Hindawi
Complexity
Volume 2018, Article ID 2818251, 16 pages
https://doi.org/10.1155/2018/2818251

WILEY | Hindawi

Research Article

A MapReduce-Based Parallel Frequent Pattern Growth Algorithm for Spatiotemporal Association Analysis of Mobile Trajectory Big Data

Dawen Xia,[1,2] **Xiaonan Lu**,[1] **Huaqing Li**,[3] **Wendong Wang**,[2] **Yantao Li**,[2] **and Zili Zhang**[2,4]

[1]*College of Data Science and Information Engineering and College of National Culture and Cognitive Science, Guizhou Minzu University, Guiyang 550025, China*
[2]*College of Computer and Information Science, Southwest University, Chongqing 400715, China*
[3]*College of Electronic and Information Engineering, Southwest University, Chongqing 400715, China*
[4]*School of Information Technology, Deakin University, Geelong, VIC 3216, Australia*

Correspondence should be addressed to Dawen Xia; dwxia@gzmu.edu.cn and Huaqing Li; huaqingli@hotmail.com

Received 14 June 2017; Accepted 13 November 2017; Published 28 January 2018

Academic Editor: Michele Scarpiniti

Frequent pattern mining is an effective approach for spatiotemporal association analysis of mobile trajectory big data in data-driven intelligent transportation systems. While existing parallel algorithms have been successfully applied to frequent pattern mining of large-scale trajectory data, two major challenges are how to overcome the inherent defects of Hadoop to cope with taxi trajectory big data including massive small files and how to discover the implicitly spatiotemporal frequent patterns with MapReduce. To conquer these challenges, this paper presents a MapReduce-based Parallel Frequent Pattern growth (MR-PFP) algorithm to analyze the spatiotemporal characteristics of taxi operating using large-scale taxi trajectories with massive small file processing strategies on a Hadoop platform. More specifically, we first implement three methods, that is, Hadoop Archives (HAR), CombineFileInputFormat (CFIF), and Sequence Files (SF), to overcome the existing defects of Hadoop and then propose two strategies based on their performance evaluations. Next, we incorporate SF into Frequent Pattern growth (FP-growth) algorithm and then implement the optimized FP-growth algorithm on a MapReduce framework. Finally, we analyze the characteristics of taxi operating in both spatial and temporal dimensions by MR-PFP in parallel. The results demonstrate that MR-PFP is superior to existing Parallel FP-growth (PFP) algorithm in efficiency and scalability.

1. Introduction

In the era of data technology (DT) with "Internet +" and "big data ×," large-scale data has been growing rapidly with 5Vs characteristics (i.e., Volume, Velocity, Variety, Value, and Veracity) [1–7]. Actually, big data usually consists of massive small files in various practical applications such as mobile trajectory data, financial and investment data, business transaction data, and health trajectory data [8]. Recently, mobile trajectory big data analytics has been a research hotspot of urban computing and smart cities, which attracts great attention from the industry, academia, and government [9–11]. In particular, taxi trajectory data is becoming one of the most significant data sources of mobile trajectory data. For example, the big taxi trajectory data used in this work is composed of 47,991 small files (each file with a size of 1.1 MB) [12] and contains a large number of GPS points which can generate the road network of Beijing as illustrated in Figure 1.

Taxi trajectory data records the movement traces and operating status of taxicabs, and to a certain extent it reflects the urban transportation conditions and includes the potentially rich driving experience of drivers [13, 14]. In particular, traffic conditions are extremely uncertain in a transportation network due to the heterogeneous and dynamic nature of traffic with nonlinear interactions between drivers and environments [13]. To facilitate the perception

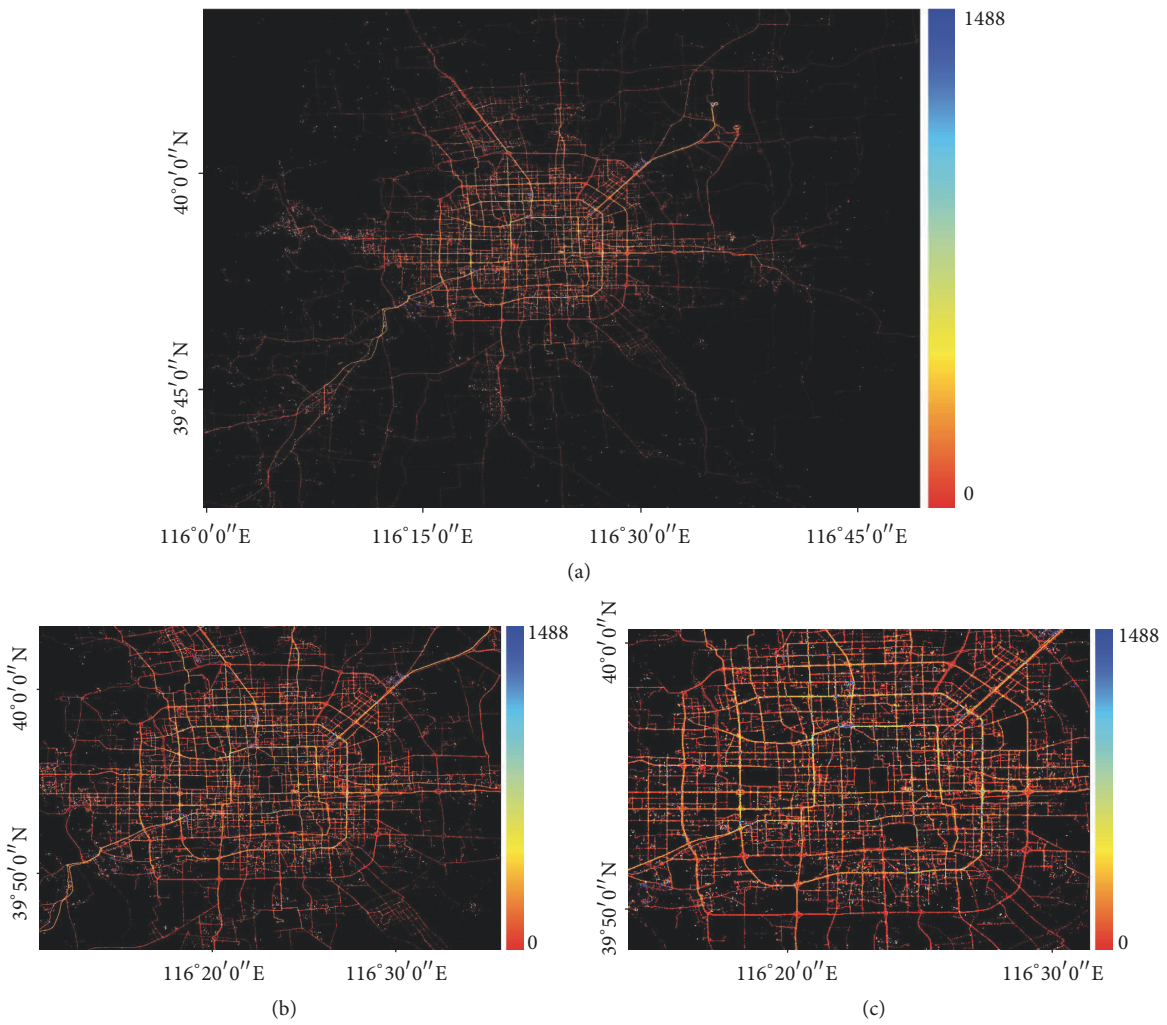

Figure 1: Distribution of the GPS points (1,232,048 records) generated by 12,000 taxicabs in Beijing at 00:14:35 AM-01:14:34 AM on 1 November 2012, where the color denotes the density of the points. (a) Data overview in Beijing. (b) Within the 5th Ring Road and (c) the 4th Ring Road of Beijing. Figure reproduced from Xia et al. (2016) [13]. Copyright ©2016 IEEE, with permission.

of traffic conditions and develop management strategies and monitoring programs, the temporal and spatial characteristics of taxi operating are analyzed and the frequent patterns in the trajectory data are excavated based on the multidimensional association analysis of trajectory big data. Moreover, the interestingly strong association rules are found among the frequent itemsets of operational status, operating time, geographical position, and running speed and direction, which can provide decision making for the passengers' travel, taxicab operation scheduling, and urban transportation planning and construction.

In data-driven intelligent transportation systems (ITS), the spatiotemporal pattern discovery of taxi trajectory big data is an efficient method for urban transportation optimization and control in smart cities. Typically, existing classical frequent pattern mining algorithms mainly utilize the breadth-first techniques (e.g., Apriori [15]) and the depth-priority techniques (e.g., FP-growth [16]). To improve mining efficiency, some optimized methods have been put forward to improve the aforementioned algorithms [17–19]. Furthermore, to break the bottleneck of mining performance on a single machine, serval parallel or distributed algorithms are presented [20, 21]. However, by using the traditional implementation methodologies in the stand-alone environment, the applications of corresponding sequential algorithms for mining frequent pattern from large-scale datasets will easily result in high memory consumption, high I/O cost, low computation performance, and other problems.

Nowadays, the Apache Hadoop project provides open-source software for reliable, scalable, and distributed computing and allows for the distributed processing of large-scale datasets across clusters of computers with straightforward programming models [13]. In particular, Hadoop Distributed File System (HDFS) [22] and Hadoop MapReduce [23] are the key components of Hadoop, and the Hadoop distributed computing platform [24] with a MapReduce parallel processing framework provides a new idea for handling big data [25]. In recent years, to meet the rapidly growing demands of large-scale data processing, a Parallel FP-growth (PFP) algorithm based on MapReduce has been proposed in [26], and the

performance of PFP has been enhanced through adding load balancing feature in [27]. Although all the aforementioned methods have many desirable properties, these approaches still ignore frequent itemset mining in large-scale datasets with massive small files on a Hadoop platform using the MapReduce framework. In addition, Hadoop has inherent defects in processing massive small files such as slow reading and writing, high memory consumption, low computational efficiency, and other issues. The massive small file processing problem has been the prominent bottleneck of Hadoop's computation performance [24, 28].

