Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2015, Article ID 793010, 18 pages
http://dx.doi.org/10.1155/2015/793010

Research Article

An Efficient MapReduce-Based Parallel Clustering Algorithm for Distributed Traffic Subarea Division

Dawen Xia,[1,2] Binfeng Wang,[1] Yantao Li,[1] Zhuobo Rong,[1] and Zili Zhang[1,3]

[1]*School of Computer and Information Science, Southwest University, Chongqing 400715, China*
[2]*School of Information Engineering, Guizhou Minzu University, Guiyang 550025, China*
[3]*School of Information Technology, Deakin University, Waurn Ponds, VIC 3216, Australia*

Correspondence should be addressed to Zili Zhang; zhangzl@swu.edu.cn

Received 21 April 2015; Revised 12 July 2015; Accepted 13 August 2015

Academic Editor: Hubertus Von Bremen

Traffic subarea division is vital for traffic system management and traffic network analysis in intelligent transportation systems (ITSs). Since existing methods may not be suitable for big traffic data processing, this paper presents a MapReduce-based Parallel Three-Phase K-Means (Par3PKM) algorithm for solving traffic subarea division problem on a widely adopted Hadoop distributed computing platform. Specifically, we first modify the distance metric and initialization strategy of K-Means and then employ a MapReduce paradigm to redesign the optimized K-Means algorithm for parallel clustering of large-scale taxi trajectories. Moreover, we propose a boundary identifying method to connect the borders of clustering results for each cluster. Finally, we divide traffic subarea of Beijing based on real-world trajectory data sets generated by 12,000 taxis in a period of one month using the proposed approach. Experimental evaluation results indicate that when compared with K-Means, Par2PK-Means, and ParCLARA, Par3PKM achieves higher efficiency, more accuracy, and better scalability and can effectively divide traffic subarea with big taxi trajectory data.

1. Introduction

With the increasingly rapid economic globalization and urbanization, traffic congestion becomes a critical problem and causes great concern among people and the governments in metropolises [1, 2]. To alleviate traffic congestion, a large amount of money was spent on traffic planning and management, especially in traffic subarea division as this is crucial for ITSs. Traffic subarea division is to divide the whole traffic area into several different subareas in order to build a multiarea hierarchical control system, based on the similarity and correlation of traffic pattern features. Traffic subarea division is an effective method when managing traffic control systems. This is because a complete urban traffic system is usually large and complicated, and it is difficult to analyze some traffic problems as a whole. Dividing a whole urban traffic system into different traffic subareas and then studying each traffic subarea system can reduce the complexity of traffic network analysis. Therefore, traffic subarea division is also a powerful tool to analyze complex traffic networks. More importantly, traffic subarea division can provide decision-making for traffic planning and management.

Recent years saw the big data era [3] for transportation coming. Massive traffic data have been growing rapidly with 5Vs characteristics (i.e., Volume, Velocity, Variety, Value, and Veracity), thereby attracting great attention among people from all walks of life (e.g., industries [4], academia [5, 6], governments [7], and other organizations [8, 9]). In particular, taxicabs are equipped with GPS sensors for dispatching and safety in most modern cities; thus, a large number of GPS trajectories of taxicabs with their present location, geoposition, time stamp, and occupancy information are being generated to a data center in a certain frequency every day [10, 11]. For instance, based on a Hadoop platform with ArcGIS, we employ large-scale taxi trajectories used in this work, to produce the road network of Beijing (as illustrated in Figure 1) essentially in agreement with the real traffic map. Figure 1 shows the density distribution of the GPS points (1,232,048 records) generated by 12,000 taxicabs in Beijing during one hour (00:14:35–01:14:34) on November 1, 2012.

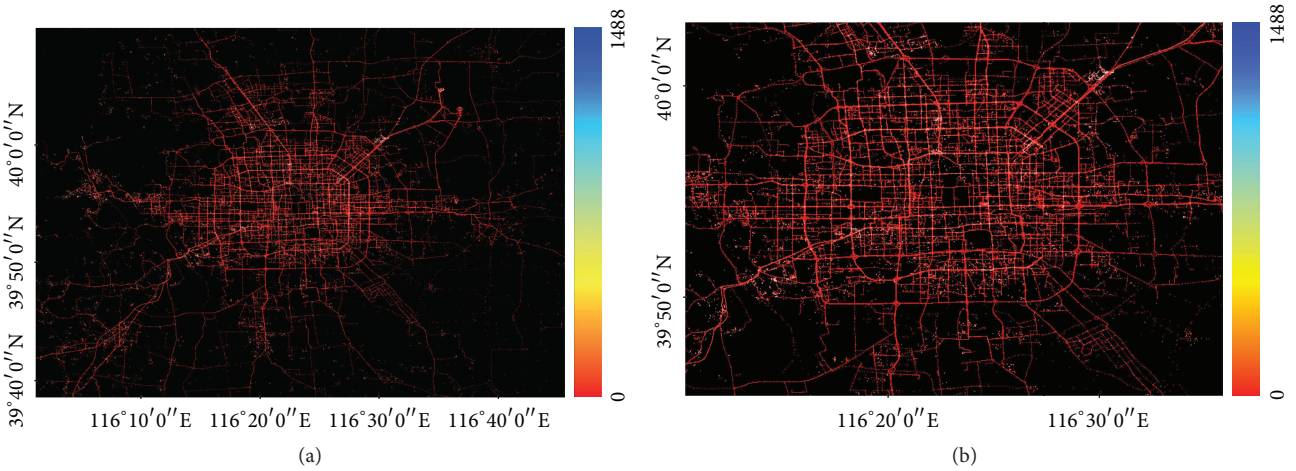

Figure 1: Road network produced via GPS points of taxi trajectories with density distribution. (a) Overview of Beijing and (b) the 5th Ring Road of Beijing.

Naturally, taxi trajectory data are becoming an important mobile trajectory data source that is widely utilized by industries, academia, and governments for many practical applications, particularly clustering GPS trajectories of taxicabs to provide a new idea for traffic subarea division. K-Means is the most commonly used partitional clustering algorithm [12], but it suffers three major shortcomings [13]: (i) its scalability is poor, (ii) K needs to be specified by the users, and (iii) the search is liable to local minima. Furthermore, K-Means has some bottlenecks in clustering the explosive growth of taxi trajectories, such as high memory consumption and I/O cost, low performance, and poor reliability. In particular, the execution time of K-Means is proportional to the product of the number of patterns and clusters in each iteration. The computational cost would be very high, especially for large data sets. Obviously, the sequential version of existing K-Means algorithms is not good at processing large-scale taxi trajectories on a single machine.

Recently, several parallel K-Means algorithms [14–22] have been proposed to meet the rapidly growing demands of clustering big data sets. Meanwhile, some methods [23–31] have been presented for traffic subarea division. All the previous approaches have almost achieved desirable properties but also have some limitations, especially the capacity of data processing that has not been improved substantially, and thus might have difficulty in dividing traffic subarea with a large number of GPS trajectories of taxicabs. To meet these challenges, this paper focuses on the improvement and parallelism of K-Means to enhance the accuracy and efficiency of large-scale taxi trajectories clustering, thereby solving traffic subarea division problem.

In this paper, we put forward a parallel K-Means optimization algorithm (Par3PKM) and implement it in a MapReduce framework on a Hadoop platform. Also, we divide traffic subarea of Beijing with a large number of GPS trajectories of taxicabs through our distributed traffic subarea division (DTSAD) method. More specifically, the distance metric and the initialization strategy of K-Means are modified in Par3PKM, and the time-consuming iteration is accomplished in the MapReduce model of computation. The accuracy and efficiency of clustering large-scale taxi trajectories are significantly improved. On the other hand, to save huge amounts of communication, memory consumption, and I/O cost, with MapReduce, DTSAD is performed on a distributed computing platform, Hadoop. Particularly, in DTSAD, a boundary identifying method can accurately connect the borders of each cluster.

The contributions of this work are summarized as follows:

(i) An efficient parallel clustering algorithm (Par3PKM) is proposed to address the existing problems of the traditional K-Means algorithm in processing massive traffic data. The evaluation results demonstrate that the Par3PKM algorithm can efficiently cluster a large number of GPS trajectories of taxicabs. In particular, it can offer a practical reference for implementing parallel computing of the same type of algorithms.

(ii) A new distributed division method (DTSAD) is presented to divide traffic subarea with large-scale taxi trajectories, and a boundary identifying method is put forward to connect the borders of clustering results. The experimental results indicate that the DTSAD method can accurately divide traffic subarea, based on the proposed Par3PKM algorithm.

(iii) The aforementioned approach is applied to the traffic subarea division of Beijing using big taxi trajectory data on a Hadoop platform with MapReduce and ArcGIS. Case studies show that the proposed approach significantly improves the accuracy and efficiency of traffic subarea division, especially the division results that are consistent with the real traffic conditions of corresponding areas in Beijing.

The remainder of this paper is organized as follows. Section 2 reviews related work, and the motivation with solution is given in Section 3. In Section 4, the Par3PKM

algorithm is described in detail. Section 5 presents the applications of our approach, and then the division results are analyzed. In Section 6, we evaluate the performance of the proposed algorithm and discuss the experimental results and then conclude the paper in Section 7.

2. Related Work

In this section, we first briefly review related work on the traffic subarea division and the parallel K-Means algorithms and then overview the MapReduce framework used in this paper.

2.1. Traffic Subarea Division. The concept of traffic subarea was first proposed by Walinchus [23], and various methods of traffic subarea division were subsequently developed for various ITSs applications including easing traffic congestion. Wong et al. [24] presented a time-dependent TRANSYT (Traffic Network Study Tool) traffic model for area traffic control, and Robertson and Bretherton [25] introduced a SCOOT (Split Cycle Offset Optimization Technique) method to optimize networks of traffic signals in real time. Ma and Yang [26] designed an expert system of traffic subarea division to trade off several demands for managing traffic network and reducing the complexity of traffic control, based on an integrated correlation index. Lu et al. [27, 28] built a model of partitioning traffic control subarea based on correlation degree analysis and optimized the strategy of subarea division. Guo et al. [29] provided a dynamic traffic control subarea division method, according to the similarity of adjacent intersections.

Furthermore, some researchers put forward the dimension-reduced processing and genetic algorithms to optimize subarea division. Li et al. [30] proposed a method to divide traffic control subarea dynamically on the basis of Back Propagation (BP) neural network. This method divides traffic subarea by considering traffic flow, distance of intersections, and cycle. In addition, to improve the performance of large-scale urban traffic networks division, Zhou et al. [31] presented a new approach to calculating the correlation degree which determines the desire for interconnection between two adjacent intersections.

