number of objects
L	Maximum number of entries in the nodes
d	Dimensionality
m	Number of map tasks
r	Number of reduce tasks
n	Number of nodes
μ	Number of mappers
ν	Number of reducers
α	Average number of absorbed objects for all microclusters in the map task of the clustering step
κ	Number of microcluster segments in the clustering step
λ	Average number of microclusters resulting from the multiple range queries in the clustering step
β	Average number of absorbed objects for all microclusters in the reduce task of the refinement step
ψ	Number of microcluster segments in the refinement step
ω	Average number of microclusters resulting from the multiple range queries in the refinement step
η	Number of border microclusters
δ	Number of indices of local final clusters

The space partitioning step takes $O(s \cdot d \cdot (r + i))$ where s is the number of samples and i is the number of iterations of K-means++. Let $A = O(s \cdot d \cdot (r + i))$.

The map task of the clustering step receives N/m objects, on average. The construction CF$^+$ tree takes $O(\frac{N}{m} \cdot L \cdot d \cdot \log_L \frac{N}{m \cdot \alpha})$. Distributing $N/(m \cdot \alpha)$ microclusters to r reduce tasks takes $O(\frac{N}{m \cdot \alpha} \cdot d \cdot r)$. Let $B = O(\frac{N}{m} \cdot L \cdot d \cdot \log_L \frac{N}{m \cdot \alpha} + \frac{N}{m \cdot \alpha} \cdot d \cdot r)$. Since μ mappers run m map tasks, the time complexity of each mapper is

$$O(\frac{m \cdot B}{\mu}).$$

In the reduce task of the clustering step, since all the map tasks produce N/α microclusters, each reduce task receives $N/(\alpha \cdot r)$ microclusters, on average. The construction CF$^+$ tree takes $O(\frac{N}{\alpha \cdot r} \cdot L \cdot d \cdot \log_L \frac{N}{\alpha \cdot \beta \cdot r})$. ERC takes $O(\kappa \cdot \lambda \cdot L \cdot d \cdot \log_L \frac{N}{\alpha \cdot \beta \cdot r})$. Finding border microclusters takes $O(\frac{N}{\alpha \cdot \beta \cdot r} \cdot d \cdot r^2)$. Let $C = O((\frac{N}{\alpha \cdot r} + \kappa \cdot \lambda) \cdot L \cdot d \cdot \log_L \frac{N}{\alpha \cdot \beta \cdot r} + \frac{N}{\alpha \cdot \beta \cdot r} \cdot d \cdot r^2)$. Because $(n - \mu - 1)$ reducers run r reduce tasks, the time complexity of each reducer is

$$O(\frac{r \cdot C}{n - \mu - 1}).$$

The refinement step reads the border microclusters and the indices of the local final clusters. The construction CF$^+$ tree takes $O(\eta \cdot L \cdot d \cdot \log_L \eta)$. ERC takes $O(\psi \cdot \omega \cdot L \cdot d \cdot \log_L \eta)$. Merging the local final clusters to the global final clusters takes $O(\eta + \delta)$. Let $D = O((\eta + \psi \cdot \omega) \cdot L \cdot d \cdot \log_L \eta + \eta + \delta)$.

Combining the above time complexities, the total time complexity of our proposed clustering method is

$$O(A + \frac{m \cdot B}{\mu} + \frac{r \cdot C}{n - \mu - 1} + D).$$

Note that an increase in the number of reduce tasks expands the border region, which can increase the number of border microclusters. This, in turn, increases the time complexity of the refinement step. Further, the entire data space was finely partitioned for the higher number of reduce tasks. Thus, it is more difficult to find the local final clusters that might become the global final clusters. Therefore, a small value of the number of reduce tasks is sufficient for our proposed method.

B. EXPERIMENTAL SETUP

A diverse range of synthetic data sets and two real data sets were used to compare the performance of our method with that of existing clustering methods including parallel *K*-means (PKMeans) [20], *mrk*-means [21], BIRCH-based distributed clustering methods, and distributed BIRCH.

BIRCH-based distributed clustering methods build a CF tree on the NameNode and then implement distributed clustering using a set of microclusters of the tree. Since PKMeans and *mrk*-means are used as the global clustering method of BIRCH, they are referred to as BIRCH-PKM and BIRCH-MRK, respectively.

Distributed BIRCH is a simple extension of BIRCH to the MapReduce programming paradigm with a single reducer. Every map task locally builds a CF tree and then sends a set of microclusters of the tree to the single reduce task. The reducer uses the input microclusters to build the CF tree. Since this tree covers all the microclusters generated by all the map tasks, an existing partitional clustering method, such as *K*-means, is applied to all the microclusters of the CF tree to obtain the final clusters. The distributed BIRCH whose global clustering method is *K*-means++ [27] is called BIRCH-MR_KM++.

PKMeans, *mrk*-means, and *K*-means++ are also called the extensions of *K*-means because they share the stopping criteria. The maximum number of iterations for these methods is set to 50 except for PKMeans for which it is set to 10. The reason for this selection is discussed in the later subsection.