To tackle the above problems, this paper aims to present a MapReduce-based Parallel Frequent Pattern growth (MR-PFP) algorithm and implement its applications in spatiotemporal association analysis of taxi operating characteristics with big trajectory data on a Hadoop distributed computing platform. In particular, MR-PFP can reduce the additional overhead of MapReduce and improve the access efficiency of HDFS by incorporating the small file processing strategies, which overcomes the existing defects of Hadoop in handling massive small files. On the other hand, MR-PFP is implemented on a MapReduce parallel processing framework through the *Map* function and the *Reduce* function, which can improve the efficiency and scalability of frequent itemset mining significantly.

The contributions of this work are summarized as follows:

(i) Three approaches (HAR, CFIF, and SF) are implemented to overcome the inherent defects of high memory consumption and I/O cost and low computational efficiency in processing massive small files for Hadoop. More importantly, with the experimental results, two selection strategies are presented to meet different requirements.

(ii) A MapReduce-based Parallel Frequent Pattern growth (MR-PFP) algorithm is proposed to find frequent itemsets in parallel by exploiting the processing strategy associated with massive small files (i.e., SF), thereby improving the efficiency and scalability of generating association rules.

(iii) MR-PFP is applied to the spatiotemporal association analysis of taxi operating characteristics using taxi trajectory big data on a Hadoop platform. The extensive performance evaluation indicates that MR-PFP is superior to the Parallel Frequent Pattern growth (PFP) algorithm in efficiency and scalability and can address the frequent itemset mining problem with large-scale datasets composed of massive small files to generate association rules efficiently.

The remainder of this paper is organized as follows. Section 2 introduces the motivation of this work by the problem statement and its formulation. The proposed MR-PFP algorithm is described in detail in Section 3. Section 4 presents the applications of MR-PFP in spatiotemporal association analysis of taxi operating characteristics via a case study. The extensive experiments and results for performance evaluation are shown in Section 5. Section 6 concludes this paper.

2. Problem Statement

In this section, we analyze the existing problems of frequent itemset mining in large-scale datasets with massive small files for generating interesting association rules and then give the corresponding problem definition.

2.1. Problem Analysis. The size of small files is commonly less than 64 MB which is the default size of a data block in HDFS. Naturally, various scientific applications contain a large number of small files in practice. For instance, the big taxi trajectory data used in this work is composed of 47,991 small files and each file is with a size of 1.1 MB. Furthermore, according to the study results reported in the National Energy Research Scientific Computing Center (NERSC), 43% of over 13 million files in a shared parallel file system are under 64 KB and 99% are under 64 MB (http://www.nersc.gov/users/storage-and-file-systems/file-systems). Another study at the Pacific Northwest National Laboratory (PNNL) shows that a system includes 12 million files, 94% of which are under 64 MB and 58% of which are under 64 KB [29]. However, the FP-tree constructed by the Parallel FP-growth (PFP) algorithm based on MapReduce cannot be stored in memory when it comes to big datasets with massive small files, which usually leads to memory overflow, huge communication overhead (or large traffic), and other problems.

On the other hand, Hadoop was originally designed to cope with large streaming files and thus stores and manages massive small files inefficiently. Storing a large number of small files leads to high memory consumption and unacceptable access cost [30]. There are inherent defects in reducing the access efficiency of HDFS and increasing the additional overhead of MapReduce when processing massive small files. Moreover, the computational efficiency of Hadoop depends on the performance of two key components (i.e., HDFS and MapReduce) [24]. For these reasons, the processing of massive small file will reduce the overall performance of association rule mining on a Hadoop platform, which is mainly shown in the following two aspects:

(i) The access efficiency of HDFS is reduced. Namenode is responsible for managing and scheduling huge amounts of metadata stored in each Datanode node, and each file, directory, and block is encapsulated into a 150-byte space stored in the memory of Namenode [31], which needs to query and retrieve the requested file blocks between Datanodes with lots of searches and hops. To this end, massive small files will occupy a large amount of Namenode limited available memory space and consume a lot of execution time, which will result in inefficient data access of HDFS significantly. Storing and managing massive small files pose a big challenge to HDFS [30].

(ii) The additional overhead of MapReduce is increased. MapReduce jobs consist of multiple Map tasks and Reduce tasks, and the Map tasks commonly execute an input block at a time. If the file is too small in size and numerous in number, a large number of Map tasks will be generated and each Map task

will only handle very small data, which will lead to an additional increase of MapReduce computational tasks inevitably. For example, a 2 GB file is equally divided into 32 files (i.e., each file with a size of 64 MB) or 2^{21} files (i.e., each file with a size of 1 KB). We can draw a conclusion that the execution time of the latter will be significantly longer than the former by comparative analysis. The reason is that the former requires only 32 Map tasks to complete the data processing task, while the latter needs 2^{21} Map tasks to operate.

To improve the performance of massive small file processing for Hadoop, MacKey et al. [32] used the Hadoop Archives method and the scheduling mechanisms to compress the metadata with massive small files. It can reduce Namenode's memory consumption and improve HDFS's system performance, but file access is quite slow [33]. Mohandas and Thampi [34] put forward a solution which integrated the Hadoop Archives and the Sequence Files to solve the massive small file processing problem, but it was not successfully implemented on applications. However, these solutions still have some limitations and cannot effectively solve existing inherent defects of Hadoop in dealing with massive small files, especially for association rule mining of big datasets with massive small files on Hadoop using MapReduce.

2.2. Problem Definition. Association rule mining is utilized to reveal the unknown interdependence among the data and discover lots of interesting association (or correlation) relationships among a large set of data items [35]. It can be decomposed into the following two subproblems:

(I) Find all frequent itemsets that have *support* greater than the user-specified minimum *support* threshold (i.e., *min_sup*) from the transaction database. That is, $support(i) \geq min\_sup$, i is a frequent itemset.

(II) Generate the desired rules from the frequent itemsets found in (I). If *confidence* satisfies the user-specified minimum *confidence* threshold (i.e., *min_conf*), strong association rules are generated. That is, $confidence(A \Rightarrow B) \geq min\_conf$: $(A \Rightarrow B)$ is a strong association rule.

From the above-mentioned problems, we can observe that the overall performance of association rule mining depends on the first subproblem (i.e., the key is to find all the frequent itemsets). That is, the problem of mining association rules can be attributed to discovering frequent itemsets, because it is quite simple and straightforward to check whether the rules are strong.

Based on a concrete analysis of the aforementioned problem, some definitions used in this work are given as follows.

Definition 1. Let $I = \{I_1, I_2, \ldots, I_m\}$ be a set of items. Let $D = \{T_1, T_2, \ldots, T_n\}$ be a set of database transactions, where each transaction T is a set of items such that $T \subseteq I$. That is, $T_i = \{I_{i1}, I_{i2}, \ldots, I_{ik(i)}\}$ and $I_{ij} \in I$, $1 \leq j \leq k(i)$, $1 \leq i \leq n$. Let A be a set of items, and a transaction T is said to contain A if and only if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \emptyset$.

Definition 2. Rule *support* and *confidence* are two measures of rule interestingness that reflect the usefulness and certainty of discovered rules, respectively. The rule $A \Rightarrow B$ has *support* s and *confidence* c in the transaction set D, where s is the percentage of transactions in D that contains $A \cup B$, and that is the probability $P(A \cup B)$; c is the percentage of transactions in D containing A which also contains B, and that is the conditional probability $P(B \mid A)$. The *support* of A is the percentage of transactions in D containing the itemset A, and that is $support(A) = \{T_i \mid 1 \leq i \leq n \wedge A \subseteq T_i \wedge T_i \in D\}$.

More formally, *support* and *confidence* are defined as

$$\begin{aligned} support\,(A) &= \frac{support\_count\,(A)}{count\,(D)} \\ support\,(A \Longrightarrow B) &= P\,(A \cup B) \\ &= \frac{support\_count\,(A \cup B)}{count\,(D)} \\ confidence\,(A \Longrightarrow B) &= P\,(B \mid A) \\ &= \frac{support\_count\,(A \cup B)}{support\_count\,(A)}. \end{aligned} \tag{1}$$

Definition 3. When the *support* and *confidence* satisfy the *min_sup* and the *min_conf*, respectively, that is, $support(A \Rightarrow B) \geq min\_sup$ and $confidence(A \Rightarrow B) \geq min\_conf$, typically the association rule is called interestingly strong.

3. Proposed MR-PFP Algorithm

In this section, we propose a MapReduce-based Parallel Frequent Pattern growth (MR-PFP) algorithm to implement the parallelism of association rule mining in big dataset with massive small files on a Hadoop platform by incorporating massive small file processing strategies.

Based on the analysis of the aforementioned problem and the problem definition, the MR-PFP algorithm (see Figure 2) is presented to address the problem of spatiotemporal association analysis by mining frequent itemsets with large-scale datasets including massive small files and particularly to overcome the inherent defects of slow reading and writing, high memory consumption, and low computational efficiency in processing massive small files for Hadoop.