Obviously, all the solutions mentioned above have many desirable properties but may have difficulty in processing a large number of taxi trajectories. In this work, we present a distributed traffic subarea division (DTSAD) method to improve the accuracy and efficiency of division, using large-scale GPS trajectories of taxicabs on a Hadoop platform with MapReduce and ArcGIS.

2.2. Parallel K-Means Algorithms. The K-Means algorithm proposed by MacQueen [32] is the most popular clustering method to partition a given data set into a user-specified number of clusters (i.e., K) [33–35] and needs several iterations over data sets before converging on a solution. The K-Means algorithm often converges to a local optimum. To address this problem, an incremental K-Means (IKM) algorithm [36] was put forward to empirically reach the global optimum but has a higher complexity. Nevertheless, the traditional K-Means algorithm requires several scans over data sets which have to be fully loaded into memory in order to enhance the efficiency of data accessing, but this requirement is hard to fulfill for handling massive data [34]. To overcome this drawback, Two-Phase K-Means [37] was introduced and can robustly find a good solution in one iteration by employing a buffering technique, but it is possibly unsuited to processing huge amounts of data.

Subsequently, Kantabutra and Couch [14] presented a parallel K-Means algorithm on a Message-Passing Interface (MPI) framework of a network of workstations (NOWs), and the main idea is that all data objects of a cluster are stored in a slave node. Data rearrangement of this method needs large-scale data transmission between slaves, which makes it difficult to deal with big data sets. Zhang et al. [15] implemented parallel K-Means based on a Parallel Virtual Machine (PVM) framework and performed on NOWs, and this parallel method requires the full load of the data sets on the master node and synchronization of data at the end of an iteration. Kraj et al. [16] proposed a parallel ParaKMeans algorithm, which makes good use of the multithreading method on a single machine to accomplish cluster and employs sufficient statistics to measure the quality of cluster in the stop condition. Pakhira [17] introduced a distributed K-Means algorithm which was executed on multiprocessor machines through randomly delivering the split data subset to each processor. Kohlhoff et al. [18] developed parallel K-Means (K_{ps}-Means) with GPUs implementation, which is efficient regardless of the dimensionality of the input data. Because the communication of different slaves takes up a large number of I/O resources and consumes huge amounts of time, these methods might have difficulty in dealing with large-scale data.

Moreover, to cluster big data effectively, some researchers implemented the parallelism of the K-Means algorithm based on a MapReduce framework. Chu et al. [19] developed a pragmatic and general framework that adopted the parallel programming method of MapReduce on a variety of learning algorithms, containing K-Means. Zhao et al. [20] put forward a PKMeans algorithm using MapReduce, and Zhou et al. [21] implemented an automatic classification of a large number of documents by PKMeans. Nguyen et al. [22] introduced a parallel Two-Phase K-Means algorithm (Par2PK-Means) to overcome the limitation of available parallel versions, and it achieves a better speedup than the sequential version.

To the best of our knowledge, all the aforementioned efforts are not successfully applied in solving traffic subarea division problem with large-scale taxi trajectories. However, we propose a parallel clustering algorithm (Par3PKM) based on MapReduce for parallel clustering of a large number of GPS trajectories of taxicabs in this work. Different from existing methods, in Par3PKM, we modify the distance metric and initialization strategy of K-Means for improving the accuracy of clustering and implement parallel computing of iteration in three phases to enhance the efficiency of computation. Furthermore, the addition of the Combiner function is employed to cut down the amount of data shuffled between the Map tasks and the Reduce tasks, thereby reducing the computational complexity of MapReduce job

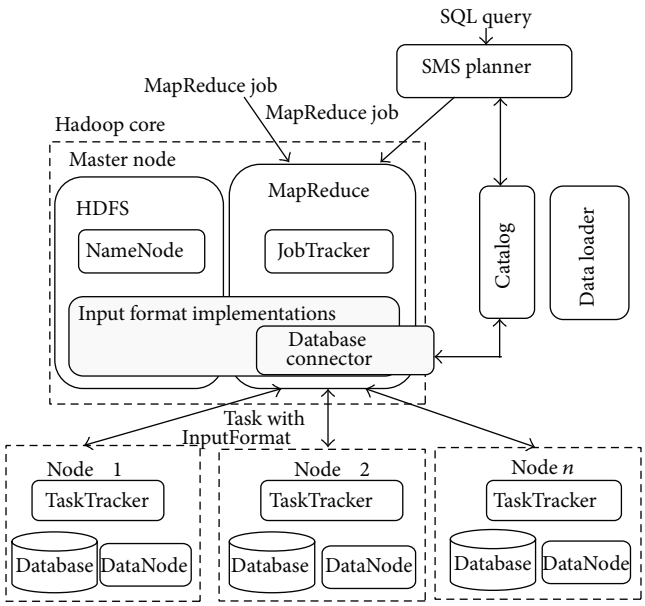

Figure 2: Architectures of HDFS and MapReduce.

and saving the limited bandwidth available on a Hadoop cluster.

2.3. MapReduce Framework. Hadoop Distributed File System (HDFS) [38] and Hadoop MapReduce [39, 40] are two core components of Hadoop [41–43], based on the open-source implementation of GFS [44] and MapReduce [45]. And their architectures are depicted in Figure 2. For further details on Hadoop, see the Apache Hadoop website (http://hadoop.apache.org/).

MapReduce is a parallel processing paradigm that allows for massive scalability across hundreds or thousands of servers on a Hadoop cluster [46] and particularly provides efficient computing framework to deal with big taxi trajectory data for traffic subarea division. As a typical methodology, the processing of MapReduce jobs includes the Map phase and the Reduce phase. Each phase has *key-value* pairs as input and output, the types of which may be selected by the programmer that specifies the Map function and the Reduce function [40]. A simple example is illustrated in Figure 3, which shows the logical data flow of a simple MapReduce job that calculates the maximum temperature for each year by mining huge amounts of weather data sets.

In this work, based on a MapReduce framework, the time-consuming iterations of the proposed Par3PKM algorithm are performed in three phases with the Map function, the Combiner function, and the Reduce function, and the parallel computing process of MapReduce is shown in Figure 4. Specifically, in Par3PKM, the incremental Combiner function is executed between the Map tasks and the Reduce tasks, which can reduce the computational complexity of MapReduce jobs and save the limited bandwidth available on a Hadoop cluster.

3. Motivation

In this section, we describe the motivation of this work and give a reasonable solution for solving traffic subarea division problem.

Naturally, GPS-equipped taxi is an essential public traffic tool in modern cities. Over the last decade, taxi trajectory data have been exploding and have become the most important mobile trajectory data source with the advantages of broad covering, extra precision, excellent continuity, little privacy, and so forth. Thus, for solving traffic subarea division problem, how to substantially improve the capacity of processing large-scale GPS trajectories of taxicabs poses an urgent challenge to us. On the other hand, as one of the most well-known clustering techniques in practice, the traditional K-Means algorithm is a simple iterative approach but has very high time complexity and memory consumption. The time requirements for K-Means are basically linear in the number of data objects. The time required is $O(K * I * n * m)$, where $K \leq n$, $I \leq n$; K denotes the number of desired clusters, I is the number of iterations required for convergence, n is the total number of objects, and m represents the number of attributes in the given data sets. In particular, the storage required is $O((K + n) * m)$. Obviously, the efficiency of the serial K-Means algorithm is low in handling a large number of taxi trajectories with limited memory on a single machine. Hence, to enhance the accuracy and efficiency of clustering, another challenge is how to optimize K-Means and implement it in a MapReduce framework on a Hadoop platform.

These challenges motivate the development of the Par3PKM algorithm with the DTSAD method, and a new solution to the above problems is illustrated in Figure 5. As shown in Figure 5, based on a Hadoop platform with MapReduce and ArcGIS, the process of the DTSAD method mainly includes the following steps. First, we preprocess correlation data extracted from large-scale taxi trajectories. Then, we cluster huge amounts of trajectory data in parallel, using the proposed Par3PKM algorithm as described in Section 4. Finally, we identify the borders of clustering results to build each traffic subarea, by our boundary identifying method as depicted in Section 5.3.

Clearly, the key to this solution is the accuracy and efficiency of clustering large-scale taxi trajectories, which determines the overall performance of traffic subarea division. Thus, the aim of this paper is to put forward an efficient parallel clustering algorithm (Par3PKM) with MapReduce implementation.

4. The Proposed Par3PKM Algorithm

In this section, the Parallel Three-Phase K-Means (Par3PKM) algorithm is proposed for efficiently and accurately clustering a large number of GPS trajectories of taxicabs, under a MapReduce framework on Hadoop.

4.1. Overview and Notation. The process of the Par3PKM algorithm is depicted in Figure 6. Based on a MapReduce framework, the Par3PKM algorithm first chooses K of the

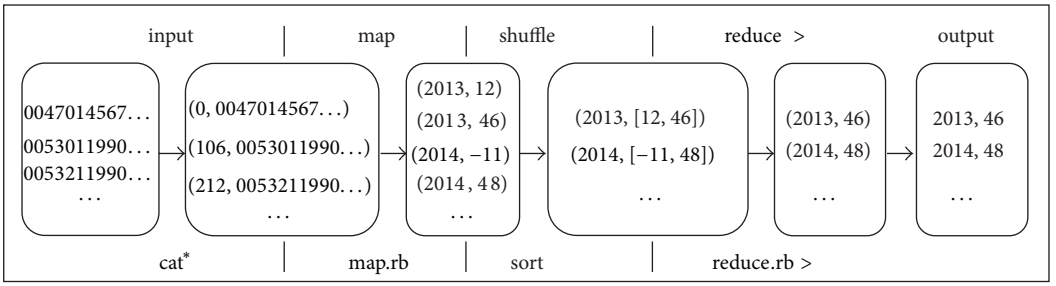

Figure 3: Logical data flow of a simple MapReduce job.

