Hindawi
Scientific Programming
Volume 2022, Article ID 3362203, 11 pages
https://doi.org/10.1155/2022/3362203

Research Article

Design and Implementation of Automatic Selection of the Most Efficient Itemset Algorithm Based on Spark

Huang Jie [1,2] and Wu Dehua [2]

[1] *Hunan Provincial Engineering Research Center for Aircraft Maintenance, Changsha 410024, Hunan, China*
[2] *Department of Aviation Electronic Equipment Maintenance, Changsha Aeronautical Vocational and Technical College, Changsha 410024, Hunan, China*

Correspondence should be addressed to Huang Jie; huangjie918@163.com

Received 23 February 2022; Revised 14 March 2022; Accepted 26 April 2022; Published 24 May 2022

Academic Editor: Punit Gupta

The combination of Spark distributed platform and High-Utility Itemset Mining can solve the problem of long running time issue of High-Utility Itemset Mining. In the experiment, we conclude that Spark-based parallel D2HUP and EFIM algorithms have a greater improvement in running time efficiency than serial algorithms. The existing research has shown that the EFIM and D2HUP algorithms are the two best algorithms for High-Utility Itemset Mining. This paper generates 118 datasets by generating and collecting the running time of the two algorithms in the real and simulated datasets, taking into account the characteristics of each dataset's length, sparse degree, and dataset size as characteristics with running time as the prediction target and then establishing a model. The accuracy of the prediction was evaluated through experiments, and a set of rules based on decision trees was generated. According to the rules, the fastest algorithm between EFIM and D2HUP can be predicted very well.

1. Introduction

Data mining (DM) is a term derived from Knowledge-Discovery in Databases (KDD) [1], which is an important step in KDD. Born at the intersection of computer and statistics, it aims to show the valuable information and concepts hidden in data to users in a more intuitive way through statistical sampling, estimation test, pattern recognition, machine learning, distributed technology, visualization, and other methods and means [2].

The essence of data mining is a kind of deeper mathematical analysis. Its main feature is to model the structured data after cleaning and transforming the huge data in the database and analyze the results to extract the key data, which can help decision-makers make business decisions.

The deep essence of data mining is a mathematical analysis, whose feature is that after the cleaning and transformation of the huge data in the database [3], the obtained structured data will be modeled, the obtained results will be analyzed, and the key data will be extracted to help decision-makers make business decisions [4].

Data analysis methods commonly used in data mining include classification, regression, clustering, and association rule analysis [5], which mine data from a different perspective. The concept of association rule was first proposed by Agrawal et al. in 1993, which is used to find meaningful connections hidden in large datasets. The discovered connections can be expressed in the form of association rule or frequent itemset. In the traditional frequent itemset mining [6], the weight of each itemset is the same, and more attention is paid to the degree of association between items and whether itemsets frequently occur, which will result in the loss of itemsets with low support but high utility value [7].

In efficient itemset mining, each item has the utility of quantity and profit, and the combination of efficient itemsets that meet the user-set threshold is not necessarily frequent [8], so the antimonotone feature in frequent itemset mining is no longer applicable [9]. Mining efficient itemsets is a challenging task. With the increase in data volume, the mining time of an efficient itemset increases exponentially [10]. At present, there are few efficient itemset mining algorithms based on the Spark distributed general framework. The combination of Spark and efficient itemset mining is expected to better solve the problem of time-consuming in the process of efficient itemset mining [11].

The two-phase algorithm proposed in literature [12] is the most classic efficient itemset mining algorithm. The algorithm is divided into two stages. In the first stage, TWU (Transaction Weight Utility) is used to narrow the search space and generate candidate itemsets. In the second stage, the efficient itemset meeting the conditions is selected as the final result. However, the upper bound of the algorithm is not compact enough, and a large number of candidate itemsets will be generated in the first stage. At the same time, in the second stage of screening the final results, the database will be scanned repeatedly, resulting in low efficiency of the algorithm. The D2HUP algorithm [6] retains original transaction information by introducing a more compact upper bound and a linked list structure named CAUL and mining efficient itemsets by recursive search. In addition, EFIM algorithm is one of the best efficient itemset mining algorithms at the present stage, which compresses the search space by merging transaction databases and further improves the efficiency of pruning through a more compact upper bound. In addition, many new algorithms are constantly proposed. So far, the best performing algorithms are EFIM and D2HUP.

2. Literature Review of Spark

2.1. Spark System. Spark is a general-purpose parallel computing framework designed for processing large amounts of data developed by the UC Berkeley AMP Lab in 2009 and made open source in 2010 [13]. Spark's biggest feature is that it is based on a memory-based parallel computing framework, similar to Hadoop's MapReduce [14], but with advantages that Hadoop does not have. Spark can store the intermediate results of running in memory, reducing disk I/O and thus accelerating computing efficiency. So Spark performs well in data mining and machine learning algorithms that require iteration. In addition, Spark supports Scala, Java, Python, and other language interfaces. Developers can choose a language they are familiar with to implement. Spark also has a wealth of APIs for developers to call, which makes it extremely easy to use [15].

Spark is an ecosystem that provides solutions such as flow operations, iterative operations, and graph operations, as shown in Figure 1.

Spark SQL. It uses SQL expressions to do data query on Spark to analyze big data query [16] and can also call user-defined functions to combine query results with analysis results to reuse data, thus improving query efficiency.

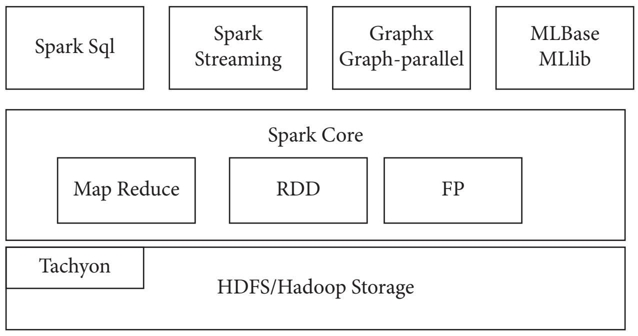

FIGURE 1: Spark ecosystem diagram.

