Cluster Comput (2015) 18:1493–1501
DOI 10.1007/s10586-015-0477-1

A distributed frequent itemset mining algorithm using Spark for Big Data analytics

Feng Zhang[1,3] · Min Liu[1] · Feng Gui[1] · Weiming Shen[2] · Abdallah Shami[3] · Yunlong Ma[1]

Received: 1 June 2015 / Revised: 7 July 2015 / Accepted: 17 August 2015 / Published online: 28 October 2015

Abstract Frequent itemset mining is an essential step in the process of association rule mining. Conventional approaches for mining frequent itemsets in big data era encounter significant challenges when computing power and memory space are limited. This paper proposes an efficient distributed frequent itemset mining algorithm (DFIMA) which can significantly reduce the amount of candidate itemsets by applying a matrix-based pruning approach. The proposed algorithm has been implemented using Spark to further improve the efficiency of iterative computation. Numeric experiment results using standard benchmark datasets by comparing the proposed algorithm with the existing algorithm, parallel FP-growth, show that DFIMA has better efficiency and scalability. In addition, a case study has been carried out to validate the feasibility of DFIMA.

✉ Yunlong Ma
evanma@tongji.edu.cn

Feng Zhang
zhangfeng0726@gmail.com

Min Liu
lmin@tongji.edu.cn

Feng Gui
guifengleaf@gmail.com

Weiming Shen
wshen@ieee.org

Abdallah Shami
ashami2@uwo.ca

1 School of Electronics and Information Engineering, Tongji University, Shanghai 201804, China

2 The Key Laboratory of Embedded System and Service Computing, Tongji University, Shanghai, China

3 Department of Electrical and Computer Engineering, Western University, London, ON N6A 5B9, Canada

Keywords Distributed data mining algorithm · Frequent itemset mining · Big data · Spark

1 Introduction

The past decade has witnessed the remarkable growth of Internet communication technology especially mobile Internet and sensor networks to perceive and obtain information. Organizations from industry, government, and academia possess and store large quantities of data which contain tremendous value. The potential value of big data [1] cannot be unearthed by simple collection or statistical analysis, currently referring to big data. Advanced big data analytics and applications require special technologies to efficiently cope with massive amounts of data. Data mining techniques [2] are now drawing attention from the practitioners of all data-related industries for this purpose. The aim of data mining is to explore data in search and interpretation of unforeseen trends or patterns between variables, and then to verify the results with the detected patterns applied to new subsets.

Since data gathered from a variety of data sources are often a series of isolated data, correlation analysis naturally becomes an important foundation for data mining and big data science [3]. Association rule mining [4,5] was proposed to discover certain interesting correlation relationships among the itemsets of the data. Furthermore, frequent itemset mining [6] is an essential step in the process of association rule mining. Some well-known conventional algorithms, including Apriori [7], FP-growth [8], and other matrix-based algorithms [9] for frequent itemset mining working on single computers, have shown good performance in dealing with small amount of data. Nevertheless, conventional approaches come across significant challenges when computing power and memory space are limited in big data era. Some practices and attempts have been made to mine frequent itemsets from massive data by using parallel computing technologies [10–12].

Parallel programming frameworks can be mainly classified into two categories: memory sharing and distributed architectures (share nothing). Although it's easier to make algorithms implemented parallelism on memory sharing framework, the scalability of them is not satisfactory enough [13]. Message passing interface (MPI) [14,15], which is a common framework for scientific distributed computing, takes the advantage of memory locality. Some researches thus apply MPI to mine frequent itemset [16,17]. In spite of certain advantages in iterative computation, the disadvantages of MPI are its high communication load due to data exchanges between different computer nodes and the lacking of fault tolerance. MapReduce [18], a framework embedded in Apache Hadoop to process large amounts of distributed data in parallel, was designed to support distributed computing in a cloud computing paradigm, turning out to be an efficient platform for parallel data mining of large scale datasets [19]. However, the MapReduce framework is not appropriate for iterative computation, because repeated read-write operations to Hadoop distributed file system (HDFS) would lead to high I/O load and time cost.

To overcome the above problem, the Spark platform [20], a memory-based distributed framework, has been used as solution architecture in this paper. We propose a distributed Apriori-like algorithm, called DFIMA, which has significantly improved performance for the frequent itemset mining algorithm. Apriori algorithm [21] is an iterative process, including candidate itemsets generation and frequent itemsets generation. DFIMA reduces the amount of candidate itemsets by using a matrix-based pruning approach. Moreover, to further improve the efficiency of iterative computation, Spark is applied to adapt the algorithm to be distributed. This paper presents extensive experiments comparing the proposed method with the existing algorithm, parallel FP-growth (PFP) [22] on the Spark 1.3 platform. The results show that DFIMA is better in terms of speed and scalability. Besides, a case study is conducted to validate the feasibility of the proposed algorithm, which is also compared with PFP. In addition to accompanying cars recognition described in this study, there are several other potential applications of the proposed approach such as mining frequent events in online social networks [23,24], understanding user pattern of E-Business [25] and identifying correlated sensors in Wireless Ad Hoc Sensor Networks [26]. The main contributions of this paper can be summarized as follows: One of the important contributions is that a matrix-based pruning technique is adopted into DFIMA, thus to greatly reduce the amount of candidate itemsets and the frequency of database scans. Furthermore, the proposed algorithm has been implemented over Apache Spark 1.3, a fast and general engine for large-scale data processing, which could provide a solution for big data analytics.