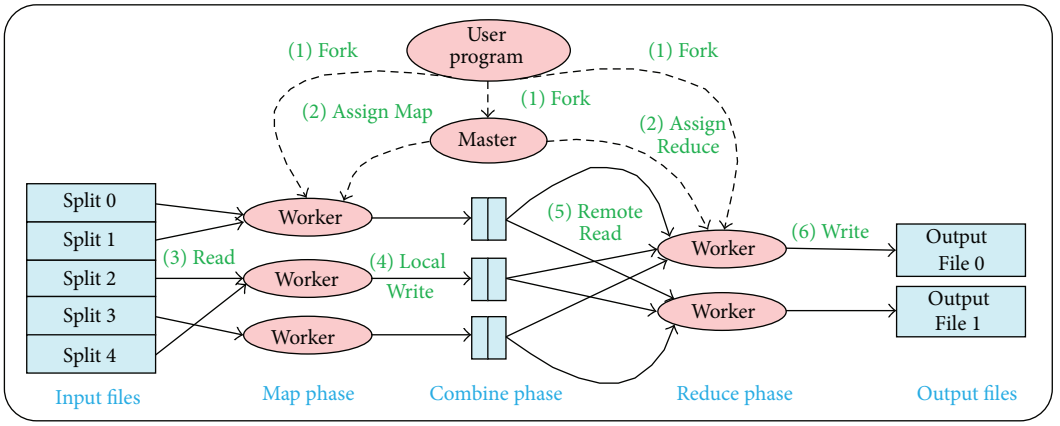

Figure 4: Parallel computing process of MapReduce.

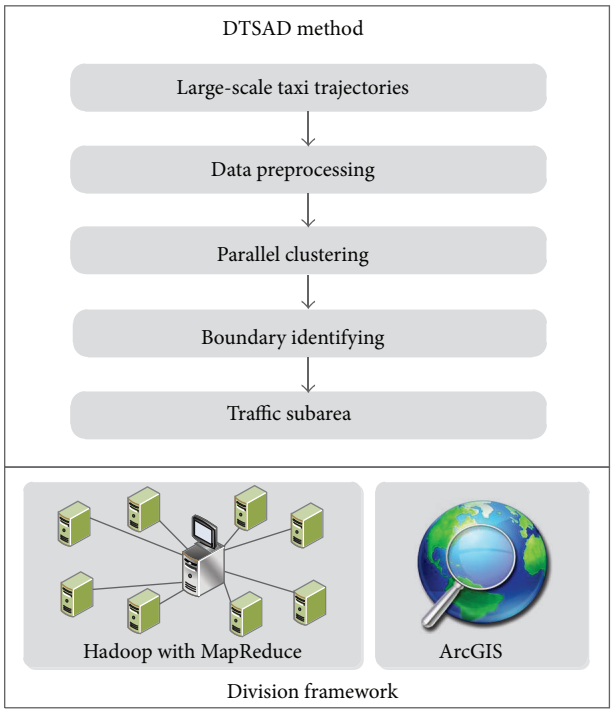

Figure 5: Process and framework of the DTSAD method.

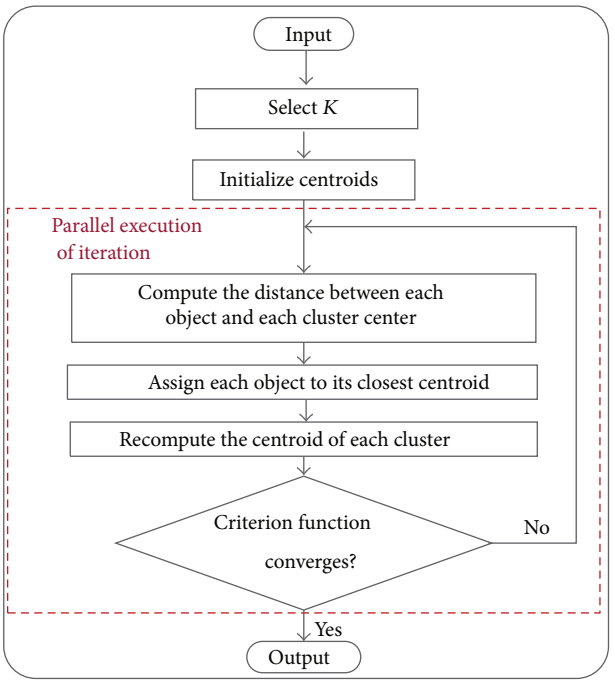

Figure 6: Process of the Par3PKM algorithm.

objects as initial centroids, where K is the number of desired clusters. Then, each of the remaining objects is assigned to the cluster to which it is the most similar, on the basis of the distance between the object and the cluster center. Finally, it computes the new center for each cluster, and this

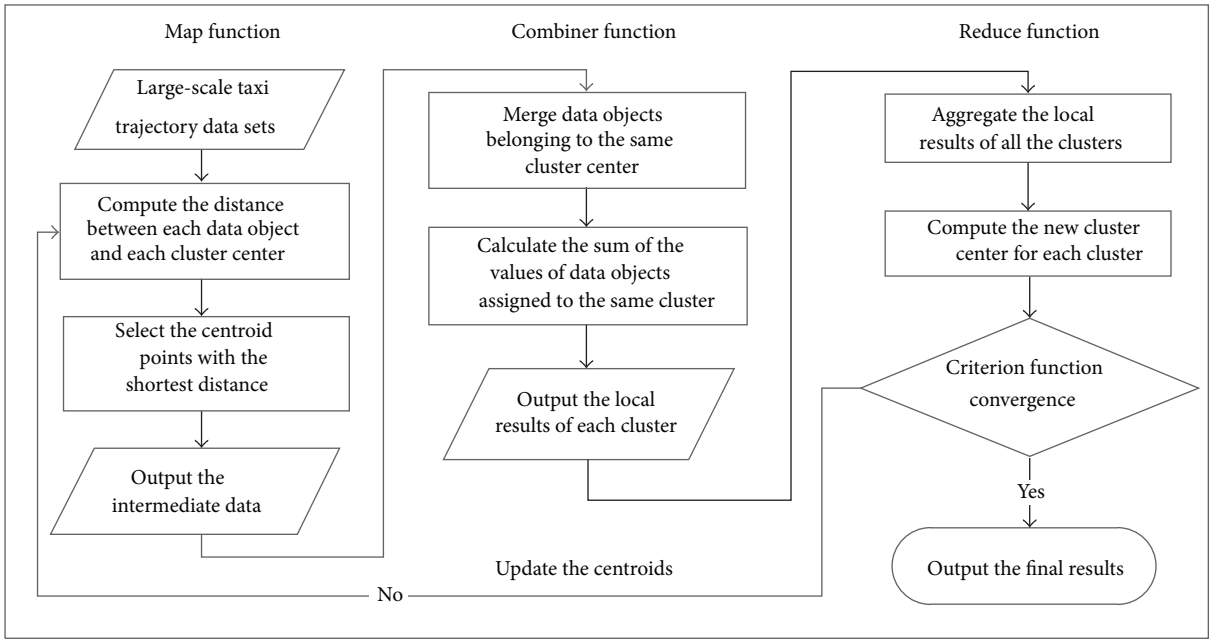

Figure 7: Parallel execution process of iteration.

process iterates until the criterion function converges. The parallel execution of iteration is illustrated in Figure 7, and its MapReduce implementation will be described in Section 4.3 in detail.

Typically, the squared-error criterion [33] is used to measure the quality of clustering. Let $X = \{x_i \mid i = 1, \ldots, n\}$ be the set of n m-dimensional vectors to be clustered into a set of K clusters, $C = \{c_k \mid k = 1, \ldots, K\}$. Let μ_k be the mean of cluster c_k.

The squared-error criterion between μ_k and the given object in cluster c_k is defined as

$$J(c_k) = \sum_{x_i \in c_k} \|x_i - \mu_k\|^2. \tag{1}$$

The goal of the Par3PKM algorithm is to minimize the sum of the squared errors (SSE) over all the K clusters, and SSE is given by the following equation:

$$\text{SSE} = J(C) = \sum_{k=1}^{K} \sum_{x_i \in c_k} \|x_i - \mu_k\|^2. \tag{2}$$

4.2. Distance Measure and Cluster Initialization. It is well known that the K-Means algorithm is sensitive to distance measure and cluster initialization. To overcome the critical limitations, we optimize the distance metric and initialization strategy for improving the accuracy and efficiency of clustering in the proposed Par3PKM algorithm.

4.2.1. Distance Metric. The traditional K-Means algorithm typically employs Euclidean metric to compute the distance between objects and cluster centers. In the proposed Par3PKM algorithm, we attempt to adopt two rules using a statistical approach [47] in terms of distance measure selection, which is more appropriate for large-scale trajectory data sets.

Two rules of distance measure are illustrated as follows:

(i) If $\kappa \leq 8.46$, square Euclidean distance (see (5)) is chosen as the distance measure of Par3PKM.

(ii) If $\kappa > 8.46$, Manhattan distance (see (6)) is selected as the distance measure of Par3PKM.

Here, κ is the kurtosis which measures the tail heaviness. It is defined as

$$\kappa = \frac{(1/n)\sum_{i=1}^{n}\left(x_i - \overline{X}\right)^4}{\sigma^4}, \tag{3}$$

where n represents the sequence length and sample mean $\overline{X}$ and sample standard deviation σ are given by

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} x_i,$$
$$\sigma = \left(\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \overline{X}\right)^2\right)^{1/2}. \tag{4}$$

The square Euclidean distance d_e^2 and the Manhattan distance d_m are, respectively, given by

$$d_e^2(x_i, \mu_k) = \sum_{j=1}^{m} \left|x_{ij} - \mu_{kj}\right|^2, \tag{5}$$

$$d_m(x_i, \mu_k) = \sum_{j=1}^{m} \left|x_{ij} - \mu_{kj}\right|, \tag{6}$$

where $x_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ and $\mu_k = (\mu_{k1}, \mu_{k2}, \ldots, \mu_{km})$ are two m-dimensional data objects.

The Par3PKM algorithm achieves more accurate clustering performance than the K-Means algorithm via the statistical based distance measure method, which will be estimated in Section 6.

4.2.2. Initialization Strategy. Different initialization strategies will lead to different efficiencies. In our Par3PKM, three initialization strategies are developed for enhancing the efficiency of clustering. To overcome the local minima of Par3PKM, for a given K with several different initial partitions, we select the partition with the smallest value of the squared error. Furthermore, inspired by [48], we take the mutual farthest data objects in the high density area as the initial centroids for Par3PKM. The method of obtaining the initial centroids from the high density area was introduced in [49]. In addition, both pre- and postprocessing on Par3PKM are taken into consideration; for example, we remove outliers in a preprocessing step and eliminate small cluster and/or merge close clusters into a large cluster in postprocessing the results.

From the experimental evaluations as described in Section 6, we can observe that the implemented initialization strategies shorten the execution time of Par3PKM with the same clustering results.