Sparking Streaming. It is a stream processing system that can perform complex operations such as Map, Reduce, and Join on real-time data and save the results to external file systems.

Graph X. It is an API provided by the Spark platform for graph operation. It provides the concept and operation of an elastic distributed attribute graph and uses its framework to realize various operations based on the graph.

MLBase/MLlib. It is a component that can provide related machine algorithms and utilities, including classification algorithms, regression algorithms, clustering algorithms, reduction and low-level optimization programs, as well as testing and data generation. Four boundaries are defined, that is, MLOptimizer, MLI, Machine Learning library (MLlib), and parallel computing framework [17].

Tachyon. This component is a distributed system that can provide high efficiency, high fault tolerance, and high reliability calculation [18].

Spark. It is the core computing framework of the ecosystem, which can be run independently or on a cluster [19]. Cluster operation requires the management system to distribute tasks to each node.

In short, the Spark system is based on RDD, and the Spark framework is a memory-based parallel computing big data platform, which can meet the computing needs of different systems with different products.

2.2. Spark Distributed Cluster Setup. Spark can run computations locally in a standalone mode or in parallel on a distributed cluster.

The mode of Spark operation is determined by the Master of the environment variable Spark-Context and the deployment mode of the cluster. The common modes of Spark are as follows:

(1) *Standalone.* Deploy Spark to Master and all nodes of Worker.
(2) Yarn client: Yarn cluster operates on Executor and Driver running locally.
(3) Yarn cluster: Executor and Driver operate on the Yarn cluster.

The three operation modes are different in that they have different resource allocation methods and different task scheduling algorithms to perform computing tasks. But they have the same workflow. Figure 2 is the workflow of Spark.

Each Spark program has its own Executor process, but each Executor process has multiple threads corresponding to it. This parallel resource allocation and scheduling mode is conducive to resource sharing between different Spark programs and greatly improves execution efficiency.

Figure 3 shows that the program uses the action to trigger the job. The job builds the DAG graph based on the dependencies of RDD, the built DAG graph is then resolved and built into different stages by the DAG Scheduler, and the dependencies between stages are calculated. Then, the Task Scheduler assigns the Task Set to the work set and divides it into multiple threads for parallel computation, and the results are returned to the Task Scheduler and then to the DAG Scheduler [20]. When the results are completed, they are returned to the driver or saved in the external storage system, and all resources are released.

3. The D2HUP and EFIM Algorithm Parallelization Based on Spark

3.1. Parallel D2HUP Algorithm Based on Data Partition

3.1.1. Main Ideas. The main design ideas for the parallelization of the D2HUP algorithm based on data partition are as follows. First, read database D and minUtil, divide dataset D into N pieces on average, and distribute N pieces of data to different nodes for calculation. The minimum threshold of each node is minUtil/n. If an itemset is not an efficient itemset on all nodes, its total utility on all nodes must be less than minUtil. Therefore, the D2HUP algorithm is used to mine candidate efficient itemsets on different nodes simultaneously, and each efficient itemset appears in at least one node. The second step is to rescan the database, calculate the real utility value of the candidate itemset obtained in each node, eliminate the itemset whose threshold value is less than minUtil, and get the final result.

3.1.2. Algorithm Design. The following steps are used to implement the D2HUP algorithm based on data partition on Spark:

Step 1: Read dataset D into RDD. Each row in the RDD stores a transaction in dataset D.

Step 2: Partition the RDD. The repartition() method is called to group the original RDD, and the different number of groups affects the parallelism of the program.

Step 3: Execute the D2HUP algorithm in parallel. The D2HUP algorithm is executed simultaneously for the grouped data obtained in the second step.

Step 4: Use broadcast() to broadcast the results.

Step 5: Iterate through the database, compare each transaction with the candidate itemset, and use the results reduceByKey(), adding the utilities of the same itemset.

Step 6: Execute filter() to remove all unmet conditions in RDD, and the final RDD is the final result.

3.2. The D2HUP Parallel Algorithm Based on Tree Structure and Spark

3.2.1. The Parallelization Idea of the D2HUP Algorithm. The main design ideas of the D2HUP parallel algorithm are as follows. First, the database D is read, the entire database is compressed in CAUL, and each item satisfying the condition in CAUL is prefixed to conduct efficient parallel itemset mining. Because each prefix item contains a separate extension suffix, each item in the original CAUL can be grouped for parallel mining tasks. Each prefix item in CAUL is assigned to different compute nodes, and different compute nodes simultaneously mine the efficient itemset of different prefix items. After the mining task of each Worker node is completed, the mined high-efficiency itemset will be sent to the master node for collection. Each node is mined independently, so this pattern does not have much communication overhead.

The essence of the D2HUP algorithm is a depth-first search tree, as shown in Figure 4. In the process of mining efficient itemsets of different prefix trees, each prefix item is treated as a child node, and the child node and its sibling nodes are independent of each other and do not affect each other. In the mining process, there is no need to exchange information, so the efficient itemset with different prefixes will only appear once on all branches. This is also an important reason for the parallel implementation of the D2HUP algorithm.

3.2.2. D2HUP Algorithm Parallelization Based on Tree Structure. Spark is a programming model based on MapReduce, parallelization of the D2HUP algorithm is to transform CAUL into RDD (Resilient Distributed Dataset), and the associated transforms and actions are called to parallelize efficient itemset mining. Implementing the D2HUP algorithm on Spark has the following steps:

Step 1: Read the dataset and build the initial linked list CAUL.

Step 2: Convert CAUL to RDD; because Spark operates on RDD, you need to convert CAUL to RDD, where each row stores a prefix item.

Step 3: According to the RDD in Step 2, divide the data into partitions. The number of partitions affects the degree of parallelism of the task.