The split size for *mrk*-means is set to $\sqrt{K \cdot n}$ proposed in [21], where n is the number of objects. The number of true clusters in our synthetic and real data sets is provided to the *K*-means extensions. Note that CF$^+$ERC_MR does not require the number of final clusters for clustering. In addition, the same threshold value is used to build the CF and CF$^+$ trees. The proper threshold values were obtained for all synthetic and real data sets by conducting the experiments with various threshold values.

The average purity, inverse purity, and execution time of these clustering methods are used as the primary performance metrics. They were obtained by repeating the experiment five times with different seeds on the same data set. The purity and inverse purity metrics are formally described in [28]. The purity of the clustering method evaluates the frequency of the most similar objects of each cluster, while the inverse purity of the clustering method evaluates similar objects placed in the same cluster. The execution times of CF$^+$ERC_MR and the other methods include the time required to build the CF$^+$ and CF trees.

All the experiments were conducted on Amazon EMR. The MapReduce system consisted of ten nodes (one NameNode and nine workers). All the nodes used the Intel Xeon Platinum 8000 series (Skylake-SP) processor with a sustained all core Turbo CPU having a clock speed of up to 3.1 GHz. The NameNode used four cores with two threads, 64 GB memory, and EBS storage. The workers used two cores with two threads, 32 GB memory, and EBS storage. The basic size of EBS storage was 1 GiB, which could be extended up to 16 TiB if a task required additional storage. In addition, we set $C = 50, L = 50$ for the CF$^+$ and CF trees, which are the same parameter values used in [5]. We set the number of reduce tasks, r, to 4 because we experimentally found out that a small value of r (*i.e.*, 4) was good enough for our proposed method.

C. SYNTHETIC DATA SETS

A collection of synthetic data sets was used in our experiments. These data sets were obtained by employing a synthetic data generator based on the parameters used in [6]. Table 2 shows the parameters used in our generator.

TABLE 2. Parameters used to generate the synthetic data sets.

Parameter	Values or Range
Cluster patterns	grid, random, sine
Number of clusters: K	100 . . . 1,600
Minimum radius range of cluster: r_{min}	0 or $\sqrt{2}$
Maximum radius range of cluster: r_{max}	$\sqrt{2}$ or 4
Minimum number of data in a cluster: n_{min}	24,000 . . . 120,000
Maximum number of data in a cluster: n_{max}	56,000 . . . 280,000
Distance multiplier (grid only): k_g	4
Number of cycles (sine only): n_c	4
Dimensionality: d	2, 4, . . . 10

All the synthetic data sets consist of K clusters with d-dimensional data. Each cluster in the data set has n_{min} to n_{max} objects within a radius of r_{min} to r_{max}. The centroids of all the clusters are placed in the data space according to the cluster patterns in Table 2. All centroids of the *grid pattern* data set are placed on a $\sqrt{K} \times \sqrt{K}$ grid, where the distance between the centroids of adjacent clusters is $k_g \frac{r_{min}+r_{max}}{2}$. All the centroids of the *random pattern* data set are randomly

placed in the data space. Further, all the centroids of the *sine pattern* data set are partitioned into n_c groups, and then the centroids of each group are placed at the curve of a different cycle of the n_c sine function.

The three default data sets used in our experiments were generated by the synthetic data generator with the parameter values shown in Table 3. These data sets are named with respect to the cluster pattern, *i.e.*, grid pattern cluster (*GPC*), random pattern cluster (*RPC*), and sine pattern cluster (*SPC*). All the data sets consisted of approximately $4,000,000$ objects each.

TABLE 3. Default data sets for experiments.

Data set	Pattern	K	r_{min}	r_{max}	n_{min}	n_{max}	d
GPC	grid	100	$\sqrt{2}$	$\sqrt{2}$	40,000	40,000	2
RPC	random	100	0	4	24,000	56,000	2
SPC	sine	100	$\sqrt{2}$	$\sqrt{2}$	40,000	40,000	2

D. PARAMETRIC ANALYSIS OF CF+ERC_MR

In this subsection, we analyze the performance of CF+ERC_MR in terms of the sample size, speedup, and workload balance of multiple reduce tasks. These analyses reveal that CF+ERC_MR performs exceptionally well in a distributed computing environment.

1) EFFECT OF THE SAMPLE SIZE

The effect of the sample size on the performance of CF+ERC_MR using *GPC* and *RPC* was analyzed, while the number of clusters K was set to 400. Each experiment was repeated five times, while the sample size was increased from 100 to 6,400. The average execution time and its standard deviation obtained from all experiments indicated the effect of the sample size.

Figure 8 shows the average execution time and its standard deviation as a function of the sample size. The purity and inverse purity were not affected by the variation in the sample size; therefore, they are not shown in this figure. A larger sample size increased the time consumed in the space-partitioning step because more objects were gathered into the NameNode from the workers on the MapReduce framework. However, CF+ERC_MR with a large sample size was occasionally faster than that with a small sample size. For example, in Figure 8a, CF+ERC_MR with 800 samples is faster than that with smaller sample size. This is because a small number of samples does not capture the overall data distribution accurately, which can increase the time consumed in the clustering and refining steps.