4.3. MapReduce Implementation. To improve the efficiency and scalability of clustering, we implement the Par3PKM algorithm in the MapReduce model of computation. The tasks of Par3PKM are mainly composed of the following three aspects:

(i) Compute the distance between the object and the cluster center.

(ii) Assign each object to its closest centroid.

(iii) Recompute new cluster centers for each cluster.

In this work, we accomplish the aforementioned tasks on a Hadoop platform using MapReduce. First, we select K data objects (vectors) as the initial cluster centers from the input data sets which are stored in HDFS. Then, we update the centroid of each cluster after iterating. As depicted in Figure 7, the parallel execution of iteration is implemented by the Map function, the Combiner function, and the Reduce function in three phases (i.e., Map phase, Combine phase, and Reduce phase), respectively.

4.3.1. Map Phase. In this phase, Map task receives each line in the sequence file as different key-value pairs which forms the input to the Map function. The Map function first computes the distance between each data object and each cluster center, then assigns each object to its closest centroid according to the shortest distance, and finally outputs the intermediate data to the Combiner function. The Map function is formally depicted in Algorithm 1.

4.3.2. Combine Phase. In this phase, the Combiner function first extracts all the data objects from $value_1$ which is the output of the Map function and merges data objects belonging to the same cluster center. Next, it calculates the sum of the values of data objects assigned to the same cluster and records the number of samples in the same cluster, so as to compute the mean value of data objects. Finally, it outputs the local results of each cluster to the Reduce function. The Combiner function is formally described in Algorithm 2.

4.3.3. Reduce Phase. In this phase, the Reduce function extracts all the data objects from $value_2$ which is the output of the Combiner function and aggregates the local results of all the clusters. Then, it computes the new cluster center for each cluster. After that, it judges whether the criterion function converges. Finally, it outputs the final results if its argument is true and executes next iteration otherwise. The Reduce function is formally illustrated in Algorithm 3.

4.4. Complexity Analysis. MapReduce is a programming model for large-scale data processing, and MapReduce programs especially are inherently parallel. Thus, the Par3PKM algorithm with MapReduce implementation puts large numbers of computational tasks into different nodes. According to this parallel processing paradigm, the time complexity of Par3PKM is $O(K * I * n * m)/p * q$, where p is the number of nodes, q is the number of Map tasks in one node, and K, I, n, and m are explained in Section 3. Moreover, the space complexity of Par3PKM is $O((K + n) * m)/p$. Only the data objects and the centroids are stored in each slave node, respectively; thus the space requirements for Par3PKM are modest.

In comparison with the traditional K-Means algorithm, the Par3PKM algorithm improves the efficiency of clustering through the following rate equation:

$$O_{\text{imp}} = \left(1 - \frac{1}{p * q}\right) \times 100\%. \tag{7}$$

Based on the above analyses, we can conclude that the Par3PKM algorithm is relatively scalable and efficient in clustering large-scale data sets, with the desired computational complexity.

5. Case Study

In this section, we apply the proposed approach to divide traffic subarea of Beijing using large-scale taxi trajectories and then analyze the results.

5.1. Data Sets and Framework. In this work, we divide traffic subarea of Beijing based on real-world trajectory data sets (http://www.datatang.com/), which contains a large number of GPS trajectories recorded by 12,000 taxicabs during a period of 30 days in November 2012. The total distance of the data sets is more than 50 million kilometers and the total size is 50 GB. Particularly, the total number of GPS points reaches 969 million.

Additionally, we perform the case based on a cluster of Hadoop with MapReduce (as described in Section 6.1) and ArcGIS, using the road network of Beijing which consists of 106,579 road nodes and 141,380 road segments.

Input:
key: the offset,
value: the sample,
centroids: the global variable.
Output: $\langle key_1, value_1 \rangle$,
key_1: the index of the closest centroid,
$value_1$: the information of sample.
(1) Construct a global variable *centroids* including the information of the closet centroid point;
(2) Construct the sample *examples* to extract the data objects from *value*;
(3) *min_Dis* = *Double.MAX_VALUE*;
(4) *index* = −1;
(5) **for** *i* = 0 to *centroids.length* **do**
(6) *distance* = *DistanceFunction*(*examples*, *centroids*[*i*]);
(7) **if** *distance* < *min_Dis* **then**
(8) *min_Dis* = *distance*;
(9) *index* = *i*;
(10) **end if**
(11) **end for**
(12) *index* = key_1;
(13) Construct $value_1$ as a string consisting of the values of different dimensions;
(14) **return** $\langle key_1, value_1 \rangle$ pairs;

Algorithm 1: Map(*key*, *value*).

Input:
key_1: the index of the cluster,
medi: the list of the samples assigned to the same cluster.
Output: $\langle key_2, value_2 \rangle$,
key_2: the index of the cluster,
$value_2$: the sum of the values of the samples belonging to the same cluster and the number of samples.
(1) Construct a counter *num_s* to record the number of samples in the same cluster;
(2) Construct an array *sum_v* to record the sum of the values of different dimensions of the samples belonging to the same cluster (i.e., the samples in the list *medi*);
(3) Construct the sample *examples* to extract the data objects from *medi*.next(), and the *dimensions* to obtain the dimension of the original data object;
(4) *num_s* = 0;
(5) **while** (*medi*.hasNext()) **do**
(6) CurrentPoint = *medi*.next();
(7) *num_s*++;
(8) **for** *i* = 0 to *dimensions* **do**
(9) *sum_v*[*i*]+ = CurrentPoint.point[*i*];
(10) //Calculate the sum of the values of each dimension of *examples*
(11) **end for**
(12) **for** *i* = 0 to *dimensions* **do**
(13) *mean*[*i*] = *sum_v*[*i*]/*num_s*;
(14) //Compute the mean value of the samples for each cluster
(15) **end for**
(16) **end while**
(17) *index* = key_2;
(18) Construct $value_2$ as a string containing the sum of the values of each dimension *sum_v*[*i*] and the number of samples *num_s*;
(19) **return** $\langle key_2, value_2 \rangle$ pairs;

Algorithm 2: Combiner(key_1, *medi*).

Input:
key_2: the index of the cluster,
medi: the list of the local sums from different clusters.
Output: $\langle key_3, value_3 \rangle$,
key_3: the index of the cluster,
$value_3$: the new cluster center.
(1) Construct a counter *Num* to record the total number of samples belonging to the same cluster;
(2) Construct an array *sum_v* to record the sum of the values of different dimensions of the samples in the same cluster (i.e., the samples in the list *medi*);
(3) Construct the sample *examples* to extract the data objects from *medi*.next(), and the *dimensions* to obtain the *dimension* of the original data object;
(4) *Num* = 0;
(5) **while** (*medi*.hasNext()) **do**
(6) CurrentPoint = *medi*.next();
(7) *Num*+ = *num_s*;
(8) **for** *i* = 0 to *dimensions* **do**
(9) *sum_v*[*i*]+ = CurrentPoint.point[*i*];
(10) **end for**
(11) **for** *i* = 0 to *dimensions* **do**
(12) *mean*[*i*] = *sum_v*[*i*]/*Num*;
(13) //Obtain the new cluster center
(14) **end for**
(15) **end while**
(16) *index* = key_3;
(17) Construct $value_3$ as a string composed of the new cluster center;
(18) **return** $\langle key_3, value_3 \rangle$ pairs;

Algorithm 3: Reduce(key_2, *medi*).

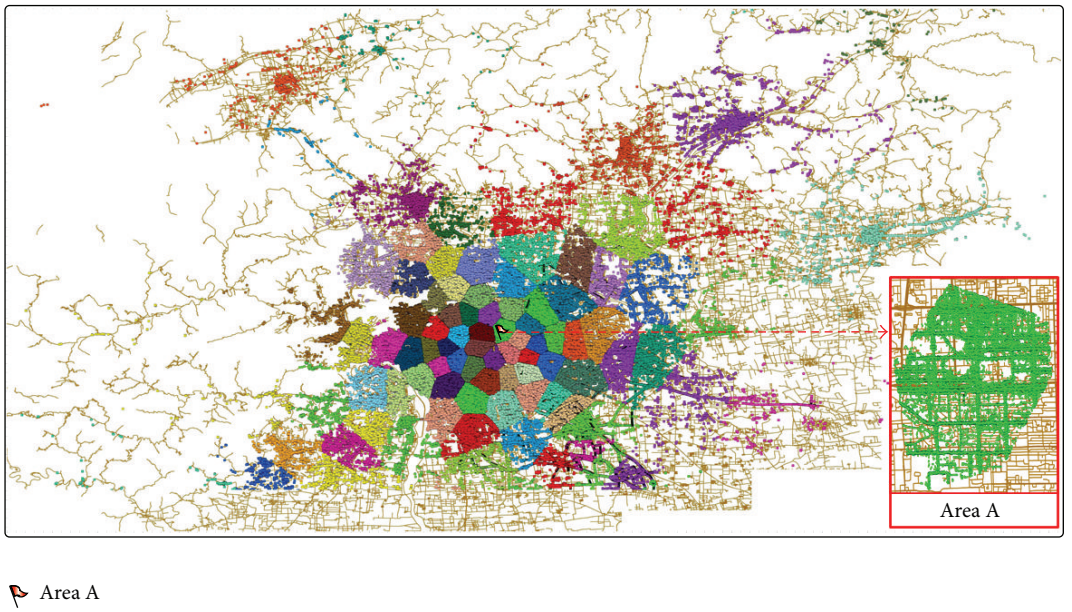

Figure 8: Clustering results of large-scale taxi trajectories.

5.2. Parallel Clustering. After data preprocessing, with the DTSAD method, we extract the relevant attributes (e.g., longitude, latitude) of the GPS trajectory records where the passengers pick up or drop off taxis from the aforementioned data sets. Then, on a Hadoop cluster with MapReduce, we cluster the extracted trajectory data sets through the Par3PKM algorithm (as depicted in Section 4) with K = 100, which is the number of desired clusters. Finally, based on the ArcGIS platform with the road network of Beijing, we plot the results in Figure 8.