Step 4: Execute the D2HUP algorithm in parallel. According to the grouping of Step 3, prefix items in each partition are grouped for efficient itemset mining. The specific steps are the same as the serial D2HUP algorithm. In the process of efficient itemset mining for each group, only the efficient itemset of that group needs to be generated because the prefixes allocated by

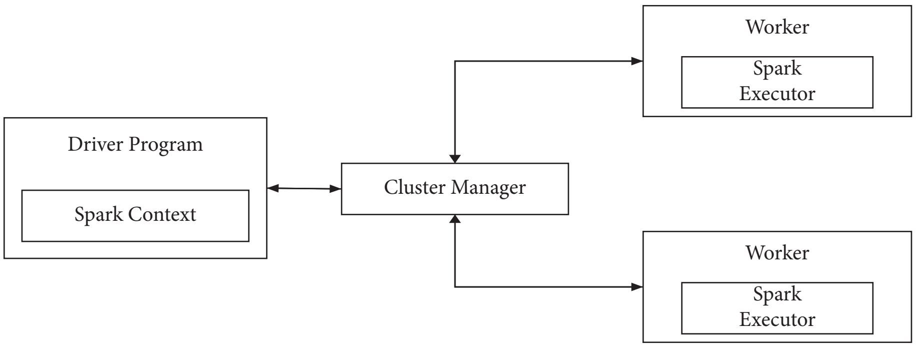

Figure 2: Spark workflow diagram.

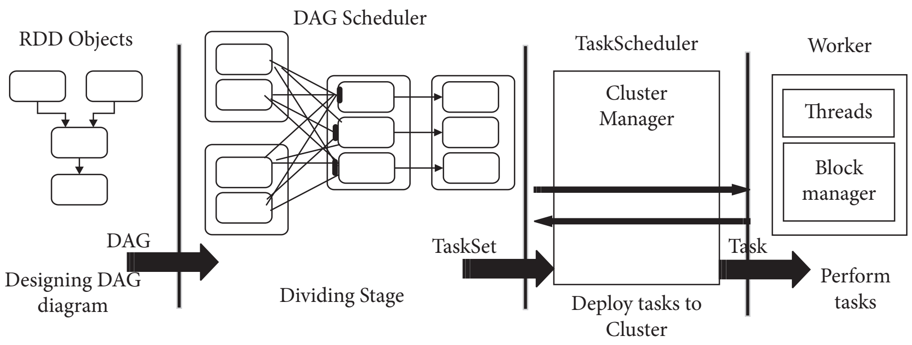

Figure 3: Spark runtime architecture.

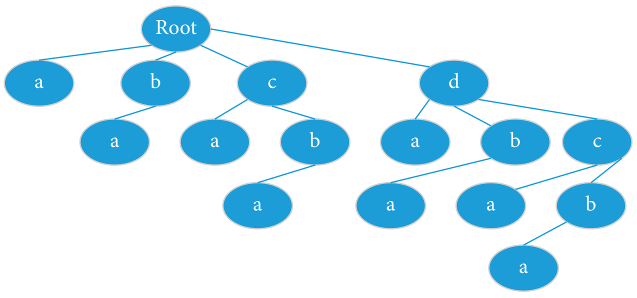

Figure 4: D2HUP depth-first search tree.

different groups are different, and there is no influence between prefixes.

Step 5: Aggregate the results. After the completion of the fourth step, all the efficient itemsets of the prefix branch are obtained for each node. Through the aggregation operation, the efficient itemsets of each node are gathered together to get the final result.

3.3. The EFIM Parallel Algorithm Based on Spark. EFIM algorithm is also a depth-first and efficient itemset mining algorithm. After scanning database D and constructing Primary (α) and Secondary (α) items about root node α, the algorithm recursively calls Search() method for efficient itemset mining. This step has been tested to take most of the program's time, so consider parallelizing the Search() method. In the Search() method called for the first time, each item $i \in$ Primary (α) is a prefix. The traversal of item i is essentially a depth-first search tree, so the extensions contained in each prefix are independent. Therefore, all items i can be grouped, and multiple compute nodes can simultaneously mine the efficient itemset with item i as the root node. Note that the term i assigned to different nodes is different. After the efficient itemsets of all nodes are mined, the results are summarized to the master node. The result is the final efficient term set.

Similar to the parallelization of the D2HUP algorithm, the EFIM algorithm is also a depth-first search tree [21], as shown in Figure 5. Each child of the root node is different and has been sorted in TWU size. Therefore, this child node and other sibling nodes are independent and independent of each other. Each candidate itemset can only appear once in the whole depth-first search tree, so the efficient itemset of the child node is the whole efficient itemset. This is also an important reason why the EFIM algorithm can be parallelized [12].

The parallelization of the EFIM algorithm is mainly concerned with the parallelization of the process of searching the efficient itemset recursively. After the whole database is scanned, the generated Primary (α) is transformed into RDD and grouped, and Action and Transform operators are called to perform parallel utility itemset mining on each subnode. The EFIM algorithm based on Spark mainly includes the following steps:

Step 1: Read dataset D.

Step 2: Calculate the local utility of each item, and then build the Primary (α) and Secondary (α).

```
input: $D$: a transaction database
output: the set of high-utility itemsets
  {
  $\alpha = \varphi$
  Calculate TS ({}) for all items $i \in I$ by scanning $D$;
  CAULRDD = JavaSparkContext parallelize (Tscaul ({}));
  PartitionRDD = MapPartitions (CAULRDD, partitions)
  Foreach partitionRDD do
  d2hupAlgorithm (prefix, prefixlength, CAUL, prefixSupport, minutil)
  }
Output = prtitionRDD.collect ()
}
```

Algorithm 1: The tree-structured D2HUP algorithm.

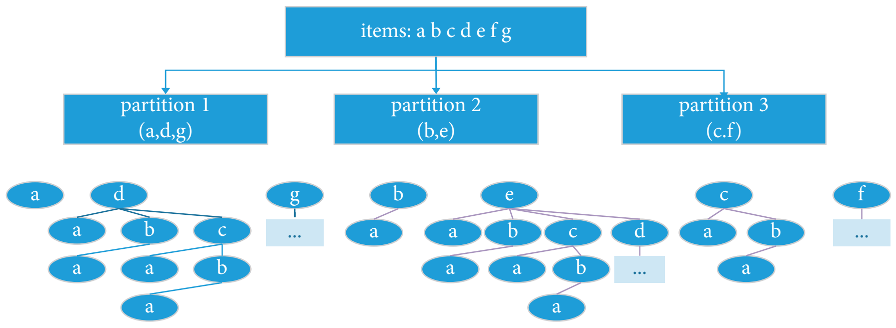

Figure 5: The EFIM algorithm structure based on Spark.