More specifically, we first implement the HAR, CFIF, and SF methods to handle massive small files on a Hadoop platform (see Section 3.2) and experimentally evaluate three approaches in terms of memory consumption and execution efficiency to put forward two strategies of massive small file processing (see Section 5.3). Furthermore, based on the selection strategies, we exploit the SF method (see Section 3.2.2) to optimize the frequent pattern growth algorithm by efficiently integrating massive small files into a large sequence file (big transaction data file or transaction database). Finally, to reduce the computational complexity of MapReduce jobs and

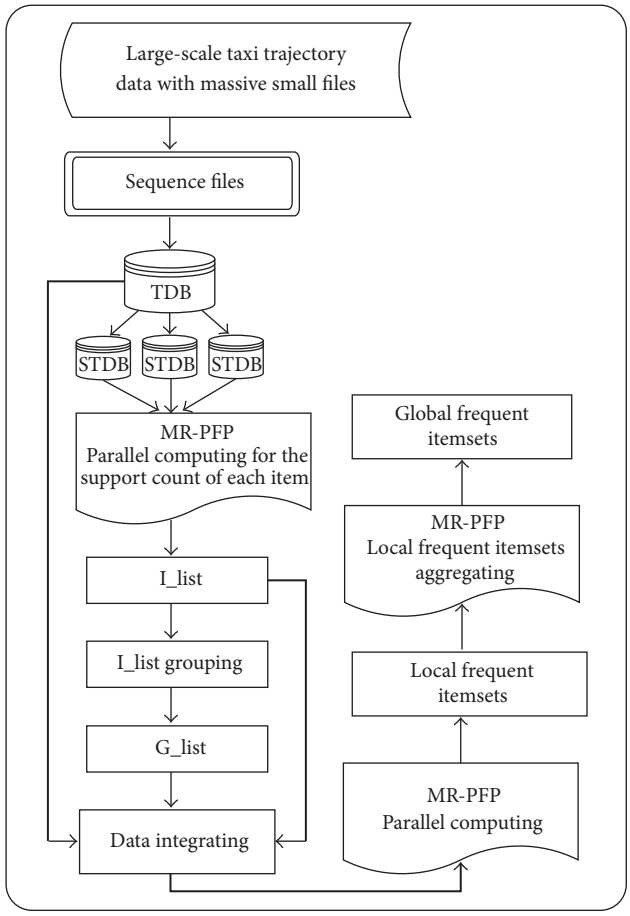

Figure 2: The flowchart of the MR-PFP algorithm.

save the limited bandwidth available on a Hadoop cluster, we implement the optimized frequent pattern growth algorithm with the MapReduce paradigm to generate the interestingly strong association rules by mining frequent itemsets in parallel (see Section 3.3).

3.1. Overview of Algorithm. Based on PFP, the massive small file processing strategy is incorporated into MR-PFP which is implemented with the MapReduce paradigm for mining frequent itemsets in parallel to overcome the inherent defects of Hadoop in handling big datasets associated with large-scale small files, thereby generating interestingly strong association rules.

As illustrated in Figure 2, the flowchart of the proposed MR-PFP algorithm mainly consists of the following six steps.

Step 1 (integrating massive small files). Based on massive small file processing strategies (see Section 5.3.3), we utilize the *Sequence Files* (SF, see Section 3.2.2) method to integrate all the small files which contain a large number of transaction datasets stored in HDFS into a large transaction data file (i.e., transaction database).

Step 2 (dividing transaction database). We equally divide a transaction database into several subtransaction databases and then assign them to different nodes of a Hadoop cluster. We employ the *balance* command to enable its file system to fulfil load balancing, when necessary.

Step 3 (calculating the *support* count of each item). We compute the *support* count for each item of subtransaction databases in parallel using the MapReduce paradigm, then sort them by *support* count in a descending order, and finally attain a set of *I_list*.

Step 4 (grouping *I_list*). We divide *I_list* into *M* sets in order to form *G_list* and sequentially assign *G_id* for each set and each *G_list* includes a set of items.

Step 5 (generating local frequent itemsets). We implement the parallel computing of FP-growth under a MapReduce framework by the *Map* function and the *Reduce* function which are performed in the Map phase and the Reduce phase, respectively, to generate local frequent itemsets. More specifically, the *Map* function compares the item of each transaction in the subtransaction databases with the item in *G_list*. Provided that they are the same, the corresponding transaction then is assigned to the machine associated with *G_list*; otherwise, the *Map* function compares to the next item in *G_list*. Finally, the independent subtransaction databases corresponding to *G_list* will be produced. The *Reduce* function recursively calculates the independent subtransaction databases generated in the Map phase and then constructs the FP-tree. The aforementioned computation method is similar to the traditional FP-tree generation process, except that the heap of size *K* (*HP*) is used to store frequent pattern of each item. In particular, the time-consuming recursive procedure is accomplished in the MapReduce model of computation.

Step 6 (obtaining global frequent itemsets). We aggregate the local frequent itemsets generated from each node on a Hadoop cluster through the MapReduce paradigm and then obtain the global frequent itemsets.

3.2. Massive Small File Processing Strategies. In this work, we implement three methods of Hadoop Archives (HAR), Sequence Files (SF), and CombineFileInputFormat (CFIF) and put forward two selection strategies to solve the massive small file processing problem on Hadoop using MapReduce. Additionally, the effectiveness of the implemented methods and the reasonability of the selected strategies are to be evaluated in Section 5.3. Based on the evaluation results, we choose the SF method to handle the massive small files in MR-PFP.

3.2.1. Hadoop Archives. Hadoop Archive (HAR) is a file archiving facility, which efficiently archives massive small files into HDFS blocks via the *archive* commands [33]. A Hadoop Archive has a *.har extension, which can be used as an input to MapReduce. In the HAR file system, the process of reading massive small files mainly contains the following three steps.

(i) Access the *Master Index* to gain the *Index* stored in memory index.

(ii) Access the *Index* to get the *Store Index* of the file itself.

(iii) Access the *Store Index* to obtain the content of the file.

3.2.2. Sequence Files. Sequence File (SF) is a method that integrates massive small files into a large sequence file via the following three classes.

(i) *WholeFileInputFormat Class*: the *isSplitable* () method overloads and returns the value, "false," to maintain the input file not to be partitioned into splits. The *getRecordReader* () method returns a customized *RecordReader*.

(ii) *WholeFileRecordReader Class*: the *FileSplit* is converted into a record, where the *key* of the record is the filename and the *value* is the content of the file. In view of the existence of only one record after the conversion, *WholeFileRecordReader* only deals with the record or is discarded. However, it uses a Boolean variable, *processed*, to indicate whether the record was processed. If the *next* () method is called, it indicates that the file is not processed. At the same time, the file will be opened and then generate a byte array with a size of the file length. Subsequently, it calls the Hadoop's IOUtils class and takes the content of the file into the byte array. Finally, an array is created on the BytesWritable instance that is passed to the *next* () method. If the return value is true, it demonstrates that the record has been successfully read.

(iii) *SmallFilesToSequenceFileConverter Class*: massive small files are integrated into a sequence file, and this class consists of the *Map* () and the *Reduce* (). The input format of data is *WholeFileInputFormat*, while the output format is *SequenceFileOutputFormat*.

3.2.3. CombineFileInputFormat. CombineFileInputFormat (CFIF) is a new type based on the *InputFormat* method which packages multiple files into a split to deal with more data by each Map task, thereby avoiding the drawbacks of traditional *FileInputFormat* that must generate a split for each file. Unlike the SF method, the CFIF method does not integrate massive small files but only produces less InputSplit, to achieve the packaging of massive small files through employing the following two classes.

(i) *CombineFileLineRecordReader Class*: packaging multiple files then will be processed by the Map tasks.

(ii) *MyMultiFileInputFormat Class*: the *CreateRecordReader*() method is implemented and the abstract class of CFIF is inherited, and then the implementation of the customized Recorder is returned.

3.3. Implementation on MapReduce. To improve the efficiency and scalability, we implement the parallelism of the proposed MR-PFP algorithm on a MapReduce framework using the *Map* function and the *Reduce* function. In the MapReduce environment, MR-PFP is mainly composed of the following four steps.

Input:
  key: *key*,
  value: T_i.
Output:
  key: a_i,
  value: 1.
(1) **for** each item a_i in T_i **do**
(2) Output(a_i, 1);
(3) **end for**
(4) **return** ⟨*key*, *value*⟩ pairs;

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

Input:
  key: a_i,
  value: list(a_i).
Output:
  key: a_i,
  value: *sum*.
(1) Int *sum* = 0;
(2) **for** each item a_i in T_i **do**
(3) *Sum*+ = 1;
(4) **end for**
(5) **return** ⟨*key*, *value*⟩ pairs;

Algorithm 2: Reduce(*key*, *value*).

Step 1. Integrate massive small files into a large sequence file using the *Sequence Files* (SF) method (see Section 3.2.2), which employs the *WholeFileInputFormat Class*, *WholeFileRecordReader Class*, and *SmallFilesToSequenceFileConverter Class*. Moreover, *SF* includes a series of ⟨key, value⟩, where the *key* is the name of small files and the *value* is the content of small files before integrating.

Step 2. Calculate *I_list* through the *Map* function and the *Reduce* function, which are formally depicted in Algorithms 1 and 2, respectively.

Step 3. Generate *G_list* based on *I_list*, and perform the parallel computing of the MR-PFP algorithm. *Map*() judges the *G_list* that the *Item* of transactions in the machine belongs to, and then this transaction is assigned to the machine corresponding to *G_list*. To avoid sending the same transaction to the same machine repeatedly, we delete the duplicate entries in a hash table. *Reduce*() processes the independent subtransaction database associated with *G_list*, which builds heap *HP* with a size of the *K* for each item in *G_list*. The *Map* function and the *Reduce* function are formally described in Algorithms 3 and 4, respectively.

Step 4. Aggregate the local frequent itemsets of each node generated from Step 3, and finally gain the global frequent itemsets. The *Map* function and the *Reduce* function are illustrated in Algorithms 5 and 6, respectively.