The rest of the paper is organized as follows: Sect. 2 describes the review of researches in parallel and distributed frequent itemset mining. Section 3 gives a brief introduction of frequent itemset mining and Apache Spark. Section 4 illustrates the proposed algorithm in details. Section 5 shows the experimental results. Finally, Sect. 6 provides some conclusions and discusses the future work.

2 Literature review

A number of research efforts have explored to address the problem of frequent itemset mining in parallel and distributed environments. Solutions in the literature generally aimed at improving PFP and adapting Apriori to be distributed.

PFP is a parallel form of the classical FP-Growth [27]. Li et al. [28] introduced PFP for query recommendation, which splits a large-scale mining task into independent and parallel tasks. A load balancing FP-Tree algorithm was developed by Yu et al. [29], the itemset for mining is divided by evaluating the tree's width and depth, and a calculate function for loading degree is given. In the work of [30], a single prefix path compression technique was developed to identify frequent closed itemsets, and a partition-based projection mechanism is established to make the mining efficient and scalable for large databases. Chen et al. [31] proposed a parallel FP-growth algorithm running on computer cluster; a projection method was used to find all the conditional pattern bases of frequent items so as to avoid memory overflow. In summary, the main principle of PFP is to group the items and then distribute the conditional databases to the mappers, which is not efficient in memory or speed.

Research efforts have already been made to improve Apriori-like algorithms or Apriori-based algorithms and convert them into distributed versions, mostly under the MapReduce environment. Lin et al. [12] developed three versions of Apriori algorithm (SPC, FPC, and DPC) on MapReduce framework, SPC is a straight-forward algorithm while FPC aims at reducing the number of scheduling invocations, and DPC features in dynamically combining candidates of various lengths. Farzanyar et al. [32] presented a pruning technique to decrease the number of partial frequent itemsets in Map phase based on MapReduce model. The MrAdam algorithm [33] was proposed to obtain the approximate collections of frequent itemsets, which combines a Chernoff bound-based approach and the MapReduce framework. Moens et al. [34] introduced two approaches for frequent itemset mining, Dist-Eclat focuses on speed whereas BigFIM is optimized to run on really large datasets. By storing metadata in the form of Transaction Identifiers, Yu et al. [35] proposed a distributed parallel Apriori algorithm (DPA).

Ozkural et al. [36] proposed a divide-and-conquer strategy to parallelize the FIM (frequent itemset mining) task, they used a top-down data partitioning scheme with selective replication. Aouad et al. [37] studied the performance of distributed Apriori-like frequent itemset mining, and found that the intermediate communication steps and remote support counts computation impose restrictions on global performance of classic distributed schemes. In order to avoid generating lots of candidate sets and scanning the transaction database repeatedly, Chen et al. [38] proposed BE-Apriori algorithm based on the classic Apriori algorithm, and applied pruned optimization strategy to reduce the generation of frequent itemsets, while used transaction reduction strategy to reduce the scale of transaction database. In general, the performance of these approaches might not be satisfactory due to the bottleneck of iterative computation when confronting with large scale datasets. Therefore, in this paper, a distributed algorithm for frequent itemset mining (DFIMA) is proposed to improve and speed-up the process of frequent itemset mining.

3 Preliminaries

3.1 Frequent itemset mining

Suppose that $I = \{I_1, I_2, \ldots, I_m\}$ is an itemset composed of m items. A database D consists of a series of transactions. Each transaction is a subset of I and has a unique label denoted by *TID*. A set of items is referred to as an itemset. An itemset that contains k items is a *k-itemset*. For instance, the set {beer, diaper} is a 2-*itemset*. The occurrence frequency of an itemset is the number of transactions that contain the itemset. Given an itemset X, the support number of X is the number of transactions in D that contain$X$. If the support number of X is greater than or equal to the specified minimum support threshold (abbreviated as *MinSup*), then the itemset X is labeled as a frequent itemset. The purpose of frequent itemset mining is to find all frequent itemset in a given database.

3.2 Apache Spark

Conceptually, Apache Spark is an open-source in-memory data analytics cluster computing framework, developed in the AMPLab at UC Berkeley. As a MapReduce-like cluster computing engine, Spark also possesses good characteristics such as scalability, fault tolerance as MapReduce does. The main abstraction of Spark is resilient distributed datasets (RDDs) [39], which make Spark be well qualified to process iterative jobs, including PageRank algorithm [40], K-means algorithm and etc. RDDs are unique to Spark and thus differentiate Spark from conventional MapReduce engines.

In addition, on the basis of RDDs, applications on Spark can keep data in memory across queries and reconstruct automatically data lost during failures [41]. RDD is a read-only data collection, which can be either a file stored in an external storage system, such as HDFS, or a derived dataset generated by other RDDs. RDDs store much information, such as its partitions, and a set of dependencies on parent RDDs called *lineage*. With the help of the *lineage*, Spark recovers the lost data quickly and effectively. Spark shows great performance in processing iterative computation because it can reuse intermediate results, keep data in memory across multiple parallel operations.