CF+ERC_MR exhibits a stable execution time and low standard deviation without strong fluctuations, despite the randomness of the reservoir sampling and K-means++ in the space-partitioning step. CF+ERC_MR with 200 samples is relatively faster than the other methods when *GPC* and *RPC* are considered. This experiment also indicates that a small sample size is sufficient to obtain a good region centroid

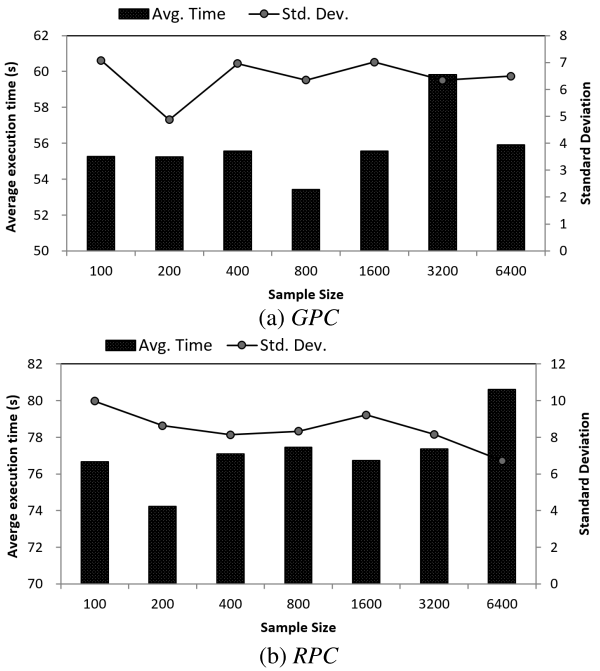


FIGURE 8. Effect of the sample size on the performance of CF+ERC_MR.

set $\mathbb{V}$. Thus, the sample size was set to 200 in all the subsequent experiments.

2) SPEEDUP OF CF+ERC_MR

Experiments were conducted regarding the speedup of CF+ERC_MR by increasing the number of nodes on the MapReduce framework using three GPC data sets: G32M, G64M, and G128M. G32M is GPC with 800 clusters. It has 32 million objects because the number of clusters is 800 and the number of objects in each cluster is 40,000. Similarly, G64M has 64 million objects and G128M has 128 million objects.

The experiments were conducted using the MapReduce framework in which the number of nodes was increased from 2 to 12 for each data set.

Figure 9 shows the speedup of CF+ERC_MR. CF+ERC_MR with an increase in the number of nodes performed more rapidly because the clustering step was performed on all nodes in a distributed manner. All n nodes of the MapReduce

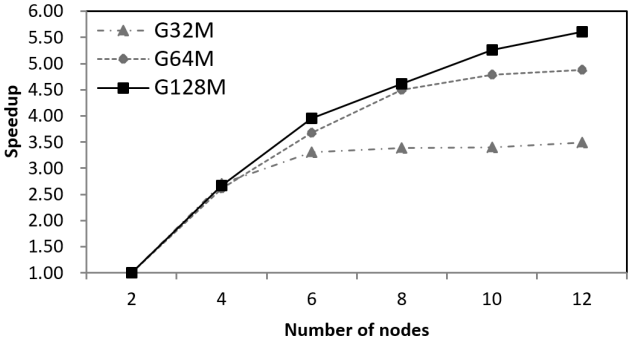


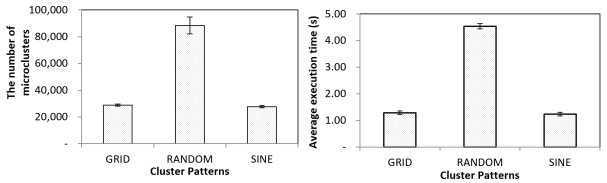
FIGURE 9. Speedup of CF+ERC_MR on GPC.

system were composed of one NameNode, μ mappers, and ν reducers, thus giving $n = \mu + \nu + 1$. The mapper takes $O(\frac{m \cdot A}{\mu})$ and the reducer takes $O(\frac{r \cdot B}{n-\mu-1})$. Thus, the speedup of CF+ERC_MR is increased as n increases.

However, in this figure, it is clear that the speedup for all the data sets does not linearly increase with the increase in the number of nodes. The network cost of the method was analogous for all the experiments on the same data set, but an increase in the number of nodes reduced the workload per node. Since the network cost portion of the total cost is higher for a higher number of nodes, the speedup difference decreases with the increase in the number of nodes.

The speedup of the method on larger data set is higher than that on smaller data set because the workload of larger data set is more than that of the smaller data set. From Figure 9, we can also infer that every reduce task deals with similar-sized workload. If each reduce task deals with workloads of different sizes, the speedup is dependent on the performance of any straggler node, which can cause fluctuations in the speedup. Figure 9 shows no fluctuations, which implies that CF+ERC_MR deals with the data set in an evenly distributed manner. We now need to show that the reduce tasks of the method deal with data sets in an evenly distributed manner.