Input:
key: *key*,
value: T_i.
Output:
key: *HashNum*,
value: T_i.
(1) *I_list* generates *G_list*;
(2) Create Hash Table *H*;
(3) Create a[];
(4) $a[]$ = Split(T_i);
(5) **for** $j = Length(a) - 1, j \geq 0, j - -$ **do**
(6) HashNum = getHashNum(*H*, *a[j]*);
(7) **if** *HashNum* ≠ *Null* **then**
(8) **if** *HashNum* in *H* **then**
(9) Delete this item;
(10) **end if**
(11) **end if**
(12) **end for**
(13) **return** ⟨*key*, *value*⟩ pairs;

Algorithm 3: Map(*key*, *value*).

Input:
key: *G_id*,
value: *TDB*(*G_id*).
Output:
key: v_i,
value: sup(v_i).
(1) Group = *G_list*;
(2) Clear LocalFP-tree;
(3) **for** each T_i in *TDB*(*G_id*) **do**
(4) Create FP-tree(LocalFP-tree, T_i);
(5) **for** each a_i in Group **do**
(6) Create heap *HP*(the Max_Size = *K*);
(7) MR-PFP(LocalFP-tree, a_i, *HP*);
(8) **for** each v_i in *HP* **do**
(9) Output(v_i, sup(v_i));
(10) **end for**
(11) **end for**
(12) **end for**
(13) **return** ⟨*key*, *value*⟩ pairs;

Algorithm 4: Reduce(*key*, *value*).

Input:
key: *v*,
value: sup(*v*).
Output:
key: a_i,
value: v + sup(*v*).
(1) **for** each item a_i in *v* **do**
(2) Output(a_i, v + sup(*v*));
(3) **end for**
(4) **return** ⟨*key*, *value*⟩ pairs;

Algorithm 5: Map(*key*, *value*).

Input:
key: a_i,
value: list(v + sup(*v*)).
Output:
key: *null*,
value: $a_i + c$.
(1) Create heap *HP*(the Max_Size = *K*);
(2) **for** each pattern *v* in v + sup(*v*) **do**
(3) **if** |*HP*| < *K* **then**
(4) insert v + sup(*v*) into *HP*;
(5) **else**
(6) **if** *sup*(*HP*[0] · *v*) < *sup*(*v*) **then**
(7) delete top element in *HP*;
(8) insert v + sup(*v*) into *HP*;
(9) **end if**
(10) **end if**
(11) **end for**
(12) **return** ⟨*key*, *value*⟩ pairs;

Algorithm 6: Reduce(*key*, *value*).

4. Case Study

This section presents the application of MR-PFP in spatiotemporal association analysis of taxi operating characteristics, based on real-world taxi trajectory big data associated with massive small files.

4.1. Dataset. In this case study, we use real-world trajectory dataset (http://www.datatang.com), which contains large-scale GPS trajectories recorded by 12,000 taxis in Beijing during a period of one month in November 2012 (i.e., between 1 November and 30 November 2012). The total size of the dataset is approximately 50 GB, and especially the total number of GPS records reaches 969,418,200 and the total distance of the dataset surpasses 50 million kilometers. In particular, the dataset is composed of 47,991 small files (i.e., each file with a size of 1.1 MB). The data sample of large-scale taxi trajectories used in this work is illustrated in Figure 3.

Moreover, with the aforementioned dataset, we utilize the proposed MR-PFP algorithm to analyze the spatiotemporal associations of taxi operating characteristics on a Hadoop distributed computing platform using the MapReduce parallel processing framework (see Section 5.1).

4.2. Association Analysis. Based on MR-PFP, we take *min_sup* = 0.3 and *min_conf* = 0.6 as an example to analyze the association among the speed, longitude, and latitude of taxi operating (i.e., the spatial dimension). Table 1 shows the interestingly strong association rules selected from the results where the value of the data items is in the form of range for rendering, due to the large scale of taxi trajectories and the large span of data-item values.

From Table 1, it could be found that the taxi operating characteristics are associated among the speed, longitude, and latitude (e.g., between longitude/speed and latitude, between longitude/latitude and speed, and between speed and longitude/latitude). The results indicate that MR-PFP can discover

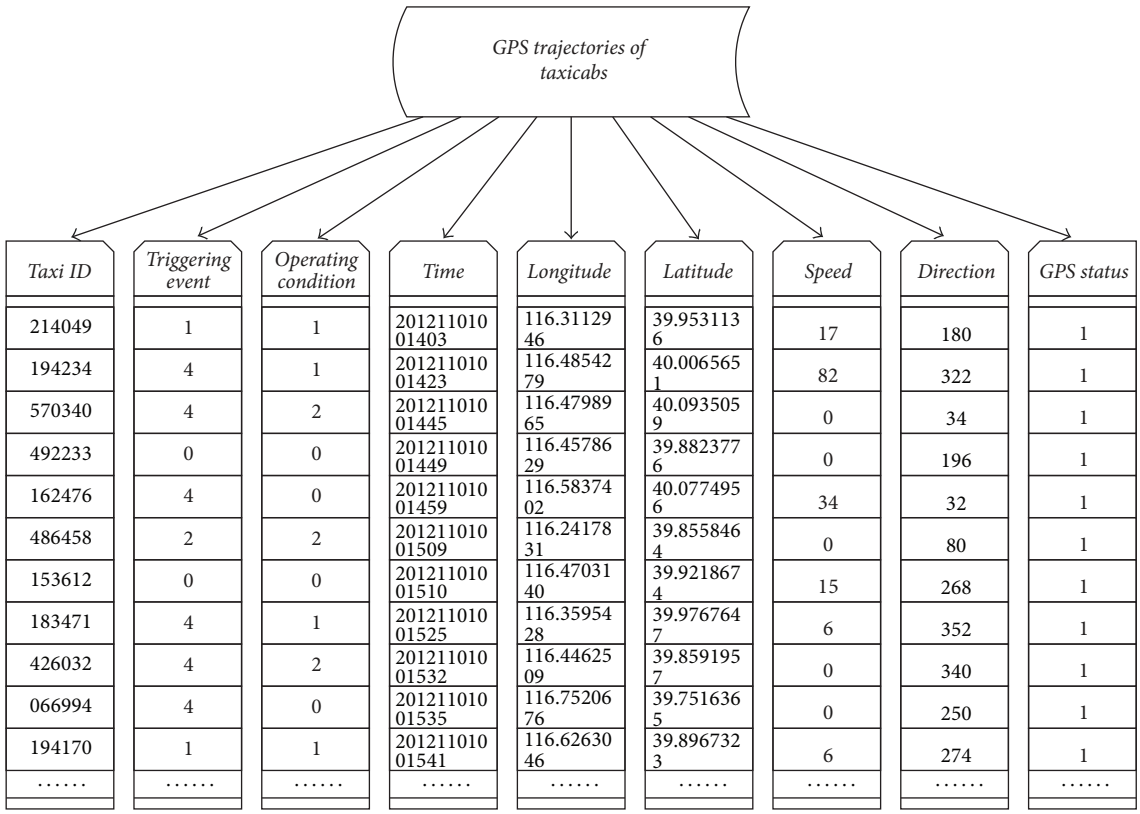

Figure 3: The data sample of large-scale taxi trajectories.

Table 1: Association rules in the spatial dimension.

Rules[1]	Confidence
GPS–lon = (116.305527–116.611407] ∧ GPS–s = (0–29.166667] ⇒ GPS–lat = (39.784276–40.234173]	0.89
GPS–lon = (116.305527–116.611407] ⇒ GPS–lat = (39.784276–40.234173]	0.92
GPS–lon = (116.305527–116.611407] ∧ GPS–lat = (39.784276–40.234173] ⇒ GPS–s = (0–29.166667]	0.63
GPS–lon = (115.846706–116.305527] ⇒ GPS–lat = (39.33438–40.009225]	0.87
GPS–lon = (116.305527–116.764347] ⇒ GPS–lat = (39.33438–40.009225]	0.81
GPS–lon = (116.305527–116.764347] ∧ GPS–lat = (39.33438–40.009225] ⇒ GPS–s = (0–43.75]	0.77
GPS–lon = (116.121999–116.489055] ⇒ GPS–lat = (39.604317–40.144194]	0.97
GPS–lon = (116.121999–116.489055] ∧ GPS–lat = (39.604317–40.144194] ⇒ GPS–s = (0–35]	0.70
GPS–s = (0–35] ⇒ GPS–lon = (116.121999–116.489055] ∧ GPS–lat = (39.604317–40.144194]	0.68
GPS–lat = (39.671802–40.009225] ⇒ GPS–s = (0–21.87]	0.66

[1]GPS–lon represents the longitude of GPS, GPS–lat denotes the latitude of GPS, and GPS–s is the speed of GPS, respectively. Here, GPS–lon = (116.305527–116.611407] indicates the longitude of GPS which is between 116.305527 and 116.611407.

frequent patterns from big dataset with massive small files in parallel, to analyze the multidimensional associations of taxi operating characteristics.

Moreover, to extensively analyze the spatiotemporal patterns of taxi operating characteristics, we select the trajectory data from 00:10:43 to 00:15:42 on 1 November 2012 and then mine its frequent itemsets using MR-PFP. The produced interestingly strong association rules are illustrated in Table 2, where we only choose the first 8 rules from each example.

As shown in Table 2, we take $min\_sup = 0.5$ and $min\_conf = 0.6$, $min\_sup = 0.45$ and $min\_conf = 0.6$, and $min\_sup = 0.46$ and $min\_conf = 0.3$ as three examples, respectively, to analyze the spatial and temporal associations of taxi operating characteristics among the speed, longitude, latitude, and time (i.e., the spatiotemporal dimensions). From these results, it could be observed that MR-PFP has the potential of frequent pattern mining in big trajectory data, thereby generating association rules which will be of great benefit to perceive real-time transportation conditions.