4 Distributed frequent itemset mining algorithm (DFIMA)

This section will introduce a distributed frequent itemset mining algorithm, which is an Apriori-like algorithm. DFIMA is a breadth first search algorithm by use of a property called apriori (see Property 1) which is referred as a support-based technique firstly introduced by Agrawal, et al. [4]. In general, Apriori algorithm can be viewed as a two-step process: in the first step, scan the database, then count the number of each item in the database, i.e. calculating the support number of each item, and pick out all items whose support number are not less than *MinSup* to form frequent 1-*itemset*. In the second step, keep generating the entire candidate $(k+1)$-*itemset* based on the whole frequent k-*itemset*, and scanning the database repeatedly till there is no candidate $(k+1)$-*itemset*.

Property 1 If X is a frequent itemset, then any subset of X is a frequent itemset. In other words, if X is a frequent $(k+1)$-*itemset*, the number of frequent k-*itemset* is greater than $k+1$.

4.1 Matrix-based pruning algorithm

In this subsection, the main principles of matrix-based pruning algorithm for frequent itemset mining are illustrated. The key to this algorithm is to acquire the Boolean vector for each item of a given database and calculate the 2-*itemset* matrix, by which the amount of candidate itemset is reduced.

Assume that database D includes n transactions $T = \{T_1, T_2, \ldots, T_n\}$, m different items $I = \{I_1, I_2, \ldots, I_m\}$ and the value of *MinSup* is set to $Min\_\sup$. The main steps are listed as follows:

Step 1: Obtain the Boolean vectors for each item of frequent 1-*itemset*.

Scan the database D to obtain n dimension Boolean vector, denoted as$V_i = (b_1, b_2, \ldots, b_j \ldots, b_n)^T$, where $b_j = \begin{cases} 1, & I_i \in T_j \\ 0, & I_i \notin T_j \end{cases}$, $(j = 1, 2, \ldots n)$ for each item $I = \{I_1, I_2, \ldots, I_m\}$.

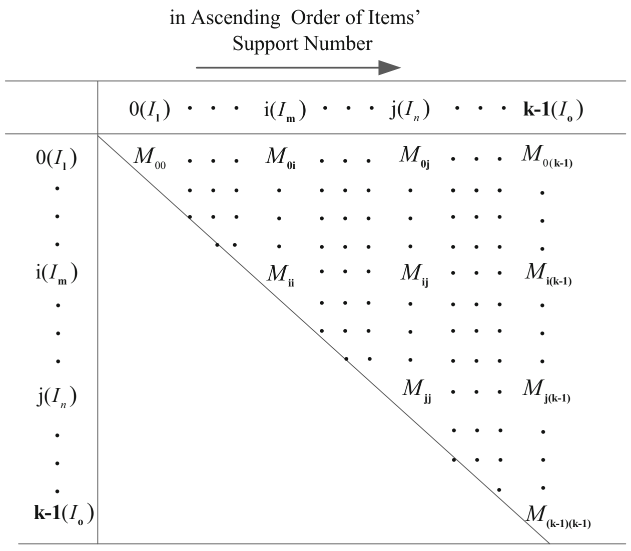

Fig. 1 The process to generate the 2-*itemset* matrix M

Thus the support number of I_i equals the number of nonzero elements in V_i. Then record the items and corresponding Boolean vectors whose support number are not less than $Min\_\sup$. After the whole frequent 1-*itemset* are obtained, sort them in ascending order of support number.

Step 2: Calculate the 2-*itemset* matrix according to the Boolean vectors of the entire frequent 1-*itemset*.

Figure 1 shows the process to generate the 2-*itemset* matrix M, of which each element is respectively produced by items of the entire frequent 1-*itemset* that has been sorted in ascending order. Specifically, if the number of the entire frequent 1-*itemset* is k, then M is a $k \times k$ matrix. Let the Boolean vector V_i and V_j be the ith and jth item of the sorted frequent 1-*itemset* respectively. The number of nonzero elements of the vector that generated by a logical *AND* operation between V_i and V_j is recorded and defined as the value of $M_{ij}(i \neq j)$, an element of matrix M. In particular, the value of M_{jj}, i.e., the diagonal elements of M, is equal to the frequency of $M_{ij} \geq Min\_\sup \quad (i = 0, 1, \ldots, j-1)$ for column j of M. We only need to calculate the value of $M_{ij}(i \leq j)$ since matrix M is symmetric. Finally, the whole frequent 2-*itemset* are acquired on the basis of the matrix M.

Step 3: Obtain frequent $(k+1)$-*itemset* by use of frequent k-*itemset* $(k \geq 2)$.

Let L_k be the whole frequent k-*itemset*. The primary thing we should consider is the number of frequent k-*itemset*. If the number is greater than $k+1$, then the frequent $(k+1)$-*itemset* may exist according to Property 1. For each frequent k-*itemset* denoted by F, which has been sorted in ascending order of support number, let the first and last item of F be written as I_{first} and I_{last} respectively. Apparently, I_{first} has the minimum support number while I_{last} has the maximum support number. Assume that the row number in M which corresponds with I_{first} is $u(u = 0, 1, \ldots, k-1)$ and the row number in M which corresponds with I_{last} is v $(v = 0, 1, \ldots, k-1)$. Then search each column of the row v in M in turn. For each column $w(w < v)$ in M, if meet the conditions that $M_{vw} \geq Min\_\sup$, $M_{vw} \geq k$ and $M_{uw} \geq Min\_\sup$, then let the corresponding item of the column w be I_{new}, and accordingly a candidate $(k+1)$-*itemset*is obtained by combining the frequent k-*itemset*, i.e., F, with the item I_{new}. In the light of Definition 1, the support number of the candidate $(k+1)$-*itemset* can be obtained. If the support number is greater than $Min\_\sup$, then the candidate $(k+1)$-*itemset* is frequent.