Figure 10 shows the workload balance of all the reduce tasks obtained by CF+ERC_MR on the three data sets: *GPC*, *RPC*, and *SPC*. Here, the error bars represent the standard deviation of the average size of microclusters (Figure 10a) and average execution time (Figure 10b). The heights of all the error bars are small compared to the average value, so the workloads of all the reduce tasks are even. Therefore, CF+ERC_MR exhibits a good speedup performance because it avoids the straggler reducer, which is considered to be one of the major factors that degrade the performance of MapReduce jobs.



(a) Average number of microclusters sent to the reduce task (b) Average execution time of the reduce task

FIGURE 10. Workload balance of CF+ERC_MR.

Next, by using three GPC data sets, we compare the execution times of CF+ERC_MR having two nodes and CF+-ERC. We conducted this experiment, since Figure 9 shows the speedup of CF+ERC_MR with respect to two nodes. Note that CF+ERC_MR requires at least two nodes to perform, while CF+-ERC is a sequential clustering method. Through this experiment, we show how CF+ERC_MR with two nodes performs over a sequential clustering method CF+-ERC.

Table 4 shows the clustering times of CF+-ERC and CF+ERC_MR with two nodes. The purity and inverse purity are omitted because they are the same in both methods.

TABLE 4. Clustering time of CF+ERC_MR and CF+-ERC.

Method	G32M	G64M	G128M
CF+-ERC	271 s	549 s	1,302 s
CF+ERC_MR with two nodes	271 s	575 s	1,338 s

CF+-ERC is shown to be faster than CF+ERC_MR for all the experiments summarized in this table. This is because CF+ERC_MR included extra steps, such as space-partitioning and refining steps, and performed additional tasks for running a MapReduce job. In view of CF+ERC_MR with these overheads, the difference between the clustering times is negligible. In addition, CF+ERC_MR with more nodes can reduce the clustering time, as shown in Figure 9.

E. EXPERIMENTAL ANALYSIS WITH SYNTHETIC DATA SETS

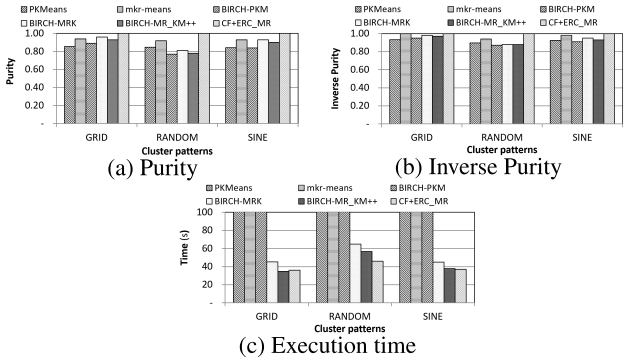
Experiments on the synthetic data sets were performed to compare the effects of the cluster patterns, number of clusters K, data set size n, and the number of dimensions d on the performance of CF+ERC_MR, PKMeans, *mrk*-means, BIRCH-based distributed clustering methods, and distributed BIRCH.

1) EFFECT OF THE CLUSTER PATTERN OF DATA SETS

The effect of cluster patterns on the performance of CF+ERC_MR and other clustering methods using the default data sets (*GPC*, *RPC*, and *SPC*) in Table 3 was analyzed.

Figure 11 shows that CF+ERC_MR exhibits the best performance in terms of average purity and inverse purity for all the cluster patterns in the data sets. Further, CF+ERC_MR is faster than other methods in most cases. It may be noted that CF+ERC_MR performs additional work in the sampling step. Only 200 objects need to be read, so the effect of additional work on the total execution time is negligible. For all the cluster patterns, the average purity and inverse purity of CF+ERC_MR are always better than those of other clustering methods. This indicates that CF+ERC_MR is suitable for the data set whose threshold value is given by the user.

Figure 11c shows the execution times of all the clustering methods. The maximum value of the vertical axis is intentionally restricted to 100 s because the execution time



(a) Purity (b) Inverse Purity

(c) Execution time

FIGURE 11. Effect of cluster patterns on the performance of different methods.

of some methods was much higher than 100 s. For example, PKMeans consumed 309 s for *GPC*, 316 s for *RPC*, and 298 s for *SPC*.

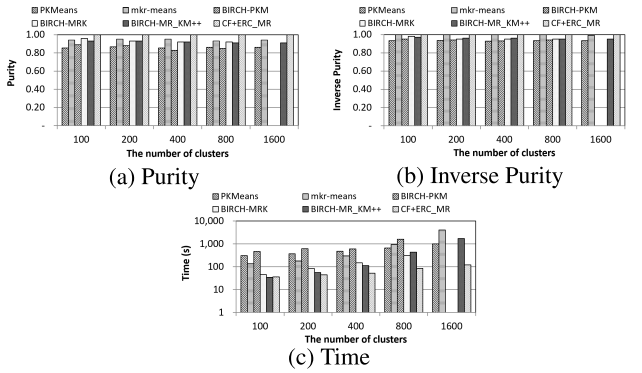
Because the maximum number of iterations in PKMeans is less than that in BIRCH-PKM, PKMeans is faster than BIRCH-PKM. PKMeans is considerably slower than the other methods because it repeatedly runs MapReduce jobs in each iteration. A MapReduce job requires additional input/outputs (I/Os) and tasks for distributed computing. Although the computational complexity of *K*-means is reduced in PKMeans because the distance computations and centroid update step are parallelly implemented in PKMeans, clustering is slowed down due to the overhead of several MapReduce jobs. It consumed a longer time than CF$^+$ERC_MR while its purity and inverse purity were less than those of CF$^+$ERC_MR in Figure 11c. When increasing the maximum number of iterations from 10 to 50, PKMeans consumed much longer time (approximately 1,400 s), but its purity and inverse purity were not only significantly increased but also always less than those of CF$^+$ERC_MR.

