10.18686/aitr.v2i3.4406

An Improved Apriori Algorithm Based on the Spark Platform

Congshuai Xia, Gang Fang, Wenqiang Gao, Qian Zhao

School of Computer Science and Engineering, Chongqing Three Gorges University, Wanzhou Chongqing 404020

Abstract: Apriori algorithm is one of the most classical algorithms used to mine frequent term sets, but because of the need to scan the calculation method of transaction set repeatedly, the computational efficiency of the algorithm is seriously reduced and it is difficult to parallelize processing. With the advent of the era of big data, the data scale is increasing. In order to solve this problem, this paper proposes a Apriori parallelization processing method based on vertical data format and Spark computing framework — GC-Apriori algorithm. Using the vertical data format, reduce the duplication between things, improve the efficiency of data storage, and the efficiency of frequent item set mining. At the same time, the broadcast variable mechanism of Spark is used to improve the overall computing efficiency. Comparing the performance with other distributed Apriori algorithms on the same scale, the computational efficiency of GC-Apriori algorithm is improved. The results show that the algorithm effectively improves the efficiency of frequent term set mining of Apriori algorithm in distributed environment.

Keywords: Apriori algorithm; Spark; Vertical data format

1. Introduction

In this paper, we propose an efficient data mining method combining the Apriori algorithm with the Spark platform. As the Apriori algorithm, the frequent set is mined by scanning the data set and calculating the support of the item set. However, traditional Apriori algorithms experience inefficiency in handling large-scale data.

The emergence of big data platforms such as Hadoop and Spark has provided new ideas for the parallelized computing of Apriori algorithm. However, the MapReduece model of Hadoop needs to read and write disks repeatedly, which is not suitable for iterative algorithms, so many improved parallelized Apriori algorithms and variant algorithms based on Hadoop[1]By adding CPU and memory in a distributed form, you can process large data sets and calculate the results, but the efficiency of the algorithm is still mediocre. Spark's memory-based computing model is undoubtedly a better solution. At present, there are many optimization work on Apriori algorithm based on Spark[2]Compared with previous Hadoop based improvements have achieved better results, but there is still the possibility of repeated scanning in the dataset.

The main contribution of this paper are as follows: a) a improved Apriori algorithm based on Spark platform: GC-Apriori, according to the vertical data format compression transactions, each project and its transaction ID set association, to reduce data redundancy and improve the efficiency of frequent set computing, combined with Spark distributed computing framework, design new parallelization implementation method, while using broadcast variable mechanism, the compressed transaction data efficiently distributed to each computer node, improve data access efficiency, improve the efficiency of parallel computing. B) Application to the public data set to verify the effectiveness and time efficiency of the algorithm, compare the algorithm with similar algorithms in the big data environment, prove its advantages in distribution, and also explain the defects of the algorithm and analyze the ideas of further optimization.

2. Spark

The Spark platform is an important framework used for big data computing based on the Hadoop platform, and it has a more obvious advantage when processing large-scale data. Spark is designed based on MapReduce parallel computing mode. Based on Hadoop, it not only inherits the excellent distributed computing characteristics of MapReduce, but also introduces a new core data structure RDD (Resilient Distributed Datasets, elastic distributed data set)[3].

Spark defines a series of operational operations on the RDD, which are called operators. RDD has two types of operators: the Transformation operator and the Action operator. The Transformation operator is used to transform operations on the RDD to generate a new RDD.

These Transformation are lazy computed, that is, they are not executed immediately, but record the logical plan to build the calculation execution until the action operator (Action) is called. The Action operator is used to trigger the actual calculation on the RDD and return the results or perform some operation. These operations lead to the execution of the Spark computing link, extracting the data from the cluster. Through this mechanism, the intermediate results with very large amount of data are avoided to various nodes of the system, and the I / O overhead of the system is reduced[4].

3. The improved parallel Apriori algorithm

An improved Apriori algorithm - GC-Apriori algorithm based on Spark platform is proposed. In order to improve the problem of multiple iterations in Apriori algorithm and scan the database, by converting the data into vertical data format, it can generate all the frequent item set after a scan. At the same time, the broadcast variable mechanism is used to reduce the overhead of data transmission between various nodes, and further improve the mining efficiency of frequent item set in distributed computing environment.

It is roughly divided into two main steps in the execution of the algorithm. In the first step, by scanning the data set, the data in horizontal format is converted into vertical data format and generates the frequent 1-item set[5]. These frequent 1-item sets are stored in vertical data structures and are broadcast as broadcast variables (broadcast variable) for Spark. Broadcast variables are shared by all nodes throughout the Spark cluster and can significantly reduce the access time overhead of the data. Since the frequent item set stores each item set and its corresponding transaction ID set, the space required to store the frequent item set is much less than that needed to store the complete data set[6].

In the second step, we iteratively calculate and output the set of frequent k + 1 items. In each iteration, all computational nodes in parallel calculate the generated set of k + 1 terms and their corresponding transaction ID set, and complete the pruning work to eliminate the set of unsatisfying k + 1 terms according to the minimum support, resulting in a new set of frequent k + 1 terms. These frequent k + 1 sets will replace the frequent k-sets in the broadcast variable and start the next iteration until no more advanced frequency sets that meet the minimum support condition.

Figure 1 shows the overall flow of the GC-Apriori algorithm.