Definition 1 Suppose that V_r is the Boolean vector of item I_r, then the support number of the itemset $Q = \{I_o, I_p, \ldots, I_q\}$ is equal to the number of nonzero elements in the vector generated by the logical *AND* operation between all the Boolean vectors of the items in itemset Q.

4.2 Implementation over Spark

In this subsection, the implementation of the above algorithm for single machine based on Spark is stated in detail. In order to reduce the memory usage, HashMap is used in our program to save the Boolean vector for each frequent 1-*itemset* and all the frequent 2-*itemset*, instead of saving the 2-*itemset* matrix M.

Step 1: Obtain the Boolean vectors for all frequent 1-*itemset*, then produce all the frequent 2-*itemset* based on these Boolean vectors.

As illustrated in Fig. 2, read the input dataset from HDFS to create the HadoopRDD first. Next, the HadoopRDD will call the *flatMap* function to produce a FlatMappedRDD which contains all items of the input data. Apply the *map* function to the FlatMappedRDD so as to transform each item to a *Tuple*-(*item*, 1), thus a MappedRDD is obtained. Besides, count the support number of each item of MappedRDD by use of the *reduceByKey* function, further to pick out the items whose support number are not less than $Min\_\sup$. Then, sort all the frequent 1-*itemset* in ascending order of their support number, and save them in an array arr. For each frequent 1-*itemset* in arr, get the relevant Boolean vector, and save them as a <item, vector> *key/value* pair in a HashMap. Finally, the entire frequent matrix can be obtained by using the array arr and these Boolean vectors. The matrix will be saved into a HashMap, whose *key* is a frequent 1-*itemset*. For an instance, let G be a *key*, then the value is a HashMap, whose *key* is an another frequent 1-*itemset* that can construct a frequent 2-*itemset* together with G, and the value of such a HashMap is the support number of current 2-*itemset*.

Step 2: Get all frequent $(k+1)$-*itemset* by using frequent k-*itemset*$(k \geq 2)$.

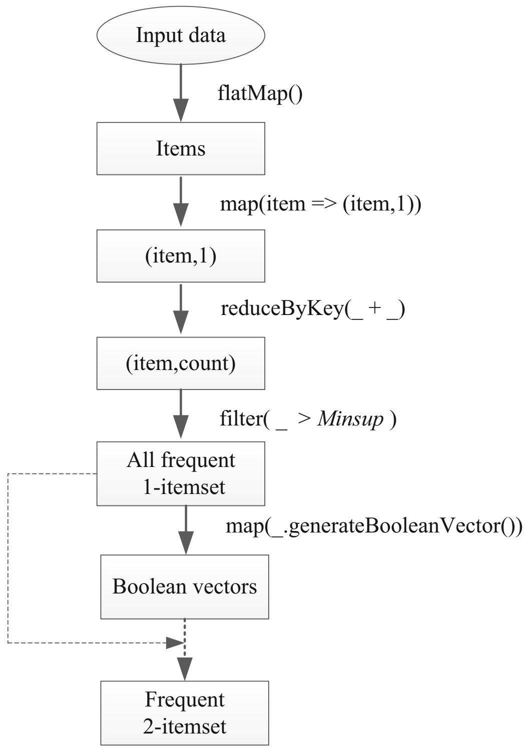

Fig. 2 The process of step 1 on Spark

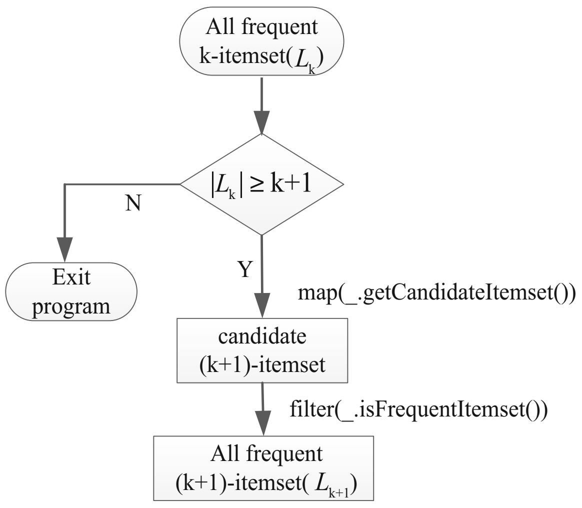
All frequent k-itemset(L_k)
$|L_k| \geq k+1$
N
Y
Exit program
map(_.getCandidateItemset())
candidate (k+1)-itemset
filter(_.isFrequentItemset())
All frequent (k+1)-itemset(L_{k+1})

Fig. 3 The process of step 2 on Spark

As shown in Fig. 3, the procedure of this step is similar to that of Step 3 stated in Sect. 4.1. To begin with, the RDDs that contain frequent k-*itemset* will call the *map* function to produce all candidate $(k + 1)$-*itemset*. Secondly, use the *broadcast* variable of Spark to save the Boolean vectors of all frequent 1-*itemset*. Moreover, we employ the Definition 1 to determine if a candidate $(k + 1)$-*itemset* is frequent. Consequently, all the frequent $(k + 1)$-*itemset* are obtained.