For analyzing the performance of CF$^+$ERC_MR and PKMeans within the comparable difference of execution times, the maximum number of iterations in PKMeans was set to 10 in all experiments. The performance of the clustering methods based on *SPC* is excluded in the subsequent experiments because it lies between the performances of the methods based on *GPC* and *RPC*.

2) EFFECT OF THE NUMBER OF CLUSTERS

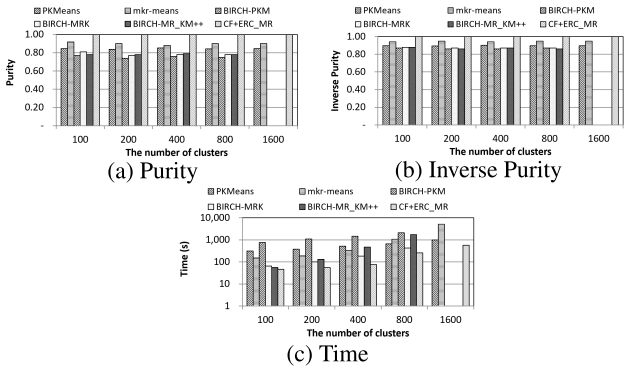
The effect of the increase in the number of clusters on the performance of CF$^+$ERC_MR and other methods was investigated by increasing the size of the data sets, while maintaining the data densities of the clusters. For this experiment, *GPC* and *RPC* with K ranging from 100 to 1,600 (Table 3) were used.

The results are shown in Figures 12 and 13. BIRCH-PKM and BIRCH-MRK are not shown in this figure because they were not able to cluster the data sets with $K = 1,600$ due to the shortage of main memory. The BIRCH-based distributed clustering methods build the CF tree on the NameNode, so they usually require a lot of memory, which indicates that they cannot cluster an exceptionally large data set.



(a) Purity (b) Inverse Purity (c) Time

FIGURE 12. Effect of the number of clusters (*K*) on *GPC*.



(a) Purity (b) Inverse Purity (c) Time

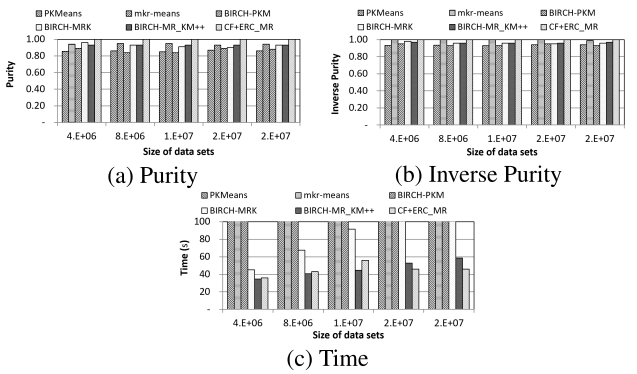
FIGURE 13. Effect of the number of clusters (*K*) on *RPC*.

The purity and inverse purity of CF$^+$ERC_MR are always higher than those of other clustering methods. CF$^+$ERC_MR is extremely faster than PKMeans and *mrk*-means, whereas it is slightly faster than BIRCH-based distributed clustering methods and distributed BIRCH. However, as the number of clusters increases, the difference in the execution time between CF$^+$ERC_MR and other clustering methods using microclusters also increases. This indicates that for exceptionally large data sets, ERC that utilizes the structure of the CF$^+$ tree by using range queries is more suitable than other clustering methods. An increase in the number of clusters increases the number of microclusters, which in turn increases the number of distance computations for clustering. Since ERC avoids unnecessary distance computations, the clustering time for several microclusters is reduced.

3) EFFECT OF THE SIZE OF DATA SETS

The effect of the increase in the size of data sets on the performance of CF$^+$ERC_MR and other methods was experimentally analyzed by using *GPC* and *RPC* with an increase in n_{min} and n_{max} (Table 3). The size of the data sets was approximately increased from 4 M (4×10^6) to 20 M (20×10^6). With K fixed to 100, increasing the size of the data sets causes increasing the data density of all clusters.

Figures 14 and 15 show that an increase in the data set size rarely affected the purities and inverse purities of the clustering methods. However, the execution times of the clustering



(a) Purity (b) Inverse Purity (c) Time

FIGURE 14. Effect of the size of clusters on *GPC*.