In summary, the results of case study show that MR-PFP can solve the small file problem in Hadoop and generate association rules in parallel using MapReduce and then

Table 2: Association rules in the spatiotemporal dimensions.

Rules[1]	*min_sup*	*min_conf*	Confidence
GPS–lon(116.3081207) ∧ GPS–lat(40.0850945) ⇒ GPS–t(20121101001456)	0.5	0.6	0.97
GPS–t(20121101001515) ⇒ GPS–lat(39.9745636)	0.5	0.6	0.82
GPS–s(13) ⇒ GPS–lat(39.8567123)	0.5	0.6	0.82
GPS–t(20121101001527) ∧ GPS–s(32) ⇒ GPS–lat(39.9517555)	0.5	0.6	0.82
GPS–t(20121101001505) ∧ GPS–lon(116.3136597) ⇒ GPS–lat(39.9311104)	0.5	0.6	0.81
GPS–s(36) ⇒ GPS–t(20121101001517) ∧ GPS–lat(39.887928)	0.5	0.6	0.81
GPS–lat(39.8962898) ⇒ GPS–s(41)	0.5	0.6	0.80
GPS–lat(39.9318008) ⇒ GPS–lon(116.3795013)	0.5	0.6	0.62
GPS–lon(116.3797607) ∧ GPS–lat(39.9029083) ⇒ GPS–t(20121101001526)	0.45	0.6	0.97
GPS–s(23) ⇒ GPS–lat(39.6109581)	0.45	0.6	0.82
GPS–lon(116.3812103) ⇒ GPS–lat(39.8552132)	0.45	0.6	0.81
GPS–t(20121101001452) ⇒ GPS–s(25)	0.45	0.6	0.80
GPS–lon(116.3844528) ⇒ GPS–t(20121101001524) ∧ GPS–lat(39.9866829)	0.45	0.6	0.79
GPS–lon(116.3852234) ⇒ GPS–s(39)	0.45	0.6	0.76
GPS–lat(39.9709435) ⇒ GPS–lon(116.3859177)	0.45	0.6	0.62
GPS–t(20121101001536) ∧ GPS–lat(39.9672508) ⇒ GPS–lon(116.3856812)	0.45	0.6	0.62
GPS–lon(116.3820038) ∧ GPS–lat(39.9646225) ⇒ GPS–t(20121101001450)	0.46	0.3	0.97
GPS–t(20121101001452) ∧ GPS–s(34) ⇒ GPS–lat(39.9438133)	0.46	0.3	0.82
GPS–lon(116.3782349) ⇒ GPS–lat(39.8775635)	0.46	0.3	0.81
GPS–lon(116.3766174) ⇒ GPS–t(20121101001509) ∧ GPS–lat(39.8671379)	0.46	0.3	0.79
GPS–t(20121101001453) ∧ GPS–lon(116.5092163) ⇒ GPS–s(30)	0.46	0.3	0.76
GPS–lat(39.9984131) ⇒ GPS–lon(116.5080185)	0.46	0.3	0.62
GPS–t(20121101001511) ∧ GPS–lat(39.8550682) ⇒ GPS–lon(116.4675217)	0.46	0.3	0.62
GPS–lat(39.8920479) ⇒ GPS–t(20121101001520) ∧ GPS–lon(116.4780731)	0.46	0.3	0.60

[1]GPS–lon represents the longitude of GPS, GPS–lat denotes the latitude of GPS, GPS–t indicates the time of GPS, and GPS–s is the speed of GPS.

implement the spatiotemporal association analysis of taxi operating characteristics based on real-world taxi trajectory big data with massive small files. The application can provide data support for the perception of traffic conditions, the development of management strategies and monitoring programs, as well as the reasonable travel of passengers, the operation schedule of the taxi and the transportation planning, and construction of the city.

5. Performance Evaluation

This section evaluates the proposed MR-PFP algorithm compared to PFP and then provides a concise and precise description of the experimental results, and finally their interpretation as well as the experimental conclusions will be drawn in detail.

In the extensive experiments, we evaluate the efficiency and scalability performances of MR-PFP in terms of speedup, scaleup, and sizeup and then validate the effectiveness of the implemented massive small file processing methods and the reasonability of the selection strategies.

5.1. Experimental Setup. In this experiment, we build a Hadoop platform composed of 1 Master and 10 Slaves with Intel Xeon (R) E7-4820 2.00 GHz CPU (4-cores) and 8.00 GB RAM, integrating with ArcGIS and road network of Beijing which consists of 106,579 road nodes and 141,380 road segments (http://www.datatang.com/data/43855). All the experiments are performed on Ubuntu 12.04 OS with Hadoop 1.0.4 and JDK 1.6.0. The experimental platform based on Hadoop with MapReduce, road network, and ArcGIS is shown in Figure 4.

Moreover, three real-world datasets from the taxi trajectory data (TTD, see Section 4.1), the National Climatic Data Center (NCDC) (ftp://ftp.ncdc.noaa.gov/pub/data/noaa), and the Frequent Itemset Mining Dataset Repository (FIMD) (http://fimi.ua.ac.be/data) are processed into three different sets of datasets (i.e., TTD: T-Dataset1, T-Dataset2, and T-Dataset3; NCDC: N-Dataset1, N-Dataset2, and N-Dataset3; FIMD: F-Dataset1, F-Dataset2, and F-Dataset3), respectively, and each set of datasets contains a large number of small files. The experimental datasets are shown in Table 3.

5.2. Evaluation on MR-PFP. Since MR-PFP can achieve exactly the same accuracy as PFP, we focus on evaluating their efficiency and scalability using three evaluation metrics (i.e., speedup, sizeup, and scaleup [12, 13, 36–38]) in this experiment.

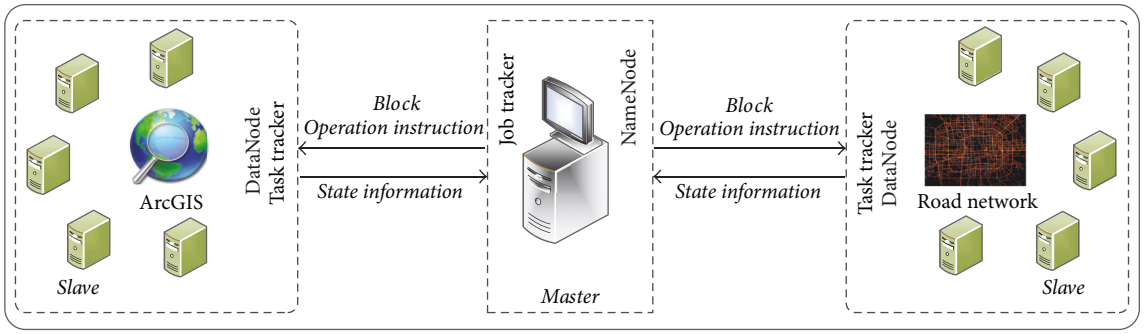

Figure 4: The experimental platform based on Hadoop with MapReduce, road network, and ArcGIS.

Table 3: The experimental datasets for performance evaluation of MR-PFP.

Datasets	Number of small files	Size of each small file	Size of all the files
T-Dataset1	931	=1.1 MB	1 GB
T-Dataset2	1862	=1.1 MB	2 GB
T-Dataset3	3724	=1.1 MB	4 GB
N-Dataset1	936	<1.1 MB	1 GB
N-Dataset2	1864	<1.1 MB	2 GB
N-Dataset3	3730	<1.1 MB	4 GB
F-Dataset1	2115	<64 KB	128 MB
F-Dataset2	4218	<64 KB	256 MB
F-Dataset3	8583	<64 KB	512 MB

5.2.1. Efficiency. To examine the efficiency of MR-PFP, in comparison with PFP, we carry out the experiments on a cluster with 4 nodes for frequent itemset mining to generate association rules using nine sets of datasets (i.e., T-Dataset1, T-Dataset2, T-Dataset3, N-Dataset1, N-Dataset2, N-Dataset3, F-Dataset1, F-Dataset2, and F-Dataset3) and then plot the results in Figure 5. Furthermore, for a more intuitive illustration, the comparisons of MR-PFP and PFP on execution time are depicted in Figure 6.

Figures 5 and 6 show that MR-PFP significantly outperforms PFP in efficiency under a MapReduce framework on a Hadoop platform. In particular, with the number of nodes and the size of datasets growing, the advantages of MR-PFP on MapReduce are more remarkable.

5.2.2. Speedup. The speedup metric evaluates how much the parallel algorithm is faster than the corresponding sequential algorithm, which is defined as

$$Speedup = \frac{T_1}{T_p}, \tag{2}$$

where T_1 is the sequential execution time of the algorithm on 1 node for association rule mining with the given dataset and T_p is the parallel execution time of the algorithm for addressing the same problem using the same dataset on a cluster with p nodes.

For the speedup evaluation of MR-PFP, we carry out the experiments on a cluster of nodes varying from 1 to 10 (i.e., the number of cores ranging from 4 to 40) by holding six sets of datasets (i.e., 128 MB, 256 MB, 512 MB, 1 GB, 2 GB, and 4 GB) constant, respectively. The results are depicted in Figure 7(a).