As illustrated in Figure 8, large-scale taxi trajectories are clustered in different areas, respectively, and each area with

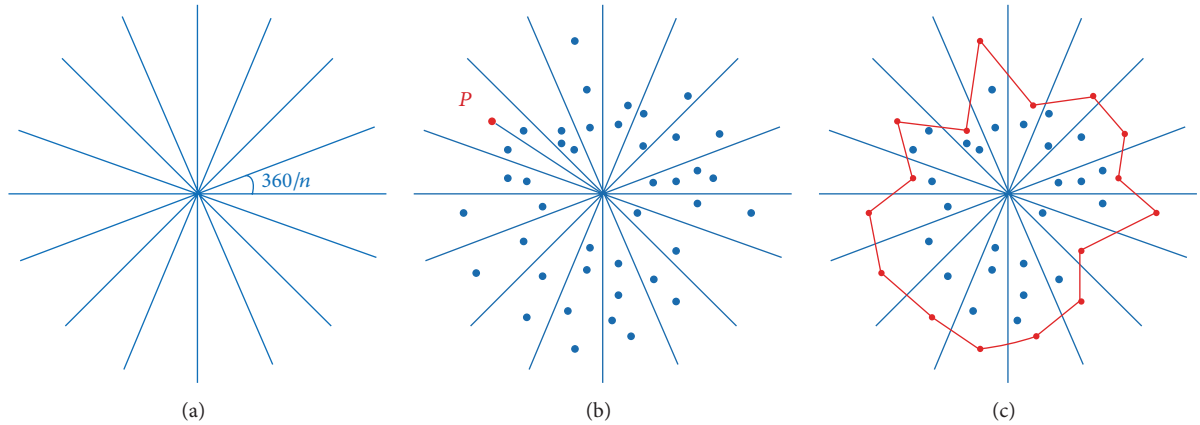

Figure 9: Process of the boundary identifying method. (a) Division of coordinate system, (b) selection of border points, and (c) connection of border points.

different colors represents a cluster. Each cluster (e.g., Area A) has obvious characteristics of traffic condition, such as the flow of people and automobile which is high in these areas, in comparison with real traffic map and traffic condition of Beijing.

5.3. Boundary Identifying. On the ArcGIS platform, we have difficulty in identifying the borders of each cluster. However, we have to connect the borders of each cluster in order to accurately form traffic subarea via our boundary identifying method, which is described in Figure 9. As illustrated in Figure 9, the boundary identifying method is mainly composed of the following three steps.

Step 1. As shown in Figure 9(a), we build a coordinate system, which is equally divided into n parts through taking (0, 0) as the origin of coordinates.

Step 2. We match each cluster center to the origin of coordinates and then map other points to the coordinate system in the same cluster. Finally, the farthest points of each part are selected (e.g., P in Figure 9(b)).

Step 3. As depicted in Figure 9(c), we connect these selected points of each part and then obtain a subarea.

Using the presented boundary identifying method with the clustering results of Par3PKM, we plot the division results of traffic subarea in Figure 10. As described in Figure 10, Area A is a typical traffic subarea shown in the lower right corner of the graph, which includes Tsinghua University, Peking University, Renmin University of China, Beihang University, and Zhongguancun. Area B is composed of the Beijing Workers' Gymnasium, Blue Zoo Beijing, Children Outdoor Paopao Paradise, and so forth. Area C consists of Beijing North Railway Station, Beijing Exhibition Center, Capital Indoor Stadium, and so forth. Area D contains Olympic Sports Center Stadium, Beijing Olympic Park, National Indoor Stadium, Beijing National Aquatics Center, Beijing International Convention Center, and so forth.

5.4. Analysis of Results. According to the division results, we can observe that some areas with similar traffic conditions are divided into the same traffic subarea, such as the Tsinghua University and the Peking University, and the Olympic Sports Center Stadium and the National Indoor Stadium. In contrast, the Blue Zoo Beijing and the Beijing Exhibition Center, and the Beijing North Railway Station and Beijing International Convention Center are classified into different traffic subareas. That is because the different regions of a traffic subarea have great similarities and correlations in traffic condition, business pattern, and other aspects.

Based on the above analysis, we conclude that the proposed Par3PKM algorithm can efficiently cluster big trajectory data on a Hadoop cluster using MapReduce. Moreover, our boundary identifying method can accurately connect the borders of clustering results for each cluster. In particular, the division results are consistent with the real traffic condition of the corresponding areas in Beijing. Overall, the results demonstrate that Par3PKM combined with DTSAD is a promising alternative for traffic subarea division with large-scale taxi trajectories and thus can reduce the complexity of traffic planning, management, and analysis. More importantly, it can provide helpful decision-making for building ITSs.

6. Evaluation and Discussion

In this section, we evaluate the accuracy, efficiency, speedup, scale-up, and reliability of the Par3PKM algorithm via the extensive experiments on real and synthetic data and then discuss the performance results.

6.1. Evaluation Setup. The experimental platform based on a Hadoop cluster, which is composed of one master machine and eight slave machines with Intel Xeon E7-4820 2.00 GHz CPU (4-core) and 8.00 GB RAM. All the experiments are performed on Ubuntu 12.04 OS with Hadoop 1.0.4 and JDK 1.6.0.

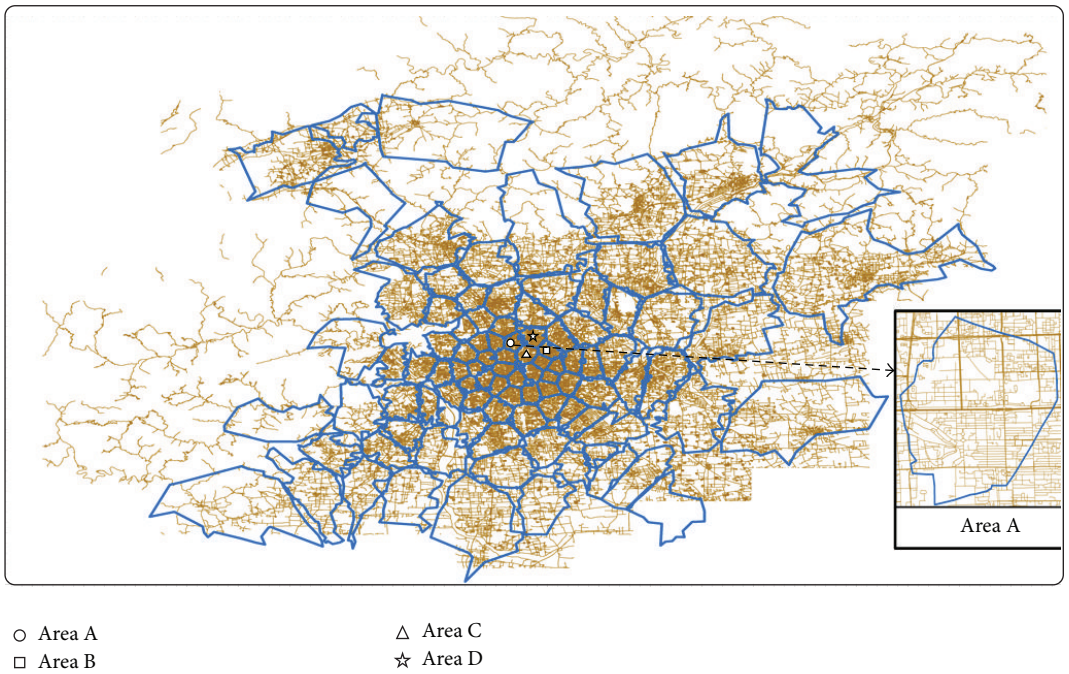

Figure 10: Division results of traffic subarea.

Table 1: Data sets of the experimental evaluations.

Name	Number of instances	Number of attributes
Iris	150	4
Haberman's Survival	306	3
Ecoli	336	8
Hayes-Roth	160	5
Lenses	24	4
Wine	178	13

In addition to a real taxi trajectory data set (as described in Section 5.1), we use six synthetic data sets (as shown in Table 1) selected from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets.html), to evaluate the performance of the Par3PKM algorithm, in comparison with K-Means, Par2PK-Means, and ParCLARA, which is the best-known K-medoid algorithm CLARA (Clustering Large Applications) [50] with MapReduce implementation. Meanwhile, each data set is processed into 80 MB, 160 MB, 320 MB, 640 MB, 1280 MB, and 2560 MB so as to further verify the efficiency of the proposed algorithm. Also, we handle seven data sets into 160 MB, 320 MB, 480 MB, 640 MB, 800 MB, 960 MB, 1120 MB, and 1280 MB for validating the scale-up of our algorithm.

6.2. Evaluation on Efficiency. We perform efficiency experiments where we execute Par3PKM, Par2PK-Means, and ParCLARA in the parallel environment with eight nodes and K-Means in the single machine environment using seven data sets with different sizes (varying from 80 MB to 2560 MB), respectively, in order to demonstrate whether the Par3PKM algorithm can process larger data sets and has higher efficiency. The experimental results are, respectively, shown in Table 2 and Figure 11.

As depicted in Figure 11, the K-Means algorithm cannot process over 1280 MB data sets in the single machine environment on account of the memory overflow, and thus the graph does not present the corresponding execution time of K-Means in this experiment. However, the Par3PKM algorithm can effectively handle more than 1280 MB data sets even larger data in the parallel environment (i.e., in "Big Data" environment). In particular, the Par3PKM algorithm has higher efficiency than the ParCLARA algorithm and the Par2PK-Means algorithm, with the improvement and the parallelism of the K-Means algorithm, such as the addition of the Combiner function in the Combine phase. To reduce the computational complexity of MapReduce job and save the limited bandwidth available on a Hadoop cluster, the Combiner function of the Par3PKM algorithm is employed to cut down the amount of data shuffled between the Map tasks and the Reduce tasks and is specified to be run on the Map output and its output forms the input to the Reduce function.

At the same time, we can find that the execution time of the K-Means algorithm is shorter than that of the Par3PKM algorithm in clustering small-scale data sets. The reason is that the communication and interaction of each node consume a certain amount of time in the parallel environment (e.g., the start of Job and Task tasks, the communication between NameNode and DataNodes), thereby leading to the execution time being much longer than the actual computation time of the Par3PKM algorithm. More importantly, we can also observe that the efficiency of the Par3PKM algorithm is improving multiply and the superiority is more marked, with the gradually increasing sizes of data sets.

Table 2: Execution time comparison on seven data sets.