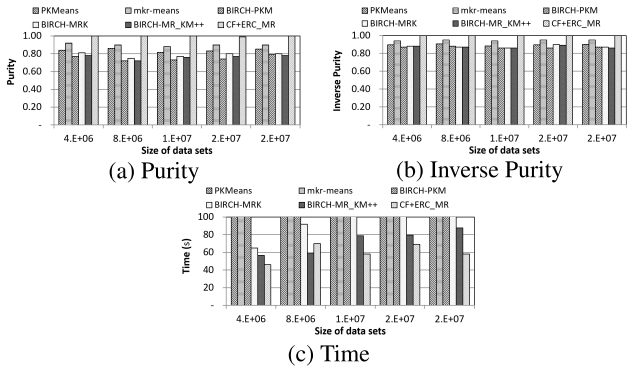


FIGURE 15. Effect of the size of clusters on *RPC*.

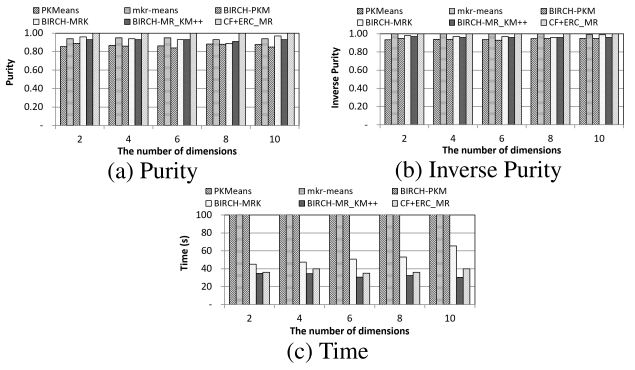


FIGURE 16. Effect of the dimensionality *d* on *GPC*.

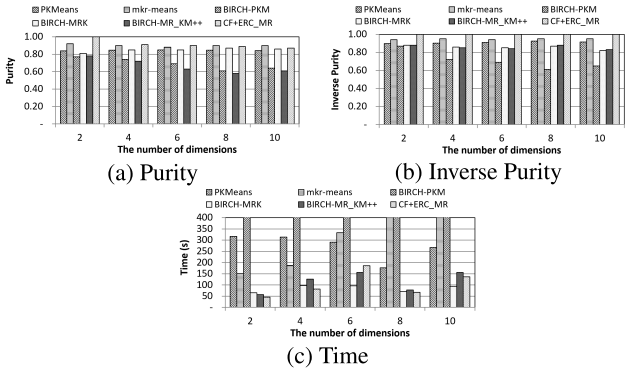


FIGURE 17. Effect of the dimensionality *d* on *RPC*.

methods were reduced when increasing the size of the data sets in Figures 14c and 15c. The reason for this is that more objects required many more distance computations.

The performance of CF+ERC_MR is the best if all the primary performance metrics are considered. However, it is slightly slower than BIRCH-MR_KM++ for smaller data sets. This is because a small data set generates small-sized summary data, but a large data set generates large-sized summary data, which increases the global clustering time. BIRCH-MR_KM++ that performs global clustering on the NameNode is inefficient in clustering exceptionally large data sets. Therefore, CF+ERC_MR clusters a large data set in a much shorter time than BIRCH-MR_KM++.

4) EFFECT OF THE DATA DIMENSIONALITY

In this subsection, the effect of the dimensionality of the data set on the performance of CF+ERC_MR and other methods is analyzed. *GPC* and *RPC* with *d* increasing from 2 to 10 (Table 3) are used in this experiment.

It is clear in Figures 16 and 17 that CF+ERC_MR exhibits a high performance in all data sets even if the dimensionality increases. An increase in the number of dimensions reduced the standard deviation of distances of all object pairs due to the "curse of dimensionality." This deteriorated the purity and inverse purity of *K*-means extensions because similar objects were assigned to the different final clusters except for *mrk*-means and BIRCH-MRK. Contrarily, the object pair within *T* was assigned to the same final cluster in CF+ERC_MR that had used the threshold value as the clustering criteria.

mrk-means was much slower than CF+ERC_MR, and its purity and inverse purity were lower than those of CF+ERC_MR. The reasons were that CF+ERC_MR handled microclusters and used the threshold value as the global clustering criteria. Figure 17b shows that the inverse purities of BIRCH-PKM and BIRCH-MRK decrease because BIRCH does not obtain appropriate microclusters when the number of dimensions is increased. Thus, we can infer that CF+ERC_MR is more suitable for clustering high-dimensional data sets than BIRCH-MRK which was often comparable to CF+ERC_MR regarding execution time.