From Figure 7(a), we can find that the speedup of MR-PFP increases relatively linearly with the growth of the number of nodes, and especially larger dataset obtains a better speedup. The speedup value of the TDD (taxi trajectory data) dataset with 4 GB (i.e., T-Dataset3) reaches 7.985, which is 79.85% (7.985/10 = 79.85%) of the ideal speedup, when the number of nodes is 10. Generally, it is very hard to achieve linear speedup due to the communication cost and the skew of the slaves [36], and particularly the time to cope with small-scale datasets is not dominant compared to the time consumed by communication and task arrangement. When the size of datasets is gradually increased, the time of intensive computation becomes significantly dominant. The results demonstrate that the proposed MR-PFP algorithm on MapReduce has a very good speedup performance over big data and performs better with larger datasets, which is nearly the same for datasets with extremely different sizes.

5.2.3. Scaleup. The scaleup metric validates how well the parallel algorithm deals with larger datasets when more nodes are available, which can be defined as

$$Scaleup = \frac{T_1}{\widetilde{T}_p}, \tag{3}$$

where T_1 denotes the sequential execution time of the algorithm for handling the given dataset on 1 node and $\widetilde{T}_p$ represents the parallel execution time of the algorithm for processing p-times larger datasets on p-times larger nodes.

To validate the scaleup of MR-PFP, we increase the number of nodes (ranging from 1 node to 10 nodes) in direct proportion to the size of datasets (varying from 128 MB to 1.25 GB, from 256 MB to 2.5 GB, and from 1 GB to 10 GB) and then plot the results in Figure 7(b).

From Figure 7(b), we can observe that all the scaleup values of MR-PFP are greater than 0.37, with the proportional growth of both the number of nodes and the size of datasets. The results indicate that the proposed MR-PFP algorithm on

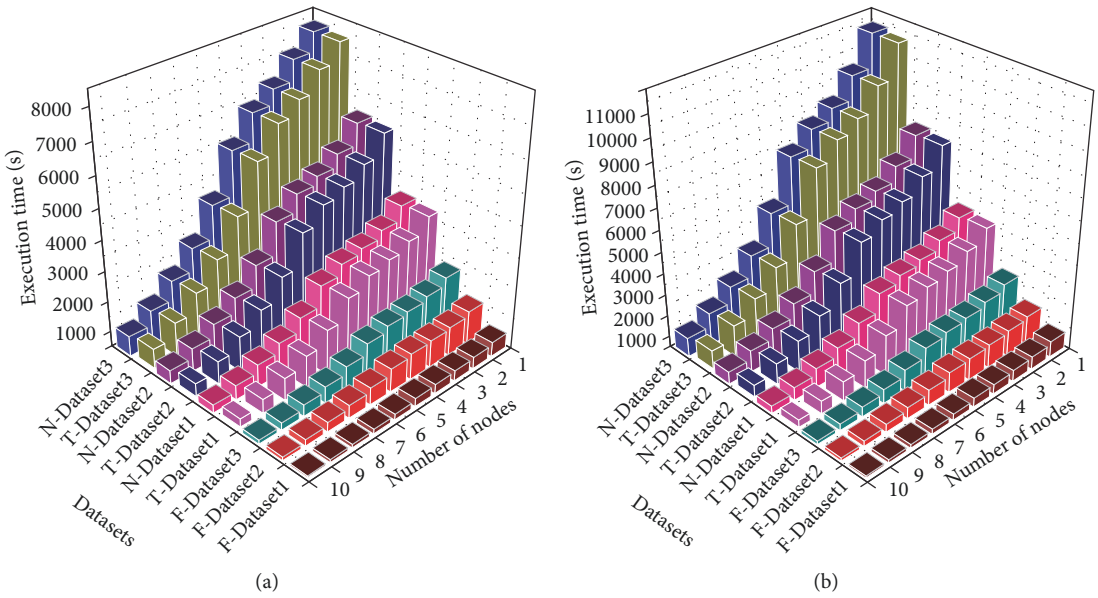

Figure 5: Execution time. (a) MR-PFP and (b) PFP.

MapReduce scales very well and has the adaptability of big datasets.

5.2.4. Sizeup. The sizeup metric measures how much longer the parallel algorithm takes on a given node, when the size of datasets is m-times larger than the original dataset. It is given by

$$Sizeup = \frac{\widetilde{T}_x}{T_x}, \tag{4}$$

where T_x represents the execution time of the algorithm for copying with the given dataset on the given node and $\widetilde{T}_x$ denotes the execution time of the algorithm for processing p-times larger datasets on the same node. Here, sizeup analysis is to keep the number of nodes constant and grow the size of the datasets by the factor p, to measure the variation of execution time.

To measure the sizeup of MR-PFP, we perform the experiments on a cluster with 10 nodes by increasing the size of datasets (ranging from 128 MB to 1.25 GB, from 256 MB to 2.5 GB, and from 1 GB to 10 GB) and then plot the results in Figure 7(c).

From Figure 7(c), we can conclude that the proposed MR-PFP algorithm on MapReduce has a very good sizeup performance. More specifically, a 10 times larger problem needs approximately 10 to 12 times more time.

5.2.5. Results Analysis. Based on the above-mentioned results, it could be found that the parallel MR-PFP algorithm implements the distributed computing and parallel processing of massive small files on a MapReduce framework, thereby improving the overall performance of frequent itemset mining to generate association rules on a Hadoop platform. More specifically, the massive small file processing strategy incorporated in MR-PFP overcomes the inherent defects of Hadoop in handling large-scale datasets with massive small files, significantly reducing memory consumption and I/O cost for enhancing data access efficiency and avoiding memory overflow, and thus greatly improves the performance of the PFP algorithm.

5.3. Evaluation on Massive Small File Processing Strategies. In the experimental evaluation, we utilize the memory consumption of Namenode and the execution time of MapReduce to evaluate the validity of Hadoop-based massive small file processing method (SF) which is incorporated into MR-PFP and to investigate the reasonability of selected strategy in comparison with the other two approaches (i.e., HAR and CFIF).

Furthermore, we use HAR, SF, and CFIF methods to deal with three sets of datasets (N-Dataset1, N-Dataset2, and N-Dataset3), respectively. Subsequently, we record the execution time of each set of datasets and compute the number of files and indexes in HDFS through executing the MaxTemperature program on MapReduce [24].

5.3.1. Memory Consumption. Based on the statistics of the number of files and indexes in HDFS using the method in [24], we calculate the memory consumption of Namenode after processing three sets of datasets through three implemented methods, respectively, and then plot the results in Figure 8(a).

From Figure 8(a), we make the conclusions as follows: (I) In the HAR method, massive small files are archived to *.har file which has two indexes (i.e., Master Index and Index) stored in Namenode, so the memory space consumed Namenode depends on the number of two indexes. (II) In the SF method, massive small files are integrated into a large sequence file which has an index of the Index stored

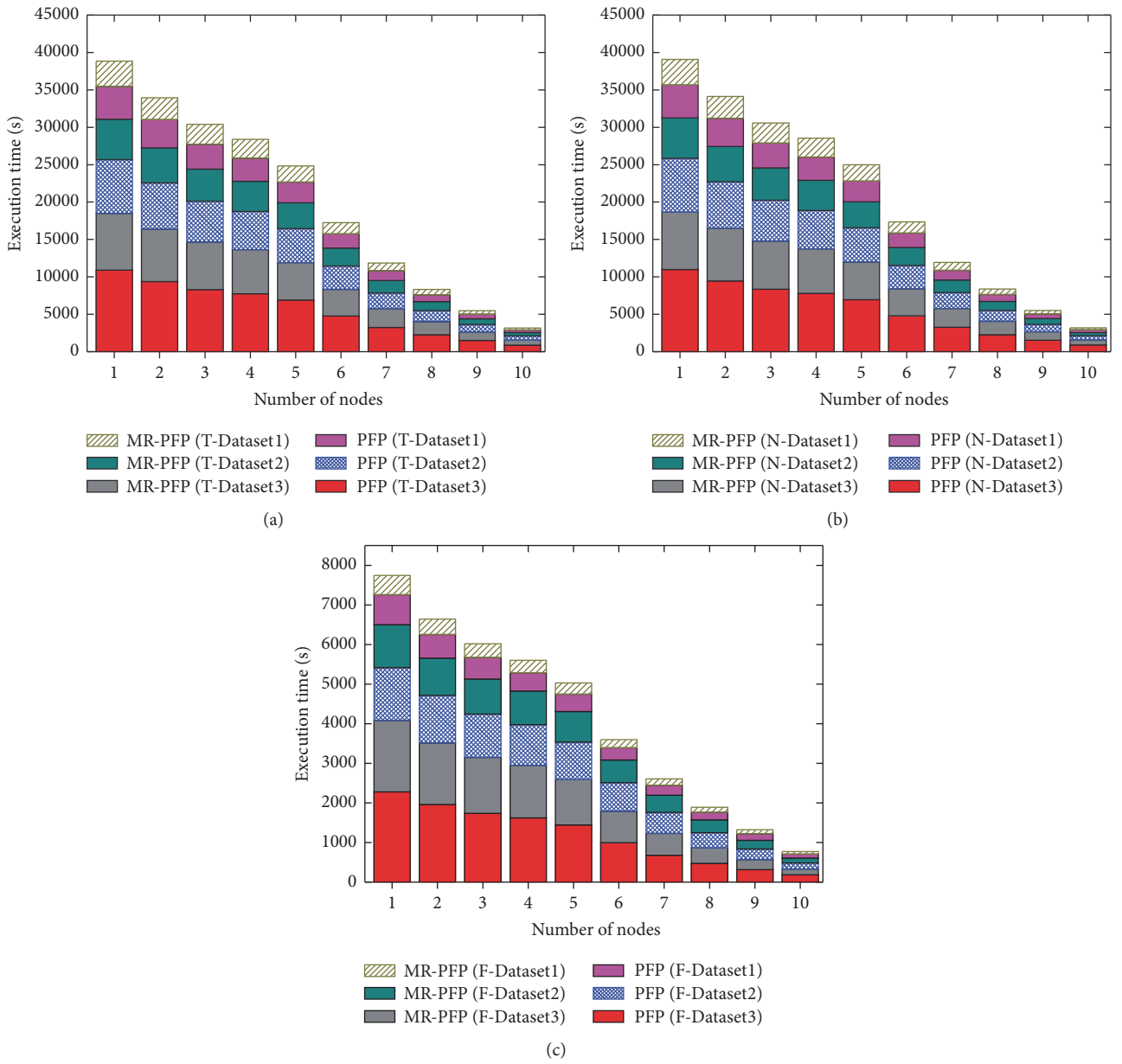

Figure 6: Execution time of MR-PFP and PFP. (a) T-Datasets, (b) N-Datasets, and (c) F-Datasets.

in Namenode, so the memory space consumed Namenode lies on the number of this Index. (III) In the CFIF method, massive small files are packaged into a InputSplit instead of integrating into a sequence file before the MapReduce tasks, and massive small files are still in the form of a single file stored in HDFS, so the memory space consumed Namenode is determined by the number of files.