Data sets	Size (MB)	Execution time (s)			
		K-Means	ParCLARA	Par2PK-Means	Par3PKM
Taxi Trajectory	80	75	662	439	312
	160	103	756	486	371
	320	195	893	579	406
	640	1125	1173	838	614
	1280	2373	1679	1348	1026
	2560	—	2721	2312	1879
Iris	80	32	612	301	213
	160	54	693	380	279
	320	116	812	440	306
	640	685	1045	630	403
	1280	1800	1248	1005	768
	2560	—	2463	2013	1420
Haberman's Survival	80	56	576	311	296
	160	60	675	400	324
	320	130	823	470	378
	640	720	987	670	426
	1280	2010	1321	1200	873
	2560	—	2449	2200	1719
Ecoli	80	38	628	330	283
	160	66	712	400	324
	320	130	835	460	375
	640	756	1104	700	482
	1280	1912	1636	1234	763
	2560	—	2479	2312	1416
Hayes-Roth	80	41	568	310	278
	160	57	643	395	347
	320	125	726	460	387
	640	715	973	660	438
	1280	1980	1479	1211	736
	2560	—	2423	2120	1567
Lenses	80	32	624	309	297
	160	59	701	389	327
	320	130	924	452	376
	640	700	1072	545	432
	1280	1895	1378	1085	814
	2560	—	2379	2089	1473
Wine	80	50	635	350	317
	160	78	705	420	356
	320	130	835	470	402
	640	730	1006	610	426
	1280	2100	1346	1245	843
	2560	—	2463	2240	1645

6.3. Evaluation on Accuracy. To evaluate the accuracy, we cluster different data sets via Par3PKM, Par2PK-Means, and ParCLARA based on a Hadoop cluster with eight nodes and through K-Means in the single machine environment, respectively. We then plot the results in Figure 12(a).

The quality of the algorithm is evaluated via the following error rate (ER) [51] equation:

$$\mathrm{ER} = \frac{O_m}{O_t} \times 100\%, \tag{8}$$

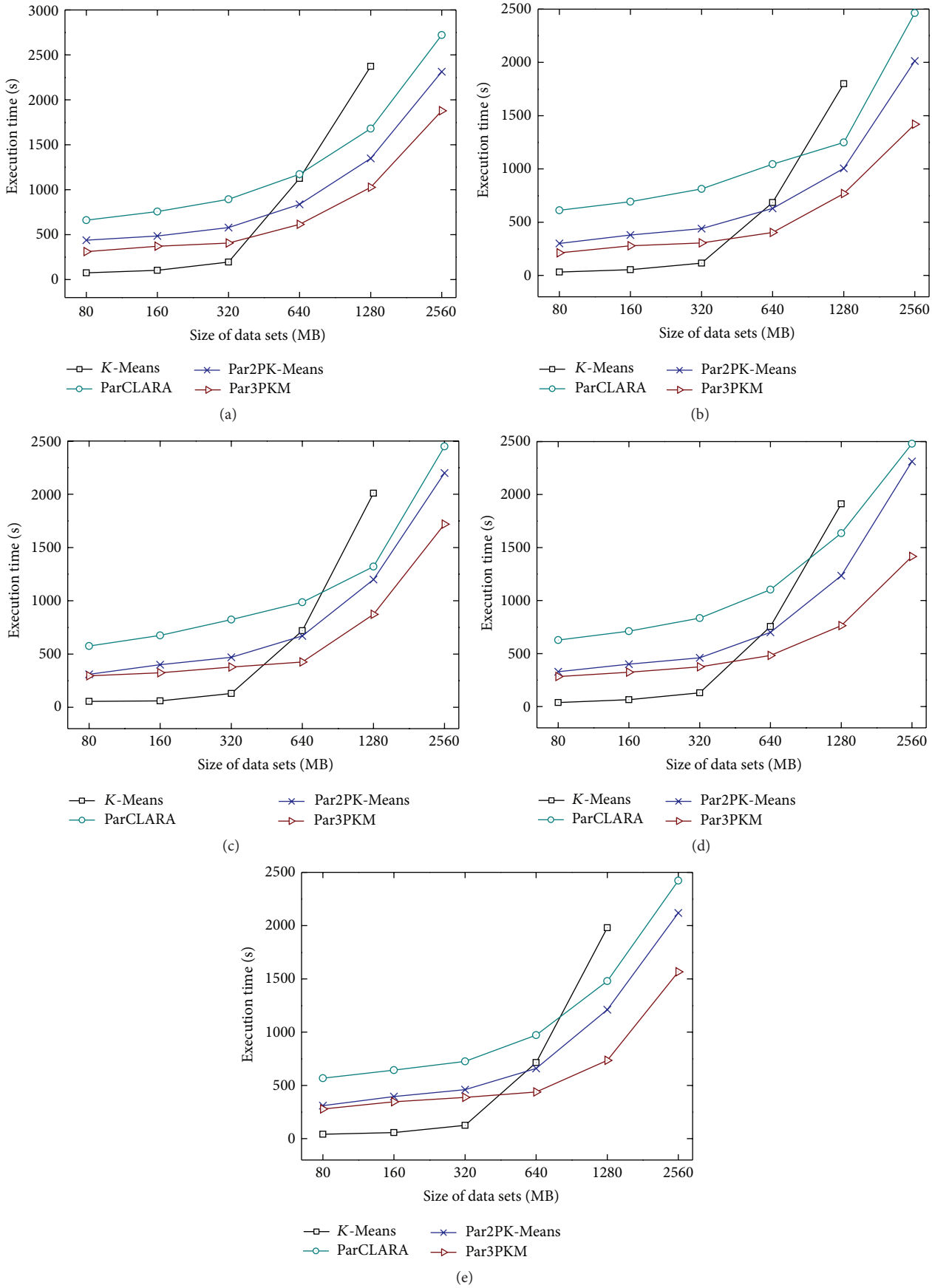

Figure 11: Continued.

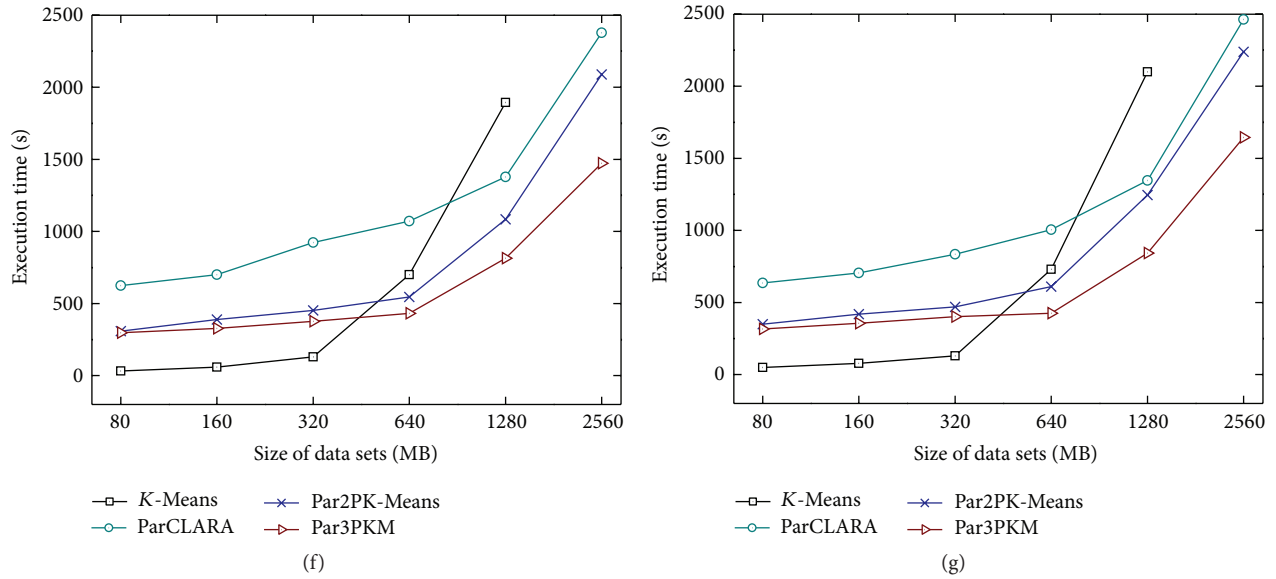

Figure 11: Efficiency comparison on different data sets. (a) Taxi Trajectory, (b) Iris, (c) Haberman's Survival, (d) Ecoli, (e) Hayes-Roth, (f) Lenses, and (g) Wine.

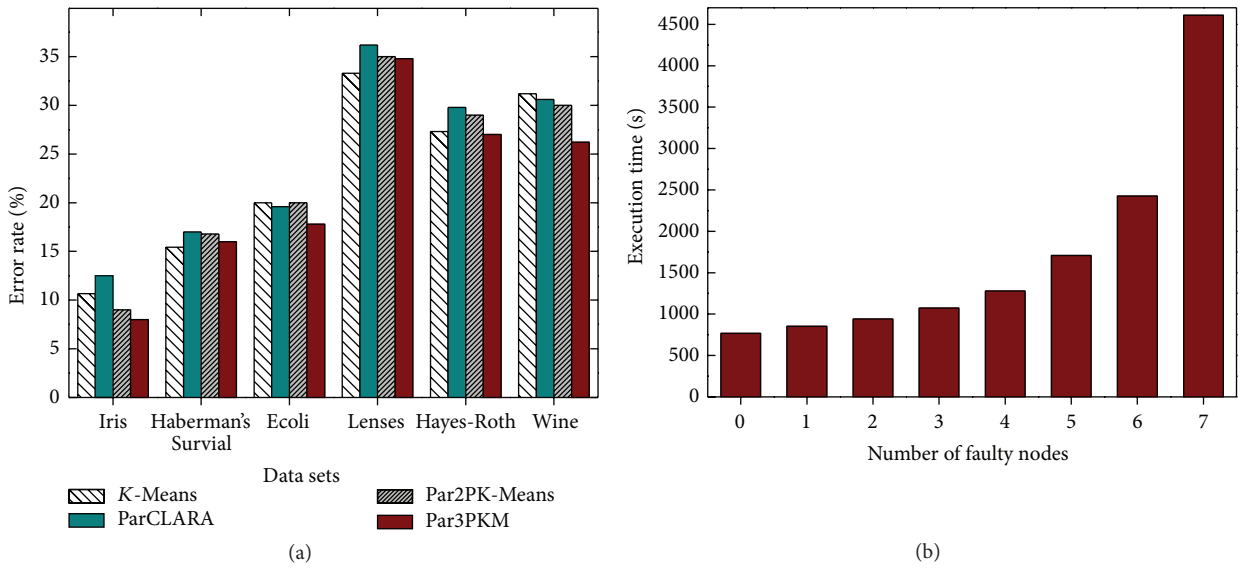

Figure 12: Accuracy and reliability. (a) Accuracy comparison of Par3PKM, Par2PK-Means, and ParCLARA on different data sets and (b) reliability of Par3PKM.

where O_m is the number of misclassified objects and O_t is the total number of objects. The lower the ER, the better the clustering.