F. PERFORMANCE ANALYSIS WITH REAL DATA SETS

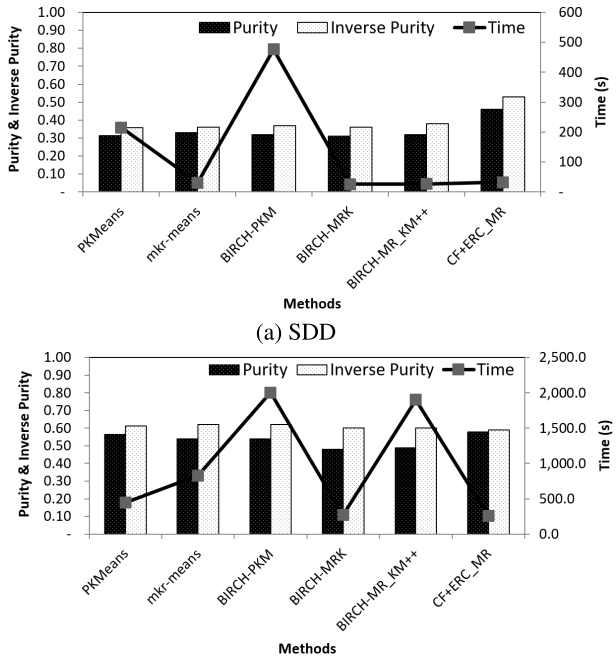
In this subsection, we compare the performance of CF+ERC_MR with that of PKMeans, *mrk*-means, BIRCH-based distributed clustering, and distributed BIRCH using real data sets. The Sensorless Drive Diagnosis data set (*SDD*) and the Amsterdam Library of Object Images data set (*ALOI*) were used for this experiment. *SDD* and *ALOI* were provided by UCI Machine Learning Repository[1] and OpenML,[2] respectively. Kobren *et al.* [29] used the *ALOI* data set to evaluate the *K*-means extensions and hierarchical clustering methods. In this study, WEKA,[3] a popular data mining application, was used to normalize the data. The parameters used are shown in Table 5.

TABLE 5. Parameters for the real data sets.

Data set	*d*	*K*	*n*
SDD	48	11	58,509
ALOI	128	1000	108,000

Figure 18a shows the purity, inverse purity, and execution time of all the methods used in the experiment. For the *SDD* data set, CF+ERC_MR outperforms the other methods in terms of purity and inverse purity. BIRCH-PKM is noticeably slower than the other methods because it runs a MapReduce

[1] https://archive.ics.uci.edu/ml/
[2] https://www.openml.org/
[3] https://www.cs.waikato.ac.nz/ml/weka/



(a) SDD

(b) ALOI

FIGURE 18. Performance of CF+ERC_MR on real data sets.

job repeatedly. The clustering times of CF+ERC_MR and BIRCH-MRK are similar, but the purity and inverse purity of BIRCH-MRK are lower than those of CF+ERC_MR. This is because BIRCH-MRK cannot use the threshold value in the global clustering method. For the *ALOI* data set, the performance of CF+ERC_MR is much better than that of the other methods. The clustering time of BIRCH-MR_KM++ exhibits a much stronger increase for the experiment conducted using the *SDD* data set as compared to that using the *ALOI* data set. This confirms that global clustering on a single machine is not suitable for clustering exceptionally large data sets, even though it deals with the summary data such as microclusters.

Overall, the experiments based on real data sets indicate that CF+ERC_MR provides a better performance than PKMeans, *mrk*-means, BIRCH-based distributed clustering methods, and distributed BIRCH. Consequently, CF+ERC_MR is more suitable for clustering real data sets.

V. CONCLUSIONS

Summary-based clustering and distributed clustering are a vital component of effective data analysis techniques for exceptionally large data sets. Here, we proposed CF+ERC_MR, which is an extension of CF+-ERC to the MapReduce framework, for clustering exceptionally large data sets with given thresholds. Our method was run parallelly on multiple reducers. The CF+ trees were built on multiple reducers, and multiple range queries were called to find the local final clusters by utilizing the structures of these CF+ trees in a parallel manner. In the refining step, the *connections* between these local final clusters were examined to sequentially determine the global final clusters. Consequently, the CF+ERC_MR method efficiently yielded the final clusters based on MapReduce.

The theoretical analysis of CF+ERC_MR was first discussed. The efficiency of CF+ERC_MR was then demonstrated by comparing its performance with that of the existing clustering methods for large synthetic and real data sets. A variety of experiments indicated that the proposed approach was significantly faster and more accurate than existing clustering methods. It also showed the robustness of our approach to various patterns, the number of clusters, the density of clusters, and the number of dimensions. Overall, our method is extremely useful for clustering exceptionally large data sets with given threshold values. In the future, we aim to extend the proposed clustering method to other types of data sets, such as categorical or mixed-type data sets.

REFERENCES

[1] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 3, pp. 267–279, Sep. 2014.

[2] A. A. Abbasi and M. Younis, "A survey on clustering algorithms for wireless sensor networks," *Comput. Commun.*, vol. 30, nos. 14–15, pp. 2826–2841, Oct. 2007.

[3] C. C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," in *Mining Text Data*. Boston, MA, USA: Springer, 2012, pp. 77–128, doi: 10.1007/978-1-4614-3223-4_4.

[4] J. Brank, M. Grobelnik, and D. Mladenic, "A survey of ontology evaluation techniques," in *Proc. Conf. Data Mining Data Warehouses (SiKDD)*, Ljubljana, Slovenia, 2005, pp. 166–170.

[5] H.-C. Ryu, S. Jung, and S. Pramanik, "An effective clustering method over CF+ tree using multiple range queries," *IEEE Trans. Knowl. Data Eng.*, early access, Apr. 15, 2020, doi: 10.1109/TKDE.2019.2911520.

[6] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," in *Proc. Int. Conf. Manage. Data (ACM SIGMOD)*, Montreal, QC, Canada, 1996, pp. 103–114.