It's worth noting that some RDDs are used repeatedly throughout the job, for example, the HadoopRDD saved with the input data is reused to get the Boolean vectors of all frequent 1-*itemset*, and the RDDs saved with the frequent k-*itemset* are reused to obtain all frequent $(k + 1)$-*itemset*. Therefore, the *cache* function is applied to store these RDDs in distributed workers of the cluster, and further to achieve in-memory computation. In this way, the input data only needs to be loaded once, which improves the performance of entire job.

In addition, some variables, stored in the memory of Spark cluster, should be shared when dealing with multiple parallel operations. For example, the variables that save the Boolean vectors of the frequent 1-*itemset*. In view of efficiency, we apply the *broadcast* function, it allows programmers to send shared variables to all workers of the cluster only once instead of delivering a copy of shared variables for every task.

5 Experimental results

In this section, comparisons of DFIMA and PFP are made to evaluate their performance of speed and scalability. Both the two algorithms have been implemented over the Spark version 1.3. After the performance evaluation, a case study is given to discuss the practicality of the two algorithms.

5.1 Performance evaluation

In this subsection, the datasets T10I4D100K and T40l10D 100K are used for experiments. These two real datasets were presented at the first *IEEE ICDM workshop on Frequent Itemset Mining* (FIMI' 03) [42]. Table 1 shows the detail of the two datasets.

The experiments were conducted on a cluster that consists of 3 computer nodes with Centos 7.0 installed. Every computer node was deployed with the same physical environment, i.e., Intel Core i5-4440M CPU 3.1 GHz, 16 GB memory and 1TB disk.

The aim of first comparison is to estimate the speed performance by analyzing the running time of DFIMA and PFP (Spark 1.3). In this case, we assume that the support degree varies while the number of computer nodes remains to be 3. Table 2 shows the specific running time (in seconds) of DFIMA and PFP for datasets T10l4D100K and

Table 1 The properties of datasets used in experiment

Dataset	Size	Transactions	Items	Average length
T10I4D100K	3.84 MB	100,000	870	10
T40l10D100K	14.8 MB	100,000	1000	40

Table 2 The running time comparison on T10I4D100K and T40I10D100K

Dataset	Support degree	PFP (Spark 1.3)	DFIMA
T10I4D100K	0.01	64s	25s
	0.03	35s	17s
	0.05	37s	14s
T40I10D100K	0.01	321s	180s
	0.03	316s	81s
	0.05	250s	67s

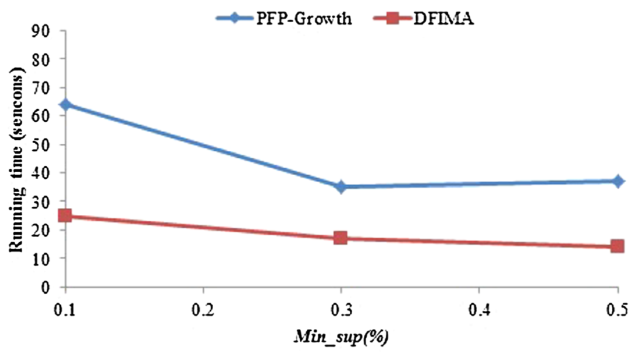

Fig. 4 The running time on T10I4D100K

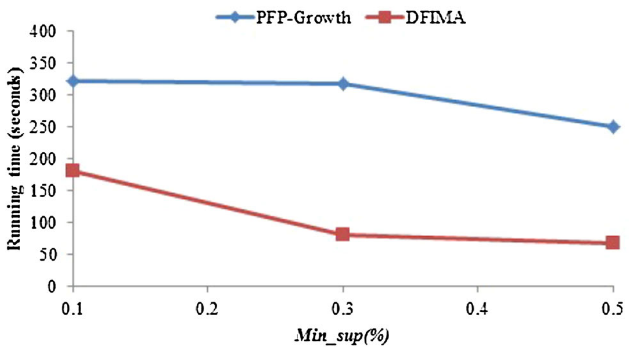

Fig. 5 The running time on T40I10D100K

Table 3 The running time comparison on “T40I10D100K” databases

Dataset	Support degree	PFP (Spark 1.3)	DFIMA
T40I10D100K*5 (73.8 MB)	0.5	46s	11s
	0.3	110s	15s
	0.1	262s	44s
	0.05	534s	126s
T40I10D100K*10 (147.6MB)	0.5	51s	16s
	0.3	58s	15s
	0.1	301s	105s
	0.05	749s	281s
T40I10D100K*20 (295.2 MB)	0.5	54s	24s
	0.3	58s	25s
	0.1	620s	284s
	0.05	1498s	598s
T40I10D100K*40 (590.4 MB)	0.5	72s	28s
	0.3	83s	26s
	0.1	1372s	578s
	0.05	2671s	611s
T40I10D100K*80 (1.2 GB)	0.5	103s	54s
	0.3	124s	51s
	0.1	2898s	365s
	0.05	5892s	583s

T40I10D100K. The support degree is set to 0.01, 0.03 and 0.05 respectively. We can see from Table 2 that the two algorithms tend to be more efficient when support degree is set to a higher level. Note that the running time of DFIMA is apparently less than that of PFP, for both datasets. For example, for dataset T40I10D100K, when support degree is 0.05, the running time of PFP is 250s, nearly 3 times bigger than the time of DFIMA, i.e., 67s.