The key algorithm pseudocode is as follows:

Algorithm: Frequent item-set mining of vertical data formats and broadcast variables
Input: frequent 1-item set V1 in vertical data format, minimum support min _ sup, total number of transactions count Output: Frequency item set L Hypothesis: sum (x) represents the number of elements in the calculated set x

1 Initialization:
a. V = V1.parallelize() # parallelization
b. L.addAll (V1) # Add the frequent 1 item set to the frequent item set set
2 while V.size > 0:
a. The temp = {} # temporary variable, recording the frequent k + 1 item set
b. foreach i in V:
i1 = i._1 # Item set
i2 = i._2 # Transaction ID set corresponding to the item set
for j in V1:
j1 = j._1 # Item set
j2 = j._2 # Transaction ID set corresponding to the item set
if (i1 ∪ j1).length-i1.length == 1 and sum (i2 j2)> transaction_count * min _ sup: # Judge whether the generated item set is k + 1 item set, and judge whether the item set meets the support condition
c. temp.dropDuplicates() # Distribution needs to be summarized and stressed
d. V = temp # updates the frequent k-item set
e. V1 = spark.sparkContext.broadcast (temp) # Update the broadcast variable for the next iteration
3. End and return L # Return all frequency item sets

4. Experimental results and analysis

4.1 Experimental conditions

Operating system: Windows 11 64 bit

CPU: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz

Memory: 16GB

Experimental platforms: JDK 1.8, Hadoop3.0, Spark3.2.0, and Python3.8

The dataset used in the experiment is the benchmark dataset in the field of frequent term set mining: Movielens-s1, which has 9, 742 records. The comparison algorithm is the Apriori algorithm and the parallel Apriori algorithm based on Spark.

4.2 Effectiveness analysis

In order to verify the correctness of the results generated by GC-Apriori algorithm, the following comparative experiments are designed: to verify whether the frequent item set and total number mined by GC-Apriori algorithm on the benchmark dataset are the same as the results generated by the parallel Apriori algorithm of stand-alone Apriori and Spark. Four support thresholds, 0.5%, 1%, 1.5%, and 2%, are set to see whether each frequent item set generated by GC-Apriori is the same as Apriori and each frequent item set generated by Spark-Apriori, and further validated by the total number of frequent items generated. To ensure experimental rigor, 10 experiments were repeated with each support threshold to avoid contingency errors.

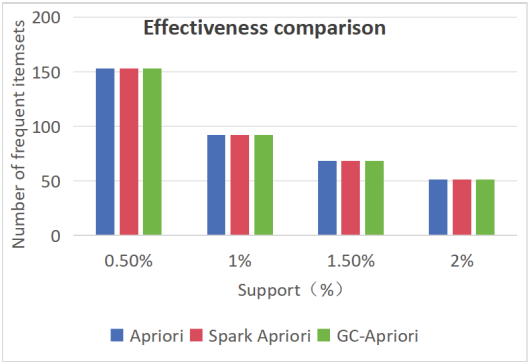

Figure 3: Effectiveness comparison

It can be observed from Figure 3 that the total number of frequent term sets produced by the three algorithms has the same support threshold. Based on the above experiments, the proposed algorithm is effective.

4.3 Time performance analysis

In order to verify the running efficiency of GC-Aprori algorithm, the running time comparison experiment was designed to test the running time on the benchmark data set, taking the average value of the running time of 10 times as the final result, and comparing the running time of the parallel Apriori algorithm of Spark. The experimental results are shown in Figure 4, where the X axis is the support threshold and the Y axis is the running time of the algorithm.

Figure 4 shows the runtime of the two Spark-based Apriori algorithms on the Movielens-s1 dataset at different support thresholds, regardless of the support set, the GC-Apriori algorithm is significantly better than the Spark Apriori algorithm.

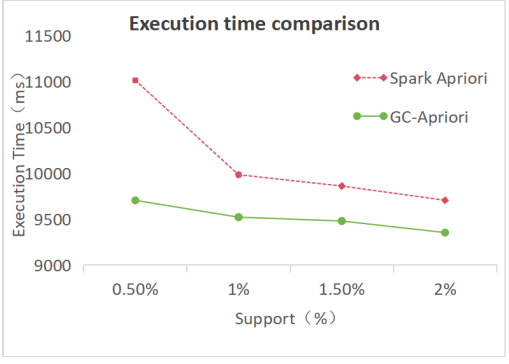

Figure 4. Execution time comparison

5. Conclusion

In this paper, we propose an improved Apriori algorithm, —— GC-Apriori, based on the Spark platform, which improves the efficiency of distributed computation by speeding up computing support and using the broadcast variable mechanism. The experimental results show that the efficiency of GC-Apriori algorithm is improved. The next step will focus on the optimization of the spatial complexity of the algorithm, realizing the common optimization of temporal and spatial complexity.

References

[1] Ye Yanbin, Chiang C C.A parallel Apriori algorithm for frequent itemsets mining[C]//Proc of International Conference on Software EngineeringResearch, Management and Applications.Washington DC:IEEE Computer Society, 2006:87-94.

[2] Qiu Hongjian, GuRong, Yuan Chunfeng, et al.YAFIM:a parallel frequent itemset mining algorithm with Spark [C]//Proc of IEEE International Parallel & Distributed Processing Symposium Workshops Washington DC:IEEE Computer Society, 2014:1664-1671.

[3] Su Weisong. Evaluation of the memory overhead of a typical memory computing system [D]. Southeast University, 2016.

[4] Wu Lei. Research on the frequent pattern mining algorithm in the big data environment [D]. Guangdong University of Technology, 2019.

[5] Xing Changzheng, An Weiguo, Wang Xing. Improvement of frequent itemset algorithm for vertical data format mining[J]. Computer Engineering and Science, 2017, 39(7):1365-1370.

[6] Zheng Jingyi, Deng Xiaoheng. Distributed frequent term set mining algorithm based on item coding [J]. Computer Application Research, 2019, 36 (4): 1059-1067.