[7] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: A new data clustering algorithm and its applications," *Data Mining Knowl. Discovery*, vol. 1, no. 2, pp. 141–182, Jun. 1997.

[8] F. Mariantonietta, S. Alessia, C. Francesco, and P. Giustina, "GHG and cattle farming: CO-assessing the emissions and economic performances in italy," *J. Cleaner Prod.*, vol. 172, pp. 3704–3712, Jan. 2018.

[9] F. Rayar, S. Barrat, F. Bouali, and G. Venturini, "A viewable indexing structure for the interactive exploration of dynamic and large image collections," *ACM Trans. Knowl. Discovery Data*, vol. 12, no. 1, pp. 1–26, Feb. 2018.

[10] J. Zhang, R. Perdisci, W. Lee, X. Luo, and U. Sarfraz, "Building a scalable system for stealthy P2P-botnet detection," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 1, pp. 27–38, Jan. 2014.

[11] J. F. M. Trinidad, J. R. Shulcloper, and M. S. L. Cortés, "Structuralization of universes," *Fuzzy Sets Syst.*, vol. 112, no. 3, pp. 485–500, Jun. 2000.

[12] A. Krause, J. Stoye, and M. Vingron, "Large scale hierarchical clustering of protein sequences," *BMC Bioinformatics*, vol. 6, p. 15, Jan. 2005.

[13] K. Elgazzar, A. E. Hassan, and P. Martin, "Clustering WSDL documents to bootstrap the discovery of Web services," in *Proc. IEEE Int. Conf. Web Services*, Miami, FL, USA, Jul. 2010, pp. 147–154.

[14] M. Mittal, V. P. Singh, and R. K. Sharma, "Random automatic detection of clusters," in *Proc. Int. Conf. Image Inf. Process.*, Shimla, India, Nov. 2011, pp. 1–6.

[15] S. Dutta and T. J. Overbye, "A clustering based wind farm collector system cable layout design," in *Proc. IEEE Power Energy Conf. Illinois*, Champaign, IL, USA, Feb. 2011, pp. 1–6.

[16] A. O. de Carvalho Filho, W. B. de Sampaio, A. C. Silva, A. C. de Paiva, R. A. Nunes, and M. Gattass, "Automatic detection of solitary lung nodules using quality threshold clustering, genetic algorithm and diversity index," *Artif. Intell. Med.*, vol. 60, no. 3, pp. 165–177, Mar. 2014.

[17] S. M. B. Netto, J. O. Bandeira Diniz, A. C. Silva, A. C. de Paiva, R. A. Nunes, and M. Gattass, "Modified quality threshold clustering for temporal analysis and classification of lung lesions," *IEEE Trans. Image Process.*, vol. 28, no. 4, pp. 1813–1823, Apr. 2019.
[18] J. M. Kleinberg, "An impossibility theorem for clustering," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, 2003, pp. 463–470.
[19] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[20] W. Zhao, H. Ma, and Q. He, "Parallel K-means clustering based on MapReduce," in *Proc. IEEE Int. Conf. Cloud Comput.*, vol. 5931, Beijing, China, Dec. 2009, pp. 674–679.
[21] S. Shahrivari and S. Jalili, "Single-pass and linear-time K-means clustering based on MapReduce," *Inf. Syst.*, vol. 60, pp. 1–12, Aug. 2016.
[22] R. Grover and M. J. Carey, "Extending map-reduce for efficient predicate-based sampling," in *Proc. IEEE 28th Int. Conf. Data Eng.*, Washington, DC, USA, Apr. 2012, pp. 486–497.
[23] T. White, *Hadoop: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly, 2012.
[24] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. 8th USENIX Symp. Operating Syst. Design Implement.*, San Diego, CA, USA, 2008, pp. 29–42.
[25] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, Mar. 1985.
[26] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, Incline Village, NV, USA, 2010, pp. 1–10.
[27] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proc. 8th Annu. ACM-SIAM Symp. Discrete Algorithms*, New Orleans, LA, USA, 2007, pp. 1027–1035.
[28] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo, "A comparison of extrinsic clustering evaluation metrics based on formal constraints," *Inf. Retr.*, vol. 12, no. 4, pp. 461–486, Aug. 2009.
[29] A. Kobren, N. Monath, A. Krishnamurthy, and A. McCallum, "A hierarchical algorithm for extreme clustering," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Halifax, NS, Canada, Aug. 2017, pp. 255–264.

HYEONG-CHEOL RYU received the B.S. degree in computer science from Hanshin University, South Korea, in 2009, and the M.S. and Ph.D. degrees in computer science and engineering from Sogang University, South Korea, in 2014 and 2020, respectively. His research interests include spatial databases and data mining.

SUNGWON JUNG (Member, IEEE) received the B.S. degree in computer science from Sogang University, Seoul, South Korea, in 1988, and the M.S. and Ph.D. degrees in computer science from Michigan State University, East Lansing, MI, USA, in 1990 and 1995, respectively. He is currently a Professor with the Computer Science and Engineering Department, Sogang University. His research interests include data mining, spatial and mobile databases, blockchain databases, and multimedia databases.