5.3.2. Execution Time. In this experiment, we record the execution time of three sets of datasets processed by three implemented methods in MapReduce, respectively. The results are depicted in Figure 8(b).

From Figure 8(b), we make the following analysis: (I) The HAR method greatly reduces the memory consumption of Namenode. The reduction of the Namenode's memory load will shorten the HDFS's deployment time, but the reading time of massive small files is slower than that read directly from HDFS. (II) The SF method significantly reduces the memory consumption of Namenode and shortens the scheduling time of HDFS. The time recorded in the experiments is the execution time of the entire mining process. That is, the time includes the execution time of nonmining task (e.g., the time in the process of integrating a large number of small files into a sequence file is counted as the execution time of the entire task). However, it results in the appearance of a longer execution time using this method. (III) The CFIF method packages a large number of small files into a single InputSplit which is processed by a single Map task, thereby reducing the time consumption of startup and shutdown in the Map task.

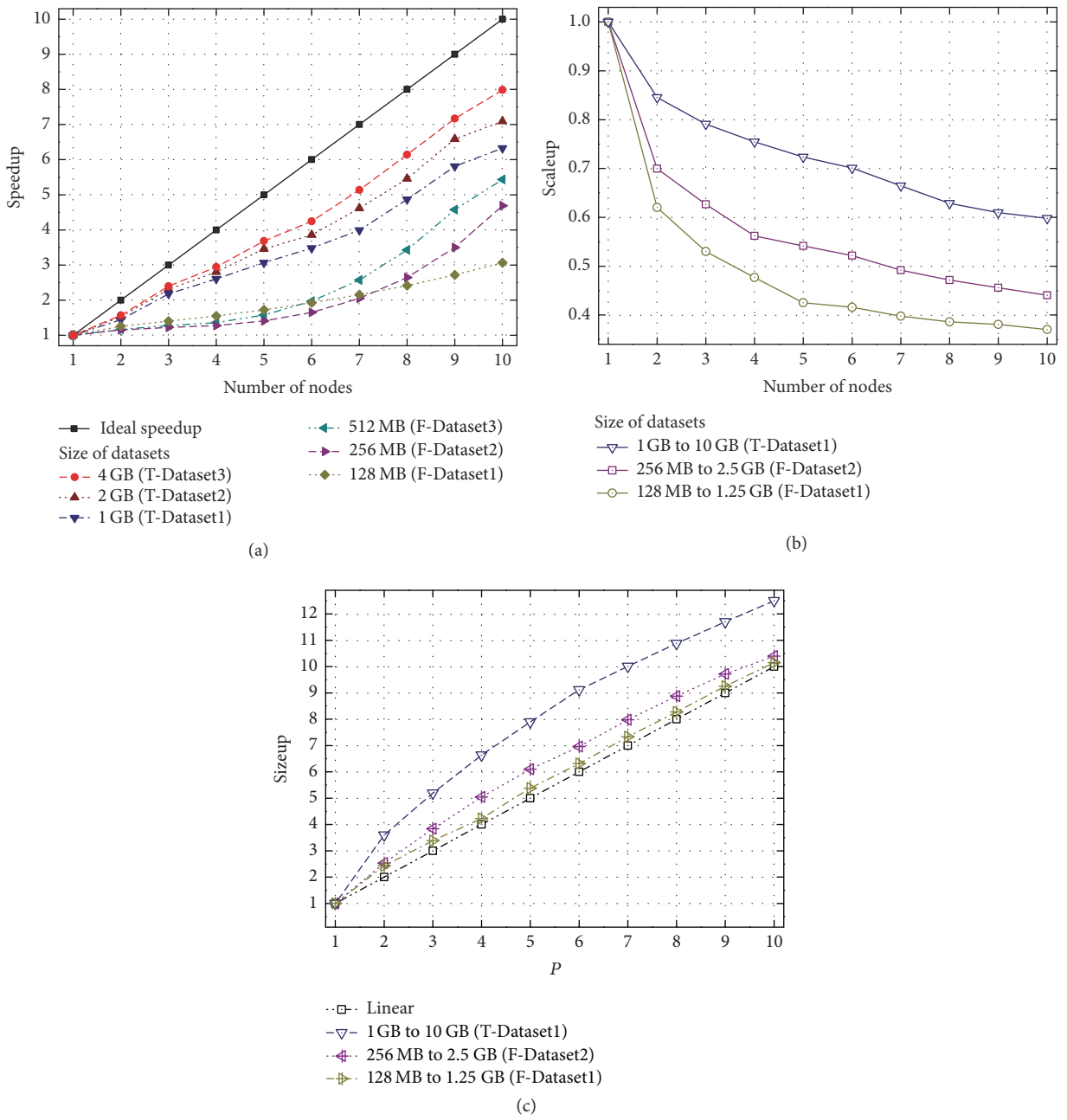

Figure 7: Speedup, scaleup, and sizeup of MR-PFP. (a) Speedup, (b) scaleup, and (c) sizeup.

5.3.3. Results Analysis. According to the aforementioned experimental results, we can observe that the implemented HAR, SF, and CFIF methods have the potential to efficiently process massive small files, thereby overcoming the inherent defects of Hadoop. More importantly, we have drawn the following conclusions: When massive small files are handled by three methods, the memory consumption of Namenode in an ascending order is SF, HAR, and CFIF, while the execution time of MapReduce in an ascending order is CFIF, HAR, and SF. In particular, the computational efficiency of MapReduce based on mining tasks in an ascending order is SF, CFIF, and HAR.

Based on the analysis mentioned above, it could be found that two different strategies of massive small file processing can be selected as follows: (I) Providing that the memory consumption is an important factor in controlling the performance of the calculation (and focusing on the efficiency of the mining task), the SF method should be selected in advance when we cope with massive small files on a Hadoop platform. (II) Providing that the execution efficiency of the entire mining process is considered, we should give priority to the CFIF method. Nevertheless, to avoid the phenomena of memory overflow and huge communication overhead, we process a large number of small files using the SF method in MR-PFP

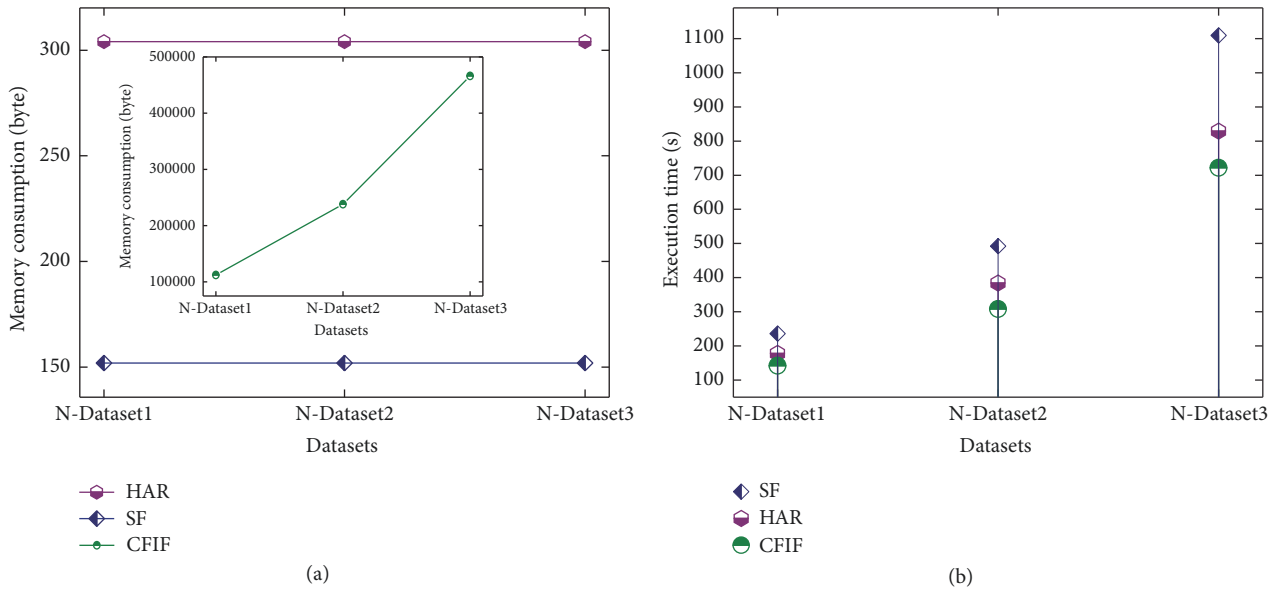

Figure 8: Memory consumption of Namenode and execution time of MapReduce for HAR, SF, and CFIF. (a) Memory consumption of Namenode and (b) execution time of MapReduce.

to address the problems of high memory consumption and I/O cost and especially implement the parallelism of frequent itemset mining in MR-PFP to solve time-consuming data transmission, low computational performance, and other issues using the MapReduce paradigm.