The running time with different support degree for dataset T10I4D100K and T40I10D100K is shown in Figs. 4 and 5 separately. The x-axis denotes the support degree and y-axis represents the running time. As the support degree grows from 0.1 to 0.5 %, we notice in Fig. 4 that the value of running time for PFP is consistently higher than the same value for DFIMA. Figure 5 shows that the superiority of DFIMA in terms of the remaining time metric is clearer when using bigger data sets. It can be seen that the difference between the two algorithms is bigger compared to the one noticed in Fig. 4.

Besides, another comparison of speed performance between DFIMA and PFP (running on Spark 1.3) was conducted on “T40I10D100K” datasets. These “large” versions of the original datasets are manually generated to increase the scale of datasets. It can be seen from Table 3 that DFIMA significantly outperforms PFP for all settings of support degree. When support number is small, the time cost of two algorithms seem to increase more quickly.

The following experiment evaluates the scalability of DFIMA, which is also measured by the running time (Spark 1.3). The dataset T10I4D100K is used here. The experiment is performed on condition that the number of cluster computer nodes ranges from 2 to 8 while the support degree remains to be 0.5 %. In Fig. 6, x-axis denotes the number of computer nodes of Spark cluster and y-axis represents the running time of DFIMA algorithm. Figure 6 illustrates the running time with various numbers of computer nodes. With more computer nodes, DFIMA needs less execution time, and the curve of DFIMA has a nearly linear decline. DFIMA shows a characteristic of near-linear scalability.

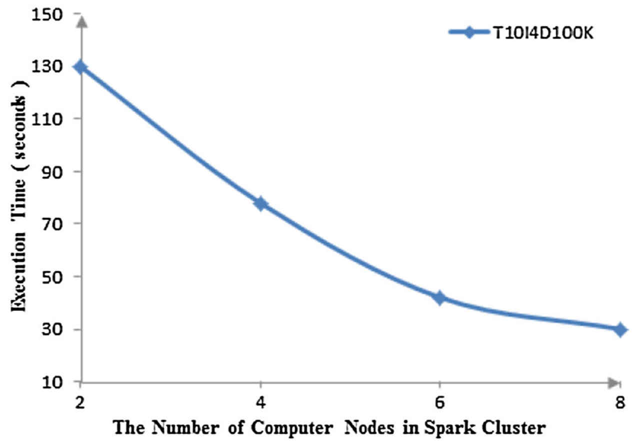

Fig. 6 The running time with different computer nodes

Table 4 The results of PFP-Growth and DFIMA on Car dataset

Dataset	Support number	PFP (Spark 1.3)	DFIMA
Car dataset	100	521s	201s
	150	435s	189s
	200	337s	158s

5.2 Case study

In this subsection, a case study is provided to evaluate the practicality and effectiveness of DFIMA, which is also compared with PFP-Growth (Spark 1.3).

An automatic license plate recognition database was generated to obtain a dataset called Car dataset, so as to simulate the application of the accompanying cars recognition, which is used to identify suspect criminal gang. It is common that cars move in a queue in dynamic traffic flows. But in some cases, when the condition of accompanying occurs with a high frequency, these cars are regarded as accompanying cars and suspected of gang crime.

We simulate a database that contains more than 40 intelligent monitoring and recording systems of vehicles on highways. Information collected in the database includes the license plate, passing time, speed and etc. of each passed vehicle. For each intelligent monitoring and recording system, data is gathered and saved as one record every five minutes. Based on the data recording of nearly 6 months, we get a Car dataset (2.1 GB) that consists of 29204016 records. Considering that cars' moving in queue is a normal phenomenon in rush hours, this simulation measures accompanying cars by defining a certain "accompanying times", namely the support number.

Table 4 shows the execution time of the two algorithms, the support number is set to 100, 150 and 200 separately. In fact, we observe that both DFIMA and PFP perform well on Spark in this simulation. However, the time cost of PFP is more than twice the value of DFIMA in all cases. Therefore, the case study also testifies that the running time of the proposed approach is significantly less than that of PFP (running on Spark 1.3).

The above experimental results indicate that DFIMA is able to efficiently reduce the amount of candidate itemsets and is capable of using as a reliable approach for frequent itemset mining with a considerable performance.

6 Conclusion and future work

The performance bottleneck due to repeated database scanning of Apriori algorithm and iterative computation on MapReduce framework hinders frequent itemset mining from massive data. This paper presents a novel distributed algorithm, called DFIMA, in which a matrix-based pruning approach is introduced and used as a means of reducing the amount of candidate itemset. Moreover, to further promote the efficiency of iterative computation, we have integrated our approach into the Spark platform. The experimental results indicated that the proposed algorithm shows better efficiency and scalability compared to PFP (running on Spark 1.3). We also notice in experiments that our method performs especially well in occasions with a relatively high support degree. Further optimization will be considered to improve DFIMA and make it suitable for more mining cases.

Acknowledgments The research work presented in this paper is partially supported by the Scientific Research Projects of the NSFC (Grant No. 61173015) and the Fundamental Research Funds for the Central Universities.