As illustrated in Figure 12(a), in comparison with other algorithms, the Par3PKM algorithm produces more accurate clustering results in most cases. The results indicate that the Par3PKM algorithm is valid and feasible.

6.4. Evaluation on Speedup. In order to evaluate the speedup of the Par3PKM algorithm, we keep seven data sets constant (1280 MB) and increase the number of nodes (ranging from 1 node to 8 nodes) on a Hadoop cluster and then plot the results in Figure 13(a). Moreover, we utilize Iris data sets (1280 MB) for further verifying the speedup of Par3PKM, in comparison with Par2PK-Means and ParCLARA, and the results are illustrated in Figure 13(b).

The speedup metric [52, 53] is defined as

$$\text{Speedup} = \frac{T_s}{T_p}, \tag{9}$$

where T_s represents the execution time of an algorithm on one node (i.e., the sequential execution time) for clustering

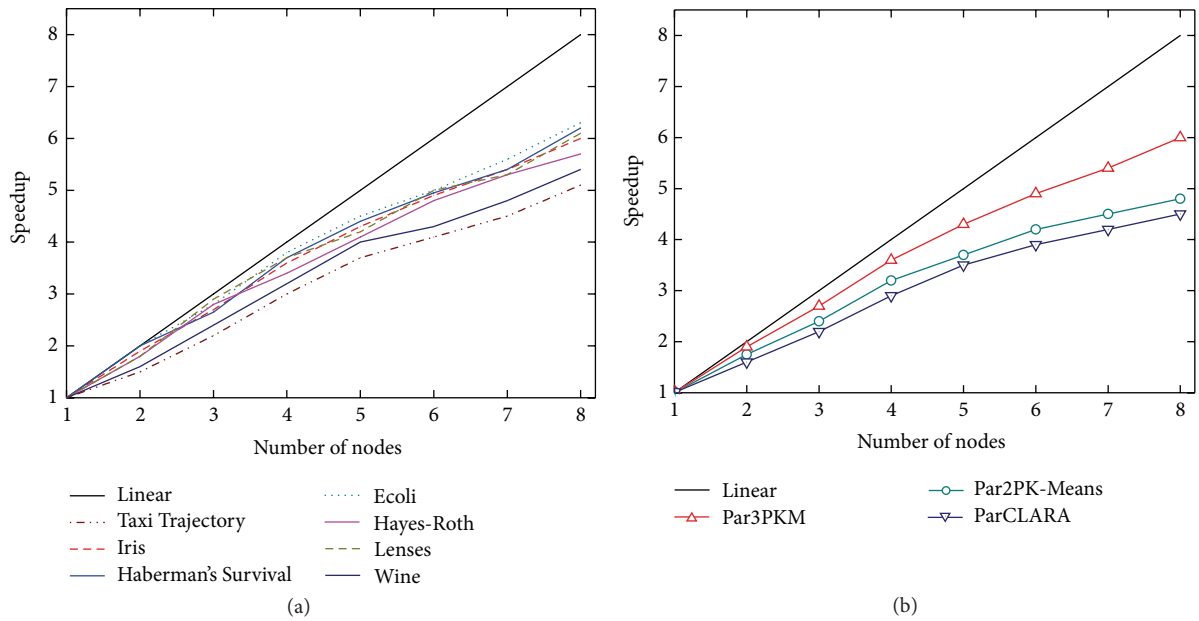

FIGURE 13: Speedup. (a) Speedup comparison of Par3PKM on different data sets and (b) speedup comparison of Par3PKM, Par2PK-Means, and ParCLARA.

objects using the given data sets and T_p denotes the execution time of an algorithm for solving the same problem using the same data sets on a Hadoop cluster with p nodes (i.e., the parallel execution time), respectively.

As depicted in Figure 13(a), the speedup of the Par3PKM algorithm increases relatively linearly with an increasing number of nodes. It is known that linear speedup is difficult to achieve because of the communication cost and the skew of the slaves. Furthermore, Figure 13(b) shows that Par3PKM has better speedup than Par2PK-Means and ParCLARA. The results demonstrate that the parallel algorithm Par3PKM has a very good speedup performance, which is almost the same for data sets with very different sizes.

6.5. Evaluation on Scale-Up. To evaluate how well the Par3PKM algorithm processes larger data sets when more nodes are available, we perform scale-up experiments where we increase the size of data sets (varying from 160 MB to 1280 MB) in direct proportion to the number of nodes (ranging from 1 node to 8 nodes) and then plot the results in Figure 14(a). Furthermore, with the Iris data sets (varying from 160 MB to 1280 MB), the scale-up comparison of Par3PKM, Par2PK-Means, and ParCLARA is depicted in Figure 14(b).

The scale-up metric [52, 53] is given by

$$\text{Scale-up} = \frac{T_s}{\widetilde{T}_p}, \tag{10}$$

where T_s is the execution time of an algorithm for processing the given data sets on one node and $\widetilde{T}_p$ is the execution time of an algorithm for handling p-times larger data sets on p-times larger nodes.

As illustrated in Figure 14(a), the scale-up values of Par3PKM are in the vicinity of 1, even less, with the proportional growth of both the number of nodes and the size of data sets. Moreover, Figure 14(b) shows that Par3PKM has better scale-up than Par2PK-Means and ParCLARA. The results indicate that the Par3PKM algorithm has very excellent scale-up and adaptability in large-scale data sets based on a Hadoop cluster with MapReduce.

6.6. Evaluation on Reliability. To evaluate the reliability of the Par3PKM algorithm, we shut down several nodes (ranging from 1 node to 7 nodes) to demonstrate whether Par3PKM can normally execute and achieve the same clustering results from Iris data sets with 1280 MB and then plot the results in Figure 12(b).

As illustrated in Figure 12(b), although the execution time of the Par3PKM algorithm increases gradually with the growth of the number of faulty nodes, the Par3PKM algorithm still normally executes and produces the same results. The results show that the Par3PKM algorithm has good reliability in "Big Data" environment, due to the high fault tolerance of the MapReduce framework on a Hadoop platform. When a node cannot execute tasks on a Hadoop cluster, the JobTracker will automatically assign the tasks of faulty node(s) to other spare nodes. Conversely, the serial K-Means algorithm on a single machine cannot normally execute the tasks when the machine is faulty, whereas the entire computational task will fail.

In summary, extensive experiments are conducted on real and synthetic data, and the performance results demonstrate that the proposed Par3PKM algorithm is much more efficient and accurate with better speedup, scale-up, and reliability.

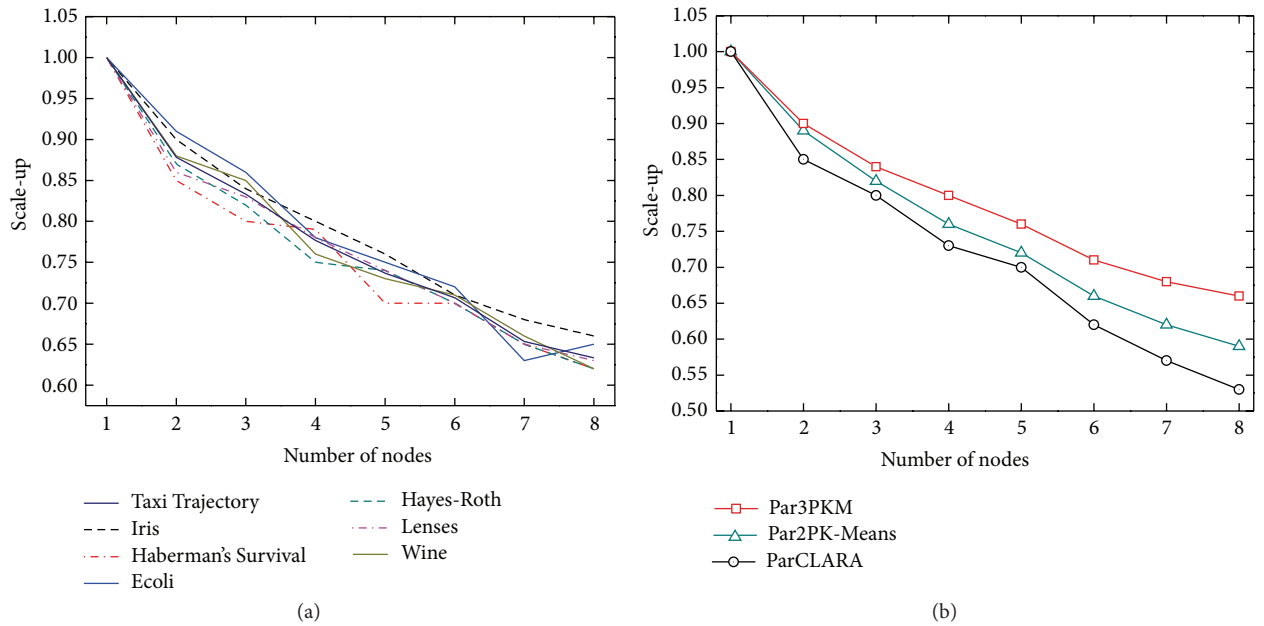

Figure 14: Scale-up. (a) Scale-up comparison of Par3PKM on different data sets and (b) scale-up comparison of Par3PKM, Par2PK-Means, and ParCLARA.

7. Conclusions

In this paper, we have proposed an efficient MapReduce-based parallel clustering algorithm, named Par3PKM, to solve traffic subarea division problem with large-scale taxi trajectories. In Par3PKM, the distance metric and initialization strategy of K-Means are optimized in order to enhance the accuracy of clustering. Then, to improve the efficiency and scalability of Par3PKM, the optimal K-Means algorithm is implemented in a MapReduce framework on Hadoop. The optimization and parallelism of Par3PKM save memory consumption and reduce the computational cost of big calculations, thereby significantly improving the accuracy, efficiency, scalability, and reliability of traffic subarea division. Our performance evaluation indicates that the proposed algorithm can efficiently cluster a large number of GPS trajectories of taxicabs and especially achieves more accurate results than K-Means, Par2PK-Means, and ParCLARA with favorably superior performance. Furthermore, based on Par3PKM, we have presented a distributed traffic subarea division method, named DTSAD, which is performed on a Hadoop distributed computing platform with the MapReduce parallel processing paradigm. In DTSAD, the boundary identifying method can effectively connect the borders of clustering results. Most importantly, we have divided traffic subarea of Beijing using big real-world taxi trajectory data sets through the presented method, and our case study demonstrates that our approach can accurately and efficiently divide traffic subarea.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Authors' Contribution

Dawen Xia and Binfeng Wang contributed equally to this work.