Step 3: Primary (α) is converted to RDD on the first call to the Search() method, and there is a line in RDD for each item $i \in \alpha$.

Step 4: Partition the transformed RDD. The number of partitions affects the degree of parallelism of the algorithm.

Step 5: Parallel EFIM algorithm is executed in each partition. According to the grouping of Step 3, the prefix item i of each partition is different, so the efficient items contained are also different. The specific steps are the same as the serial algorithm of EFIM. After the completion of this step, each group of efficient itemsets prefixed with item i will have unique grouping results of different prefixes, so they will not affect each other.

Step 6: After the result aggregation is completed in Step 4, the high-efficiency items of each group obtained are gathered to the master node and persisted to the hard disk or printed out.

3.4. The Result of the Experiment. The parameters that have a great influence on Spark-based efficient mining algorithm are as follows:

(1) minUtil, that is the threshold value given by the user. Different threshold settings affect the number of efficient itemsets that meet the conditions finally mined. The threshold designed for the experiment in this paper refers to the threshold set for datasets in literature.

(2) The size of the dataset. In this paper, we use the common open dataset of the mining domain of the high efficient term set (Chess, Pumsb, BMS, Connect, Accidents, Kosarak, and Chainstore) and four simulated datasets. The real datasets can be downloaded from the UCI or SPMF.

(3) The influence of the Spark parameter. There are several parameters that have a great impact on Spark. The total number of Executor processes in a cluster is executors, the number of single Executor processes is executor-cores, and the number of data partitions. The experiments carried out in this section aim at these three parameters and perform performance experiments by setting different thresholds.

3.4.1. The Influence of Partition Parameters on the Algorithm. To study the effect of partition number on experimental results, set other parameters in Spark to a fixed value, as shown as follows: executors = 4, driver-memory = 100 GB, executor-cores = 10, and executor-memory = 100 GB. The effect of different partition numbers on the running efficiency of the program is shown in Figures 6–8.

Figure 8 shows the influence of partition parameters on the running efficiency of the program, where the triangular broken line is the parallel EFIM algorithm, and the dotted broken line is the parallel D2HUP algorithm. As can be seen from the figure, the running efficiency of parallel EFIM and D2HUP programs is gradually accelerated as the number of

```
input: $D$: a transaction database
minutil: a user-specified threshold
output: the set of high-utility itemsets
  {
  $\alpha = \varphi$
  Calculate Primary ($\alpha$) and Secondary ($\alpha$) for all items by scanning, using a utility-binary; $i \in I$
  itemsToExploreRDD = JavaSparkContext.Parallelize (item $i \in primary(\alpha)$)
  partitionRDD = MapPartitions (itemsToExplorRDD, partitions)
  for each partitionRDD do
  Search ($\alpha$, $D$, Primary ($\alpha$), Secondary ($\alpha$), minutil)
  }
Output = partitionRDD.collect ()
}
```

Algorithm 2: The EFIM algorithm based on Spark.

```
    input: $\alpha$: an itemset,
    $\alpha$-$D$: the $\alpha$ projected database,
    Primary ($\alpha$): the primary items of $\alpha$,
    Secondary ($\alpha$): the secondary items of $\alpha$,
    minutil: threshold.
    output: the set of high-utility itemsets that are extensions of $\alpha$
(1) for each item $i \in primary(\alpha)$ do
(2) $\beta = \alpha \cup \{i\}$
(3) Scan $\alpha$-$D$ to calculate $u$ ($\beta$) and create $\beta$-$D$
(4) if $u$ ($\beta$) $\geq$ minutil, then output $\beta$
(5) Calculate su ($\beta$, $z$) and lu ($\beta$, $z$) for all item $z \in$ Secondary ($\alpha$) by scanning $\beta$-$D$ once, using two utility-bin arrays.
(6) Primary ($\beta$) = {$z \in$ Secondary ($\alpha$)|su ($\beta$, $z$) $\geq$ minutil}
(7) Secondary ($\beta$) = {$z \in$ Secondary ($\alpha$)|lu ($\beta$, $z$) $\geq$ minutil}
(8) Search ($\beta$, $\beta$-$D$, Primary ($\beta$), Secondary ($\beta$) minutil);
(9) end
```

Algorithm 3: The Search procedure.

partitions increases. It is worth noting that the threshold value (minUtil) in the experimental process is user-defined by the user, and the itemset larger than the threshold value is regarded as the efficient itemset. The threshold value is set with reference to the dataset in reference [14].

3.4.2. *The Influence of Executors Parameters on the Algorithm.*

To verify the effect of executors on the running time of the algorithm, set all parameters other than executor to a fixed value: driver-memory = 100 GB, executor-cores = 2, executor-memory = 20 GB, and partition = 16. The results are shown as follows.

Tables 1 and 2 show the impact of executors' parameter on the running efficiency of the program on different datasets. It can be found from the above table that, with the increase of executors, the running time of both algorithms decreases to some extent.

3.4.3. *The Influence of Executor-Cores Parameters on the Algorithm.*

To verify the impact of executor-cores on algorithm runtime, parameters other than executor-cores are set as fixed values: driver-memory = 100 GB, executors = 4, executor-memory = 20 GB, and partition = 16. The results are shown as follows.

Tables 3 and 4 show the influence of executor-cores on the operating efficiency of programs on different datasets. It can be found from the above table that, with the increase of executor-cores, the operating time of both algorithms decreases to some extent.

3.4.4. *Time Ratio of Parallel Algorithm to Serial Algorithm.*

Under the same threshold condition, serial time and partition parameter are set to 16 when both algorithms are parallel program time, as shown in Table 5.