References

1. Sandhu, R., Sood, S.K.: Scheduling of big data applications on distributed cloud based on QoS parameters. Clust. Comput. **18**, 1–12 (2014). doi:10.1007/s10586-014-0416-6
2. Han, L., Ong, H.Y.: Parallel data intensive applications using MapReduce: a data mining case study in biomedical sciences. Clust. Comput. **18**(1), 403–418 (2015). doi:10.1007/s10586-014-0405-9
3. Chen, Y., Li, F., Fan, J.: Mining association rules in big data with NGEP. Clust. Comput. **18**, 1–9 (2015). doi:10.1007/s10586-014-0419-3
4. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, vol. 1215, pp. 487–499 (1994)
5. Agrawal, R., Shafer, J.C.: Parallel mining of association rules. IEEE Trans. Knowl. Data Eng. **8**(6), 962–969 (1996). doi:10.1109/69.553164
6. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using fp-trees. IEEE Trans. Knowl. Data Eng. **17**(10), 1347–1362 (2005). doi:10.1109/TKDE.2005.166
7. Mohamed, M.H., Darwieesh, M.M.: Efficient mining frequent itemsets algorithms. Int. J. Mach. Learn. Cybern. **5**(6), 823–833 (2014). doi:10.1007/s13042-013-0172-6

8. Totad, S.G., Geeta, R.B., Reddy, P.P.: Batch incremental processing for FP-tree construction using FP-Growth algorithm. Knowl. Inf. Syst. **33**(2), 475–490 (2012). doi:10.1007/s10115-012-0514-9
9. Zhen-yu, L., Wei-xiang, X., Xumin, L.: Efficiently using matrix in mining maximum frequent itemset. In: WKDD'10. Third International Conference on Knowledge Discovery and Data Mining, 2010, pp. 50–54. IEEE (2010)
10. Ye, Y., Chiang, C. C.: A parallel apriori algorithm for frequent itemsets mining. In: Fourth International Conference on, Software Engineering Research, Management and Applications, 2006, pp. 87–94. IEEE (2006)
11. Upadhyaya, S.: Parallel approaches to machine learning—a comprehensive survey. J. Parallel Distrib. Comput. **73**(3), 284–292 (2013). doi:10.1016/j.jpdc.2012.11.001
12. Lin, M.Y., Lee, P.Y., Hsueh, S.C.: Apriori-based frequent itemset mining algorithms on MapReduce. In: Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ACM 76 (2012). doi:10.1145/2184751.2184842
13. Moens, S., Aksehirli, E., Goethals, B.: Frequent itemset mining for big data. In: IEEE International Conference on Big Data, pp. 111–118, IEEE (2013)
14. Pacheco, P.S.: Parallel Programming with MPI. Morgan Kaufmann Publishers Inc, San Francisco (1997)
15. Li, S., Hoefler, T., Hu, C., et al.: Improved MPI collectives for MPI processes in shared address spaces. Clust. Comput. **17**(4), 1139–1155 (2014). doi:10.1007/s10586-014-0361-4
16. Otey, M.E., Wang, C., Parthasarathy, S., et al.: Mining frequent itemsets in distributed and dynamic databases. In: Third IEEE International Conference on Data Mining, ICDM 2003, pp. 617–620. IEEE (2003)
17. Kaosar, M.G., Xu, Z., Yi, X.: Distributed Association rule mining with minimum communication overhead. In: Proceedings of the Eighth Australasian Data Mining Conference, vol. 101, pp. 17–23. Australian Computer Society Inc (2009)
18. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008). doi:10.1145/1327452.1327492
19. Zaharia, M., Chowdhury, M., Franklin, M.J., et al.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, pp. 10–10 (2010)
20. Jiang, H., Chen, Y., Qiao, Z., et al.: Scaling up MapReduce-based big data processing on Multi-GPU systems. Clust. Comput. **18**(1), 369–383 (2015). doi:10.1007/s10586-014-0400-1
21. Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. Lect. Notes Comput. Sci. **1910**, 13–23 (2000)
22. Pramudiono, I., Kitsuregawa, M.: Parallel FP-growth on PC cluster. In: Advances in Knowledge Discovery and Data Mining, pp. 467–473. Springer, Berlin (2003)
23. Gu, H., Hang, H., Lv, Q., et al.: Fusing text and frienships for location inference in online social networks. In: 2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT), vol. 1, pp. 158–165. IEEE (2012)
24. Gu, H., Xie, X., Lv, Q., et al.: Etree: effective and efficient event modeling for real-time online social media networks. In: 2011 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), vol. 1, pp. 300–307. IEEE (2011)
25. Priyadarsini, S., Viswanathan, R.: Web usage mining for better understanding of user pattern to improve productivity of E-business. Int. J. Appl. Eng. Res. **9**(11), 1753–1763 (2014)
26. Boukerche, A., Samarah, S.: A novel algorithm for mining association rules in wireless ad hoc sensor networks. IEEE Trans. Parallel Distrib. Syst. **19**(7), 865–877 (2008)
27. Zhou, L., Wang, X.: Research of the FP-growth algorithm based on cloud environments. J. Softw. **9**(3), 676–683 (2014). doi:10.4304/jsw.9.3.676-683
28. Li, H., Wang, Y., Zhang, D., et al.: Pfp: parallel fp-growth for query recommendation. In: Proceedings of the 2008 ACM Conference on Recommender Systems, ACM 107–114 (2008). doi:10.1145/1454008.1454027
29. Yu, K.M., Zhou, J., Hsiao, W.C.: Load balancing approach parallel algorithm for frequent pattern mining. In: Parallel Computing Technologies, pp. 623–631. Springer, Berlin (2007)
30. Pei, J., Han, J., Mao, R.: CLOSET: an efficient algorithm for mining frequent closed itemsets. In: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, vol. 4(2), pp. 21–30 (2000)
31. Chen, M., Gao, X., Li, H.: An efficient parallel FP-Growth algorithm. In: CyberC'09. International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009, pp. 283–286. IEEE (2009)
32. Farzanyar, Z., Cercone, N.: Efficient mining of frequent itemsets in social network data based on MapReduce framework. In: Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ACM 1183–1188 (2013). doi:10.1145/2492517.2500301
33. Fumarola, F., Malerba, D.: A parallel algorithm for approximate frequent itemset mining using MapReduce. In: 2014 International Conference on High Performance Computing & Simulation (HPCS), pp. 335–342. IEEE (2014)
34. Moens, S., Aksehirli, E., Goethals, B.: Frequent itemset mining for big data. In: 2013 IEEE International Conference on Big Data, pp. 111–118. IEEE (2013)
35. Yu, K., Zhou, J., Zhou, J., et al.: A load-balanced distributed parallel mining algorithm. Expert Syst. Appl. **37**(3), 2459–2464 (2010). doi:10.1016/j.eswa.2009.07.074
36. Ozkural, E., Ucar, B., Aykanat, C.: Parallel frequent item set mining with selective item replication. IEEE Trans. Parallel Distrib. Syst. **23**(10), 1632–1640 (2011). doi:10.1109/TPDS.2011.32
37. Aouad, L.M., Le-Khac, N.A., Kechadi, T.M.: Performance study of distributed Apriori-like frequent itemsets mining. Knowl. Inf. Syst. **23**(1), 55–72 (2010). doi:10.1007/s10115-009-0205-3
38. Chen, Z., Cai, S., Song, Q., et al.: An improved Apriori algorithm based on pruning optimization and transaction reduction. In: 2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), pp. 908–1911. IEEE (2011)
39. Zaharia, M., Chowdhury, M., Das, T., et al.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, USENIX Association 2-2 (2012)
40. Haveliwala, T.H.: Topic-sensitive pagerank: a context-sensitive ranking algorithm for web search. IEEE Trans. Knowl. Data Eng. **15**(4), 784–796 (2003). doi:10.1109/TKDE.2003.1208999
41. Xin, R.S., Rosen, J., Zaharia, M., et al.: Shark: SQL and rich analytics at scale. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM 13–24 (2013). doi:10.1145/2463676.2465288
42. Goethals, B., Zaki, M.J.: FIMI'03: Workshop on frequent itemset mining implementations. In: Third IEEE International Conference on Data Mining Workshop on Frequent Itemset Mining Implementations, pp.1–13. IEEE (2003)