Acknowledgments

The authors would like to thank the academic editor and the anonymous reviewers for their valuable comments and suggestions. This work was partially supported by the National Natural Science Foundation of China (Grant no. 61402380), the Scientific Project of State Ethnic Affairs Commission of the People's Republic of China (Grant no. 14GZZ012), the Science and Technology Foundation of Guizhou (Grant no. LH20147386), and the Fundamental Research Funds for the Central Universities (Grants nos. XDJK2015B030 and XDJK2015D029).

References

[1] Y. Qi and S. Ishak, "Stochastic approach for short-term freeway traffic prediction during peak periods," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 660–672, 2013.

[2] A. de Palma and R. Lindsey, "Traffic congestion pricing methodologies and technologies," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 6, pp. 1377–1399, 2011.

[3] V. Marx, "The big challenges of big data," *Nature*, vol. 498, no. 7453, pp. 255–260, 2013.

[4] D. Agrawal, P. Bernstein, E. Bertino et al., "Challenges and opportunities with big data: a community white paper developed by leading researchers across the United States," White Paper, 2012.

[5] "Special online collection: dealing with data," *Science*, vol. 331, no. 6018, pp. 639–806, 2011.

[6] "Big data: science in the petabyte era," *Nature*, vol. 455, no. 7209, pp. 1–136, 2008.

[7] R. R. Weiss and L. Zgorski, *Obama Administration Unveils 'Big Data' Initiative: Announces $200 Million in New R&D Investments*, Office of Science and Technology Policy, Executive Office of the President, 2012.

[8] J. Manyika, M. Chui, B. Brown et al., "Big data: the next frontier for innovation, competition, and productivity," Tech. Rep., McKinsey Global Institute, 2011.

[9] Y. Genovese and S. Prentice, "Pattern-based strategy: getting value from big data," Gartner Special Report, 2011.

[10] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie, "T-finder: a recommender system for finding passengers and vacant taxis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2390–2403, 2013.

[11] J. Yuan, Y. Zheng, C. Zhang et al., "T-drive: driving directions based on taxi trajectories," in *Proceedings of the 18th International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS '10)*, pp. 99–108, ACM, San Jose, Calif, USA, November 2010.

[12] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, Waltham, Mass, USA, 3rd edition, 2011.

[13] D. Pelleg and A. Moore, "X-means: extending K-means with efficient estimation of the number of clusters," in *Proceedings of the 17th International Conference on Machine Learning (ICML '00)*, pp. 727–734, Stanford, Calif, USA, June 2000.

[14] S. Kantabutra and A. L. Couch, "Parallel K-means clustering algorithm on NOWs," *NECTEC Technical Journal*, vol. 1, no. 6, pp. 243–247, 2000.

[15] Y. Zhang, Z. Xiong, J. Mao, and L. Ou, "The study of parallel K-means algorithm," in *Proceedings of the 6th World Congress on Intelligent Control and Automation (WCICA '06)*, pp. 5868–5871, IEEE, Dalian, China, June 2006.

[16] P. Kraj, A. Sharma, N. Garge, R. Podolsky, and R. A. McIndoe, "ParaKMeans: implementation of a parallelized K-means algorithm suitable for general laboratory use," *BMC Bioinformatics*, vol. 9, article 200, 13 pages, 2008.

[17] M. K. Pakhira, "Clustering large databases in distributed environment," in *Proceedings of the IEEE International Advance Computing Conference (IACC '09)*, pp. 351–358, Patiala, India, March 2009.

[18] K. J. Kohlhoff, V. S. Pande, and R. B. Altman, "K-means for parallel architectures using all-prefix-sum sorting and updating steps," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 8, pp. 1602–1612, 2013.

[19] C.-T. Chu, S. K. Kim, Y.-A. Lin et al., "Map-reduce for machine learning on multicore," in *Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS '06)*, pp. 281–288, Vancouver, Canada, 2006.

[20] W. Zhao, H. Ma, and Q. He, "Parallel K-means clustering based on MapReduce," in *Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09)*, pp. 674–679, Beijing, China, December 2009.

[21] P. Zhou, J. Lei, and W. Ye, "Large-scale data sets clustering based on MapReduce and Hadoop," *Journal of Computational Information Systems*, vol. 7, no. 16, pp. 5956–5963, 2011.

[22] C. D. Nguyen, D. T. Nguyen, and V.-H. Pham, "Parallel two-phase K-means," in *Proceedings of the 13th International Conference on Computational Science and Its Applications (ICCSA '13)*, pp. 24–27, Ho Chi Minh City, Vietnam, June 2013.

[23] R. J. Walinchus, "Real-time network decomposition and subnetwork interfacing," Tech. Rep. HS-011 999, Highway Research Record, 1971.

[24] S. C. Wong, W. T. Wong, C. M. Leung, and C. O. Tong, "Group-based optimization of a time-dependent TRANSYT traffic model for area traffic control," *Transportation Research Part B: Methodological*, vol. 36, no. 4, pp. 291–312, 2002.

[25] D. I. Robertson and R. D. Bretherton, "Optimizing networks of traffic signals in real time: the SCOOT method," *IEEE Transactions on Vehicular Technology*, vol. 40, no. 1, pp. 11–15, 1991.

[26] Y.-Y. Ma and X.-G. Yang, "Traffic sub-area division expert system for urban traffic control," in *Proceedings of the International Conference on Intelligent Computation Technology and Automation (ICICTA '08)*, pp. 589–593, Hunan, China, October 2008.

[27] K. Lu, J.-M. Xu, S.-J. Zheng, and S.-M. Wang, "Research on fast dynamic division method of coordinated control subarea," *Acta Automatica Sinica*, vol. 38, no. 2, pp. 279–287, 2012.

[28] K. Lu, J.-M. Xu, and S.-J. Zheng, "Correlation degree analysis of neighboring intersections and its application," *Journal of South China University of Technology*, vol. 37, no. 11, pp. 37–42, 2009.

[29] H. Guo, J. Cheng, Q. Peng, C. Zhu, and Y. Mu, "Dynamic division of traffic control sub-area methods based on the similarity of adjacent intersections," in *Proceedings of the IEEE 17th International Conference on Intelligent Transportation Systems (ITSC '14)*, pp. 2208–2213, Qingdao, China, October 2014.

[30] C. Li, Y. Xie, H. Zhang, and X.-L. Yan, "Dynamic division about traffic control sub-area based on back propagation neural network," in *Proceedings of the 2nd International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC '10)*, pp. 22–25, Nanjing, China, August 2010.

[31] Z. Zhou, S. Lin, and Y. Xi, "A fast network partition method for large-scale urban traffic networks," *Journal of Control Theory and Applications*, vol. 11, no. 3, pp. 359–366, 2013.

[32] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, University of California Press, Berkeley, Calif, USA, 1967.

[33] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[34] X. Wu, V. Kumar, J. R. Quinlan et al., "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[35] W. Zou, Y. Zhu, H. Chen, and X. Sui, "A clustering approach using cooperative artificial bee colony algorithm," *Discrete Dynamics in Nature and Society*, vol. 2010, Article ID 459796, 16 pages, 2010.

[36] D. T. Pham, S. S. Dimov, and C. D. Nguyen, "An incremental K-means algorithm," *Proceedings of the Institution of Mechanical Engineers Part C: Journal of Mechanical Engineering Science*, vol. 218, no. 7, pp. 783–794, 2004.

[37] D. T. Pham, S. S. Dimov, and C. D. Nguyen, "A two-phase K-means algorithm for large datasets," *Journal of Mechanical Engineering Science*, vol. 218, no. 10, pp. 1269–1273, 2004.

[38] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proceedings of the 26th Symposium*

on Mass Storage Systems and Technologies (MSST '10), pp. 1–10, IEEE, Incline Village, Nev, USA, May 2010.

[39] A. Ene, S. Im, and B. Moseley, "Fast clustering using MapReduce," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11)*, pp. 681–689, San Diego, Calif, USA, August 2011.

[40] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, Sebastopol, Calif, USA, 3rd edition, 2012.

[41] D. Xia, Z. Rong, Y. Zhou, Y. Li, Y. Shen, and Z. Zhang, "A novel parallel algorithm for frequent itemsets mining in massive small files datasets," *ICIC Express Letters Part B: Applications*, vol. 5, no. 2, pp. 459–466, 2014.

[42] D. Xia, Z. Rong, Y. Zhou, B. Wang, Y. Li, and Z. Zhang, "Discovery and analysis of usage data based on Hadoop for personalized information access," in *Proceedings of the IEEE 16th International Conference on Computational Science and Engineering—Big Data Science and Engineering (CSE-BDSE '13)*, pp. 917–924, IEEE, Sydney, Australia, December 2013.

[43] D. Xia, B. Wang, Z. Rong, Y. Li, and Z. Zhang, "Effective methods and strategies for massive small files processing based on Hadoop," *ICIC Express Letters*, vol. 8, no. 7, pp. 1935–1941, 2014.

[44] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pp. 29–43, Bolton Landing, NY, USA, October 2003.

[45] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[46] P. Zikopoulos, C. Eaton, D. deRoos, T. Deutsch, and G. Lapis, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, McGraw-Hill, New York, NY, USA, 2011.

[47] W. K. D. Pun and A. B. M. S. Ali, "Unique distance measure approach for K-means (UDMA-Km) clustering algorithm," in *Proceedings of the IEEE Region 10 Conference (TENCON '07)*, pp. 1–4, IEEE, Taipei, Taiwan, November 2007.

[48] A. M. Fahim, A. M. Salem, F. A. Torkey, and M. A. Ramadan, "An efficient enhanced K-means clustering algorithm," *Journal of Zhejiang University SCIENCE A*, vol. 7, no. 10, pp. 1626–1633, 2006.

[49] M. Zhu, *Data Mining*, University of Science and Technology of China Press, 2002.

[50] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, 1990.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS '12)*, pp. 1097–1105, Lake Tahoe, Nev, USA, December 2012.

[52] S. Englert, J. Gray, T. Kocher, and P. Shah, "A benchmark of NonStop SQL release 2 demonstrating near-linear speedup and scaleup on large databases," *ACM SIGMETRICS Performance Evaluation Review*, vol. 18, no. 1, pp. 245–246, 1990.

[53] X. Xu, J. Jäger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," *Data Mining and Knowledge Discovery*, vol. 3, no. 3, pp. 263–290, 1999.