Parallelization D2HUP/EFIM represents the ratio of the parallel time of two algorithms. From the above experiments, when the dataset is small, the acceleration effect is not obvious because the serial program consumes less time, while the parallel program uses Spark for startup time and other reasons. But when the dataset size increases, the serial program consumption time increases. Compared with the serial algorithm, the Spark D2HUP parallel and EFIM parallel algorithms have greatly improved the running time.

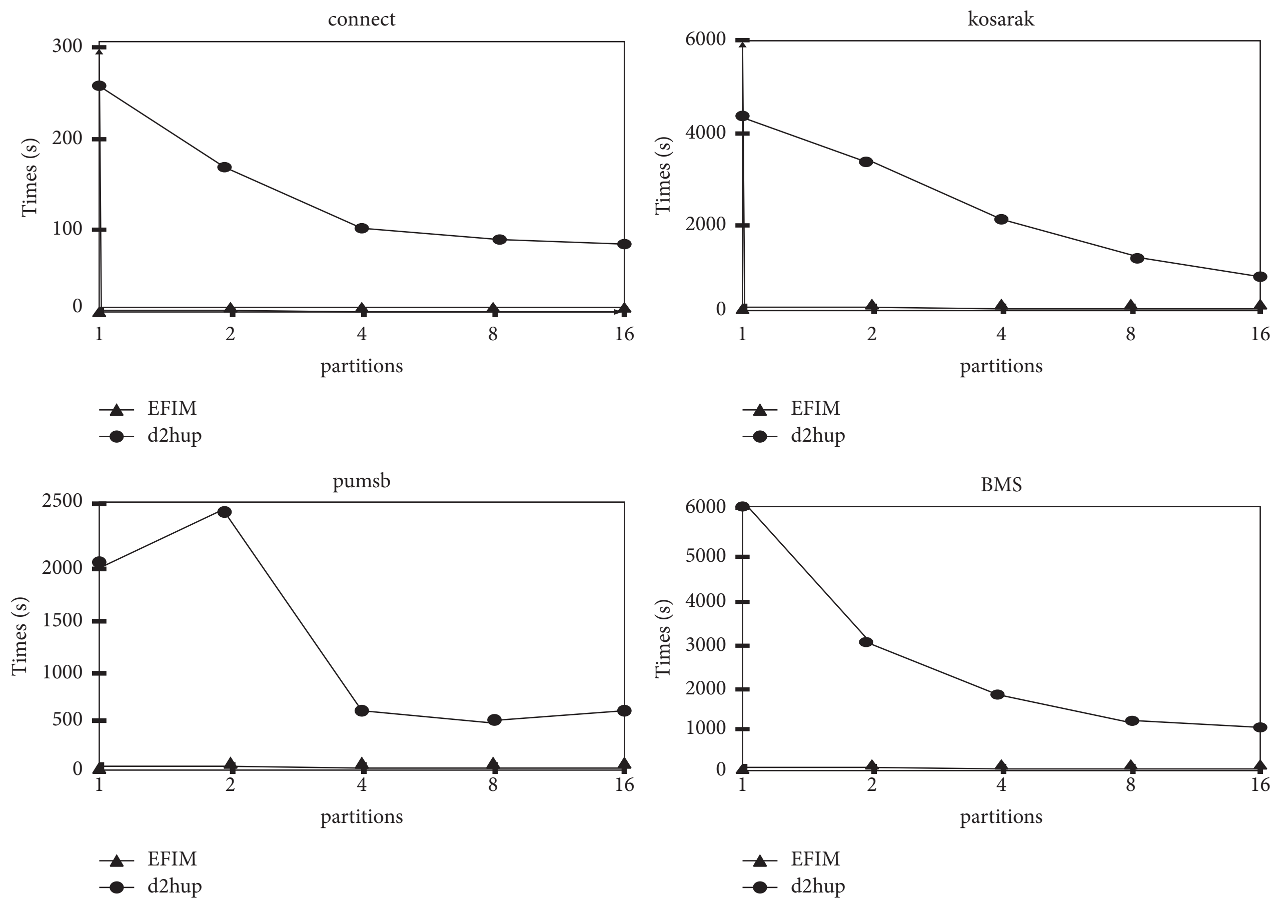

Figure 6: The influence of partition parameters on the algorithm.

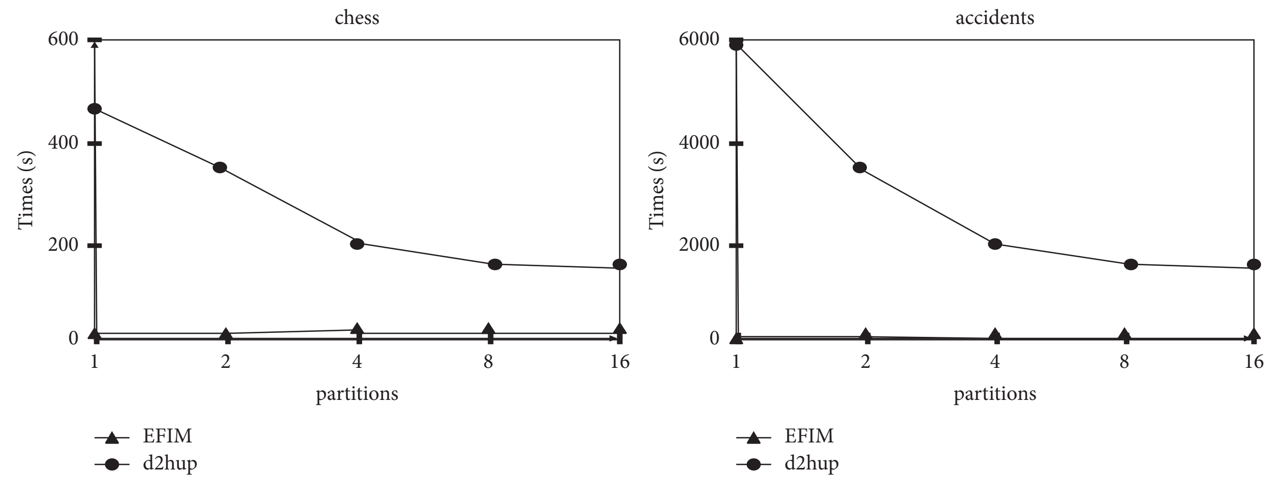

Figure 7: The influence of partition parameters on the algorithm.