6. Conclusions and Future Work

In this paper, we proposed a MapReduce-based Parallel Frequent Pattern growth (MR-PFP) algorithm using massive small file processing strategies and particularly applied it to analyze the spatiotemporal patterns of taxi operating characteristics with big trajectory data on a Hadoop distributed computing platform. Moreover, with the extensive experiments on real-world big datasets with large-scale small files, we evaluated the performance of MR-PFP in terms of efficiency and scalability and then verified the effectiveness of three methods of massive small file processing and the reasonability of two selection strategies. Briefly speaking, based on a MapReduce parallel processing framework, MR-PFP can make up the inherent defects of Hadoop in handling big datasets associated with massive small files and reduce memory consumption and save I/O cost, thereby improving data access efficiency and avoiding memory overhead. The experimental results also demonstrated that, in comparison with PFP, MR-PFP had higher efficiency and better speedup, scaleup, and sizeup performance.

Although the spatiotemporal association analysis of mobile trajectory big data with massive small files can be addressed much more efficiently via MR-PFP on Hadoop using MapReduce, there are still several issues for further investigation. When the *I_list* is grouped in MR-PFP, the highly frequent *Items* may be allocated in a *G_list* node so that their transactions are extremely large and the computational tasks are too high, while the computational costs of other nodes are relatively low, thereby resulting in the imbalanced load problem inevitably. In future work, we will further study the *I_list* grouping algorithm and consider adding two strategies of the group balancing and the load balancing in MR-PFP.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant nos. 61762020, 61773321, and 61528206), the High-Level Innovative Talents Project of Guizhou (Grant no. QRLF201621), the Science and Technology Top-Notch Talents Support Project of Colleges and Universities in Guizhou (Grant no. QJHKY2016065), the Science and Technology Foundation of Guizhou (Grant nos. QKHJC20161076 and QKHJZ20142094), the Youth Science and Technology Talents Development Project of Guizhou (Grant nos. QJHKY2017129 and QJHKY2017137), the Natural Science Foundation of Chongqing CSTC (Grant nos. cstc2014jcyjA40016, cstc2015jcyjA40044, and cstc2014jcyjA40030), the Special Financial Grant from the China Postdoctoral Science Foundation (Grant no. 2017T100670), the China Postdoctoral Science Foundation (Grant no. 2016M590852), and the Fundamental Research Funds for the Central Universities (Grant nos. XDJK2016B016 and XDJK2015B030). The authors would like to acknowledge Datatang (Beijing) Technology Co., Ltd., for providing the experimental data and thank Dr. Zhiju Xie for her great help in polishing the language.

References

[1] V. Marx, "Biology: the big challenges of big data," *Nature*, vol. 498, no. 7453, pp. 255–260, 2013.

[2] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[3] J. Zhang, F.-Y. Wang, K. Wang, W.-H. Lin, X. Xu, and C. Chen, "Data-driven intelligent transportation systems: a survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1624–1639, 2011.

[4] E. I. Vlahogianni, B. B. Park, and J. W. C. van Lint, "Big data in transportation and traffic engineering," *Transportation Research Part C: Emerging Technologies*, vol. 58, p. 161, 2015.

[5] Y. Xia, L. Zhang, and Y. Liu, "Special issue on big data driven Intelligent Transportation Systems," *Neurocomputing*, vol. 181, pp. 1–3, 2016.

[6] X.-W. Chen and X. Lin, "Big data deep learning: challenges and perspectives," *IEEE Access*, vol. 2, pp. 514–525, 2014.

[7] J. Klüver, J. Schmidt, and C. Klüver, "Word morph and topological structures: a graph generating algorithm," *Complexity*, vol. 21, no. S1, pp. 426–436, 2016.

[8] B. Castellani, R. Rajaram, J. Gunn, and F. Griffiths, "Cases, clusters, densities: modeling the nonlinear dynamics of complex health trajectories," *Complexity*, vol. 21, no. S1, pp. 160–180, 2016.

[9] Y. Zheng, "Trajectory data mining: an overview," *ACM Transactions on Intelligent Systems and Technology*, vol. 6, no. 3, article 29, 2015.

[10] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: concepts, methodologies, and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 5, no. 3, article 38, 2014.

[11] G. Pan, G. Qi, W. Zhang, S. Li, Z. Wu, and L. T. Yang, "Trace analysis and mining for smart cities: issues, methods, and applications," *IEEE Communications Magazine*, vol. 51, no. 6, pp. 120–126, 2013.

[12] D. Xia, B. Wang, H. Li, Y. Li, and Z. Zhang, "A distributed spatial-temporal weighted model on MapReduce for short-term traffic flow forecasting," *Neurocomputing*, vol. 179, pp. 246–263, 2016.

[13] D. Xia, H. Li, B. Wang, Y. Li, and Z. Zhang, "A MapReduce-based nearest neighbor approach for big-data-driven traffic flow prediction," *IEEE Access*, vol. 4, pp. 2920–2934, 2016.

[14] D. Xia, B. Wang, Y. Li, Z. Rong, and Z. Zhang, "An efficient MapReduce-based parallel clustering algorithm for distributed traffic subarea division," *Discrete Dynamics in Nature and Society*, vol. 2015, Article ID 793010, 2015.

[15] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*, vol. 1215, pp. 487–499, Santiago de Chile, Chile, September 1994.

[16] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 1–12, 2000.

[17] G. Grahne, L. V. S. Lakshmanan, and X. Wang, "Efficient mining of constrained correlated sets," in *Proceedings of the 16th International Conference on Data Engineering (ICDE '00)*, pp. 512–521, IEEE, San Diego, CA, USA, March 2000.

[18] Y.-J. Tsay, T.-J. Hsu, and J.-R. Yu, "FIUT: a new method for mining frequent itemsets," *Information Sciences*, vol. 179, no. 11, pp. 1724–1737, 2009.

[19] K.-C. Lin, I.-E. Liao, and Z.-S. Chen, "An improved frequent pattern growth method for mining association rules," *Expert Systems with Applications*, vol. 38, no. 5, pp. 5154–5161, 2011.

[20] S. K. Tanbeer, C. F. Ahmed, and B.-S. Jeong, "Parallel and distributed algorithms for frequent pattern mining in large databases," *IETE Technical Review*, vol. 26, no. 1, pp. 55–66, 2009.

[21] E. T. Wang and A. L. Chen, "Mining frequent itemsets over distributed data streams by continuously maintaining a global synopsis," *Data Mining and Knowledge Discovery*, vol. 23, no. 2, pp. 252–299, 2011.

[22] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST '10)*, pp. 1–10, IEEE, Incline Village, NV, USA, May 2010.

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

[24] T. White, Hadoop: The Definitive Guide, 3rd ed., O'Reilly Media, Inc., Sebastopol, Calif, USA, 2012.

[25] 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 (CSE '13) Joint with the IEEE 2nd International Conference on Big Data Science and Engineering (BDSE '13)*, pp. 917–924, IEEE, Sydney, Australia, December 2013.

[26] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang, "PFP: parallel FP-growth for query recommendation," in *Proceedings of the ACM International Conference on Recommender Systems (RecSys '08)*, pp. 107–114, ACM, Lausanne, Switzerland, October 2008.

[27] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Z. Huang, and S. Feng, "Balanced parallel FP-growth with MapReduce," in *Proceedings of the 2010 IEEE Youth Conference on Information Computing and Telecommunications (YC-ICT '10)*, pp. 243–246, IEEE, Beijing, China, November 2010.

[28] X. Li, B. Dong, L. Xiao, L. Ruan, and Y. Ding, "Small files problem in parallel file system," in *Proceedings of the International Conference on Network Computing & Information Security (NCIS '11)*, pp. 227–232, IEEE, Guilin, China, May 2011.

[29] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig, "Small-file access in parallel file systems," in *Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS '09)*, pp. 1–11, IEEE, Rome, Italy, May 2009.

[30] B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, and Y. Li, "A novel approach to improving the efficiency of storing and accessing small files on Hadoop: a case study by PowerPoint files," in *Proceedings of the IEEE International Conference on Services Computing (SCC '10)*, pp. 65–72, IEEE, Miami, FL, USA, July 2010.

[31] Y. Zhang, Z. Zhu, H. Cui, X. Dong, and H. Chen, "Small files storing and computing optimization in Hadoop parallel rendering," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 20, Article ID e3847, 2017.

[32] G. MacKey, S. Sehrish, and J. Wang, "Improving metadata management for small files in HDFS," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops (CLUSTER '09)*, pp. 1–4, IEEE, New Orleans, LA, USA, September 2009.

[33] C. Vorapongkitipun and N. Nupairoj, "Improving performance of small-file accessing in Hadoop," in *Proceedings of the 11th IEEE International Joint Conference on Computer Science and Software Engineering (JCSSE '14)*, pp. 200–205, IEEE, Chon Buri, Thailand, May 2014.

[34] N. Mohandas and S. M. Thampi, "Improving Hadoop performance in handling small files," *Communications in Computer and Information Science*, vol. 193, no. 4, pp. 187–194, 2011.

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

[36] 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.

[37] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating MapReduce for multi-core and multiprocessor systems," in *Proceedings of the 13th IEEE International Symposium on High Performance Computer Architecture (HPCA '07)*, pp. 13–24, IEEE, Scottsdale, AZ, USA, February 2007.

[38] Q. He, C. Du, Q. Wang, F. Zhuang, and Z. Shi, "A parallel incremental extreme SVM classifier," *Neurocomputing*, vol. 74, no. 16, pp. 2532–2540, 2011.