Feng Zhang is a Ph.D. candidate in School of Electronics and Information Engineering at Tongji University, Shanghai, China. He received his Bachelor degree (2011) from Anhui Architecture University, Anhui, China. His research interests focus on intelligent algorithms and data mining with big data in domain of WSN.

Min Liu is a full professor in School of Electronics and Information Engineering at Tongji University, Shanghai, China. He received his Bachelor degree (1993) from China University of Geosciences (Wuhan), and his Master (1996) and Ph.D. (1999) degrees from Zhejiang University, China. He worked as a post-doctoral fellow in the computer science and engineering department of Shanghai JiaoTong University from July 1999 to April 2001. As product architecture engineer, he developed an enterprise resources management system in Asia-bridge Software Co. Ltd., from May 2001 to Oct. 2004. He has been working on system engineering and service computing to collaborative MRO and intelligent manufacturing for about 12 years. He has published over 60 papers in scientific journals and international conferences in the related areas.

Feng Gui is a master student in School of Electronics and Information Engineering at Tongji University, Shanghai, China. He received his Bachelor degree (2012) from Anhui University, Anhui, China.

Weiming Shen is a Senior Research Scientist at the National Research Council Canada and an Adjunct Professor in software engineering at the University of Western Ontario. He received his Bachelor (1983) and Master's (1986) degrees from Northern (Beijing) Jiaotong University, China and his PhD degree (1996) from the University of Technology of Compiegne, France. He has been working on intelligent software agents and applications to collaborative engineering design and intelligent manufacturing for about 19 years. Dr. Shen is a Co-Chair of the Technical Committee on Computer Supported Cooperative Work in Design (CSCWD) with the IEEE SMC Society.

Abdallah Shami is a full professor in Department of Electrical and Computer Engineering, Western University, London, Ontario, Canada. He received his Bachelor degree (1997) from Lebanese University, and his Master (2001) and Ph.D. (2003) degrees from City University of New York. His research interests focus on Network Virtualization, Network Function Virtualization (NFV), Cloud Computing, Agile Computing, Energy Limited Communication, etc.

Yunlong Ma is a lecture in School of Electronics and Information Engineering at Tongji University, Shanghai, China. He received his Bachelor (1999), Master (2002) and Ph.D. (2005) degrees from Tongji University, China.