4. Design of Automatic Selection of the Most Efficient Itemset Algorithm Based on Spark

4.1. Design of Algorithm. In order to realize automatic selection of the fastest efficient itemset mining algorithm, this paper adopts the data-driven method to carry out experiments. First, different dimensional features of the dataset are constructed, and these dimensional features are taken as attributes of the classification algorithm. Second, all datasets were tested on EFIM and D2HUP algorithms, respectively, and the running time was recorded. Finally, the running results of EFIM and D2HUP are taken as labels for prediction.

The flowchart of the system algorithm is shown in Figure 9.

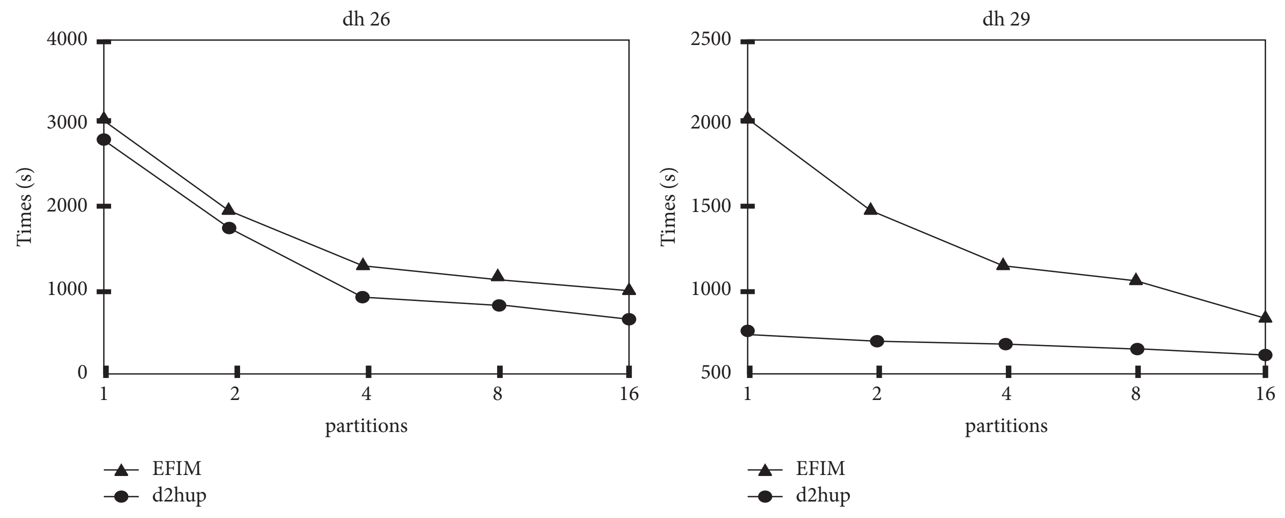

Figure 8: The influence of partition parameters on the algorithm.

Table 1: Different executive time of executors in the EFIM algorithm.

Datasets	4	8	12	16
dh26	1410 (s)	1660 (s)	1423 (s)	1106 (s)
dh29	1233 (s)	1066 (s)	1000 (s)	842 (s)

Table 2: Different executive time of executors in the D2HUP algorithm.

Datasets	4	8	12	16
dh26	233 (s)	228 (s)	170 (s)	120 (s)
dh29	543 (s)	438 (s)	448 (s)	440 (s)

Table 3: Different executive time of executor-cores in the EFIM algorithm.

Datasets	4	8	12	16
dh26	2196 (s)	1411 (s)	1001 (s)	936 (s)
dh29	1246 (s)	1233 (s)	862 (s)	810 (s)

Table 4: Different executive time of executor-cores in the D2HUP algorithm.

Datasets	4	8	12	16
dh26	270 (s)	233 (s)	180 (s)	173 (s)
dh29	553 (s)	543 (s)	548 (s)	536 (s)

Given a dataset *D*, build the following feature dimensions of the dataset, which are shown in Table 6.

In this paper, the mainstream classification algorithm logistic regression, C45, random forest, and SVM were used to predict the constructed datasets, extract the classification rules, and draw conclusions through analysis.

Table 5: The time acceleration ratio of serial to parallel.

Datasets	D2HUP speed-up radio	EFIM speed-up radio	Parallelization D2HUP/EFIM
Chess	2.97	0.24	8.97
Pumsb	2.74	1.03	16.70
BMS	1.51	0.36	544.13
Connect	2.23	0.68	27.32
Accident	2.26	1.06	50.54
Kosarak	2.95	11.64	20.08
dh26	3.10	9.48	0.17
dh29	1.45	4.18	0.71

4.2. Design of Experiment. The exposed datasets of the efficient itemsets are Accidents, BMS, Chess, Connect, Foodmart, Mushroom, Chainstore, Pumsb, Kosarak, and so on. However, due to the small sample size of the real dataset, the following strategies were adopted to construct the simulated dataset:

(1) Generate simulated datasets. Given the five parameters of maximum rows, maximum columns, item number, and maximum utility value of the dataset, 19 simulated datasets were generated by dividing them into fixed columns and random columns.

(2) Split nine real datasets in proportion. The real dataset was split into 20%, 40%, 50%, 60%, and 80% and divided into random percentage and fixed first few hundredths to generate 90 split datasets

(3) There are nine real data sets.

According to the above two strategies, a total of 118 datasets were generated; the EFIM algorithm and D2HUP algorithm were used for experiments in the above datasets to record the running speed. If the EFIM algorithm is fast, the tag is 1. If the D2HUP algorithm is fast, the tag is 0.

K-folded is a method of cross validation.

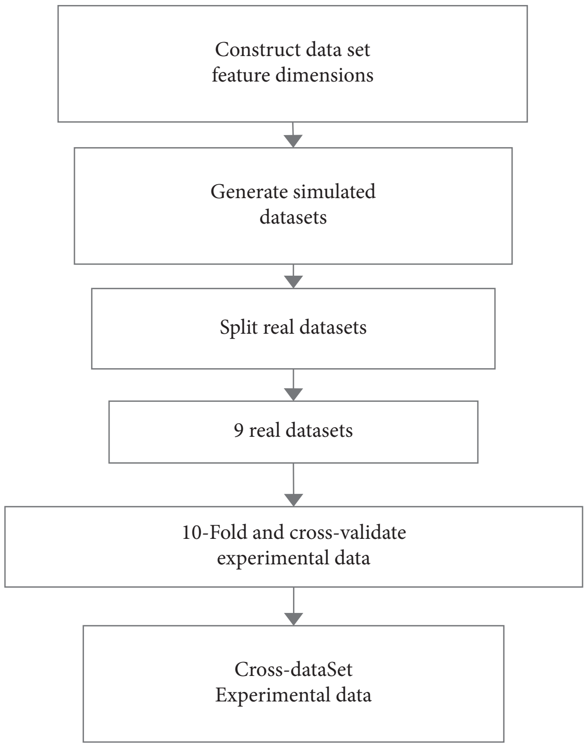
Construct data set feature dimensions
Generate simulated datasets
Split real datasets
9 real datasets
10-Fold and cross-validate experimental data
Cross-dataSet Experimental data

Figure 9: The flowchart of the system algorithm.

Table 6: Construction of data dimensions.

D	Transaction database *D*
I	The set of all the different terms in database *D*
dbTrans	Total number of transactions in database *D*, |D|
dbItems	The total number of different terms in database *D*, |*I*|
transMaxLen	The total length of the longest transaction in database *D*
transMinLen	The total length of the shortest transaction in database *D*
dbUtil	The sum of all transaction utilities in database *D*
sumItems	The total number of terms
itemsAvg	The average length of items in database *D*, sumItems/dbItems
transLen1	The number of transactions of length 1 in database *D*
transLen2_5	The number of transactions of length 2 to 5 in database *D*
transLen100	The number of transactions in database *D* with a length greater than 100
itemWightAvg	The average weight of the utility of the item in database *D*, totalWeight/sumItems
transAvg	Average transaction length in database *D*, sumItems/dbTrans
maxUtility	The maximum value of all utilities in database *D*
minUtility	The minimum value of all utilities in database *D*
fileSize	Database file size
Density	Database *D* sparse degree, transAvg/dbTrans

In general, we evaluate the generalization error of the model through experimental tests so as to select the learner with better performance. The dataset was divided into a training set and a test set, through which the machine learning model was obtained, and the performance of the model was evaluated. Due to the limited number of actual samples, in order to reuse the data, this experiment uses the k-fold cross-validation method.

Table 7: The result of the experiment.

Algorithm 10	Cross-validation accuracy (%)
Simple logistic	92.36
C45 (J48)	90.68
Random forest	90.68
SVM (SMO)	91.52

Cross-validation refers to dividing the dataset D into k mutually exclusive subsets of similar size. In the process of dividing, each subset shall try to maintain the consistency of data distribution. Each time, one subset of $k-1$ is taken as the training set and the rest as the test set. In this experiment, k equals 10.

4.3. The Result of the Experiment. The obtained datasets were trained on logistic regression, C45, random forest, and SVM, respectively, and the experimental records were recorded using 10-fold cross-validation. The experimental results are shown in Table 7.

Figure 10 shows the decision rules extracted from the C45 classification algorithm generation model. Given a dataset D, the user can decide how to choose the optimal mining algorithm for a high-consumption itemset according to the decision rules generated in Figure 10. First, if the number of transactions with a length of 6–10 in database D is no more than 490, then the EFIM algorithm is selected for the experiment (tag 1). Otherwise, for further judgment, if

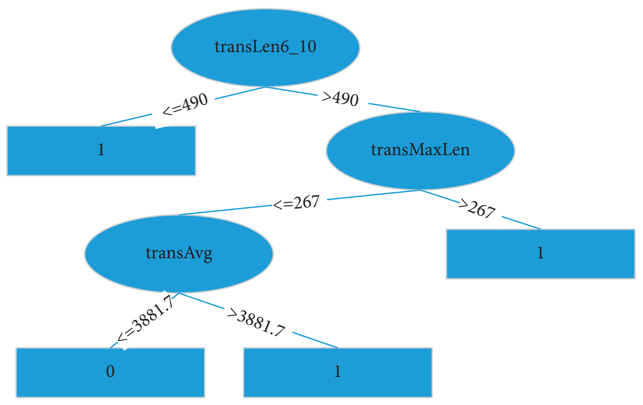

Figure 10: Extraction rule.

the maximum transaction length (transMaxLen) in the database is greater than 267, then the EFIM algorithm is selected for the experiment (tag 1). Determine whether the average length of items in the database (itemsAvg) is greater than 3881.7. If it is greater than 3881.7, the EFIM algorithm is selected; otherwise, the D2HUP algorithm is selected. It is worth noting that, according to Figure 10, the important parameters are the number of transactions with a length of 6–10 in database D (transLen6, 10), the average length of items in database D (itemsAvg), and the length of the longest transaction in the database (transMaxLen).

4.4. Cross-Dataset Experiment Result. In order to further verify the classification model based on the Spark set of high efficiency with the prediction accuracy of the parallel algorithm, 118 will generate the dataset of eight datasets on the Spark of the results as a test dataset (Chess, Pumsb, BMS, Connect, Accidents, Kosarak, dh26, dh29, etc.) and the rest of the dataset as the training dataset for training method (left); the final classification results are obtained as follows.

Through experiments, it can be concluded that the classification algorithm model based on serial D2HUP and EFIM algorithm also has a good prediction of the running results of parallel programs.

In large-scale datasets, the efficiency of efficient itemset mining is an urgent problem to be solved. This paper parallelizes the serial algorithm based on the Spark platform, which greatly improves the running efficiency of the algorithm. In this paper, parallel D2HUP and EFIM algorithms are designed and implemented. The parallelization of an efficient itemset based on Spark is realized to reduce the running time of the mining algorithm.

5. Conclusions

In large-scale datasets, the efficient mining of itemsets is an urgent problem to be solved. In this paper, the serial algorithm is parallelized based on the Spark platform, which greatly improves the running efficiency of the algorithm. At the same time, since the D2HUP algorithm and EFIM algorithm have their own advantages and disadvantages in different datasets, and their running times are greatly different, it is a meaningful work to select an algorithm for efficient itemset mining in a given dataset.

The main work contents of this paper are as follows:

(1) By studying serial D2HUP and EFIM algorithms, design and implement parallel D2HUP and EFIM algorithms. Among them, the D2HUP algorithm realizes two different versions based on data partitioning and structure partitioning, and through large-scale experimental comparative analysis, it is concluded that the parallel efficient itemset mining algorithm has a greater improvement in running time efficiency than the serial algorithm.

(2) In this paper, 118 datasets were obtained by generating and sorting out the running time of the two algorithms in the real and simulated datasets. The length, sparsity, and size of each data were taken as the features, and the running time was taken as the prediction target to establish the model. The decision rules are extracted from the model so that users can better choose the fastest and most efficient itemset mining algorithm to conduct experiments based on the given rules.

In conclusion, the automatic selection efficient itemset algorithm based on Spark has a fast running time and the most accurate prediction.

Data Availability

There are commonly used Accidents in public data sets of efficient use, such as BMS, Chess, Connect, Foodmat, Mushroom, Chainstore, Pumsb, and Kosarak. However, due to the small number of samples of real datasets, it is necessary to build simulated datasets. Please refer to the experimental design for the construction of simulated data. The constructed simulated dataset was verified and compared on Spark based the EFIM algorithm and D2HUP algorithm, and the experimental dismissal was obtained.

Conflicts of Interest

The authors all declare that they have no conflicts of interest.

Acknowledgments

This research was supported by the Scientific Research Project of Education Department of Hunan Province: "Research on Cloud Storage Security Architecture Technology Based on Blockchain" (Grant no. 21C1563).

References

[1] Q. H. Duong, P. Founier-Viger, H. Ramampiaro, K. Nørvåg, and T. L. Dam, "Efficient high utility itemset mining using buffered utility-lists," *Applied Intelligence*, vol. 48, no. 12, pp. 1–19, 2017.

[2] G. Chen, H. Liu, L. Yu, and Q. X. Wei, "A new approach to classification based on association rule mining," *Decision Support Systems*, vol. 42, no. 2, pp. 674–689, 2006.

[3] G. C. Lan, T. P. Hong, and V. S. Tseng, "Projection-based utility mining with an efficient indexing mechanism," in *Proceedings of the International Conference on Technologies and Applications of Artificial Intelligence*, pp. 137–141, IEEE, Hsinchu, Taiwan, November 2011.

[4] A. dankon and M. Cheriet, "Support vector machine," *Computer Science*, vol. 1, no. 4, pp. 1–28, 2002.

[5] H. Yao and H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data & Knowledge Engineering*, vol. 59, no. 3, pp. 603–626, 2006.

[6] J. Widom, "ACM SIGMOD international conference on management of data," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of data*, p. 373, New York, NY, USA, June 1996.

[7] C. Engle, A. Lupher, R. Xin et al., "Shark:first data analysis using coarse-grained distributed memory," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 55–62, Scottsdale, AZ, USA, May 2012.

[8] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 55–64, Hawaii, Maui, USA, Octomber 2017.

[9] W. Song, Y. Liu, and J. Li, "BAHUI: fast and memory efficient mining of high utility itemsets based on bitmap," *International Journal of Data Warehousing and Mining*, vol. 10, no. 1, pp. 1–15, 2014.

[10] A. Gittens, A. Devarakonda, E. Racah et al., "Matrix Factorizations at scale: a comparison of scientific data analytics in Spark and C+MPI using three case studies," in *Proceedings of the IEEE International Conferences On Big Data*, pp. 204–213, IEEE, Washington, DC, USA, December 2017.

[11] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: a frequent-pattern tree approach," *Data Mining and Knowledge Discovery*, vol. 8, no. 1, pp. 53–87, 2014.

[12] Y. Liu, W. Liao, and A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Proceedings of thePacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pp. 689–695, Springer-Verlag, Hanoi, Vietnam, May 2015.

[13] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graph X a resilient distributed graph system on Spark," in *Proceedings of the International Workshop on Graph Data Management Experiences and Systems*, pp. 1–6, New York, NY, USA, June 2013.

[14] T. White and D. Cutting, "Hadoop: the definitive guide," *O'Reilly Media Inc Gravenstein Highway North*, vol. 215, no. 11, pp. 1–4, 2012.

[15] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS-Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[16] G. Holmes, A. Donkin, and I. H. Witten, "WEKA: a machine learning workbench," in *Proceedings of the 1994 Second Australian and New Zealand Conference on*, pp. 357–361, IEEE, Brisbane, QLD, Australia, November 2022.

[17] A. Inokuchi, T. Washio, and H. Motoda, "An Apriori-based algorithm for mining frequent substructures from graph data," in *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 13–23, Springer-Verlag, Lyon, France, September 2010.

[18] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2371–2381, 2015.

[19] X. Meng, J. Bradley, B. Yavuz et al., "MLlib:machine learning in Apache spark," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2015.

[20] U. Yun, H. Ryang, and K. H. Ryu, "High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3861–3878, 2014.

[21] S. Zida, P. Fournier-Viger, C. W. Lin, C. W. Wu, and V. S. Tseng, "EFIM :a fast and memory efficient algorithm for high-utility itemset mining," *Knowledge & Information Systems*, vol. 51, pp. 1–31, 2016.