Spark-Based Parallel Eclat Algorithm Postprint

Authors: 冯兴杰, Pan Xuan

Date: 2018-05-20T00:00:00+00:00

Abstract

Through in-depth analysis of the Spark big data platform and the Eclat algorithm, this paper proposes a Spark-based Eclat algorithm (referred to as SPEclat). To address the shortcomings of serial algorithms in processing large-scale data, this method introduces improvements in multiple aspects: to reduce the overhead incurred by candidate itemset support counting, the data storage scheme is modified; data is partitioned by prefix and distributed to different computing nodes, thereby compressing the search space and enabling parallel computation. The algorithm is ultimately implemented by harnessing the advantages of the Spark cloud computing platform. Experimental results demonstrate that the algorithm can efficiently process massive datasets and exhibits favorable scalability in the face of large-scale data volume growth.

Full Text

Preamble

A Parallel Eclat Algorithm Based on Spark

Feng Xingjie a,b, Pan Xuan a†

(a. College of Computer Science & Technology; b. Information Network Center, Civil Aviation University of China, Tianjin 300300, China)

Abstract: Through in-depth analysis of the Spark big data platform and the Eclat algorithm, this paper proposes a Spark-based Eclat algorithm (SPEclat). To address the shortcomings of serial algorithms when processing large-scale data, this method introduces improvements in multiple aspects: it modifies data storage structures to reduce the overhead from candidate itemset support counting; it groups data by prefix and distributes it across different computing nodes to compress the data search space and enable parallel computation. The algorithm is ultimately implemented by leveraging the advantages of the Spark cloud computing platform. Experiments demonstrate that the algorithm runs

efficiently when processing massive datasets and exhibits good scalability in the face of large-scale data growth.

Keywords: association rule mining; big data; Spark; projection tree; parallelization

0 Introduction

Frequent pattern mining [1] is one of the important research directions in data mining. In the past, numerous algorithms have been proposed for frequent pattern mining, such as Apriori [2], FP-Growth [3], and Eclat [4]. The rapid development of Internet technologies has accelerated the arrival of the big data information age, bringing about increases in data volume and computational difficulty that have rendered traditional data mining approaches unsustainable. Today, to generate valuable information from large-scale datasets, algorithms must perform repeated iterative computations on raw data, making data mining exceptionally cumbersome. Distributed and parallel computing methods can effectively solve the problems of storing and computing massive data volumes, while also providing resource sharing, high transparency, cost-effectiveness, high reliability, and great flexibility—making them the optimal strategy for processing large datasets.

The MapReduce [5] framework based on Hadoop and the Spark cloud computing platform are the most commonly used distributed programming and computing frameworks. In recent years, many scholars have parallelized traditional algorithms using the MapReduce framework. Compared to the MapReduce framework, the Spark programming framework represents a more refined and advanced implementation of the MapReduce philosophy, making significant improvements upon issues such as excessive rigidity, I/O intensity, unsuitability for constructing iterative algorithms, and low execution efficiency.

Regarding the parallelization of the Eclat algorithm, reference [9] proposed a MapReduce-based parallel Eclat algorithm called PEclat (parallel Eclat). This algorithm requires multiple iterations of Map/Reduce tasks, resulting in lower performance. Reference [10] proposed a prefix-grouping-based parallel mining algorithm called MREclat under the MapReduce framework. When counting support for candidate k-itemsets, this algorithm calculates candidate itemset support by intersecting transaction sequences corresponding to two frequent itemsets used to generate the candidate itemset, which generates substantial computational overhead that grows exponentially with data volume, affecting the algorithm' s practicality. Qiu H et al. were the first to implement parallelization of the Apriori algorithm on Spark [12] and proposed the YAFIM algorithm, which uses horizontal data structures to mine frequent itemsets and solves many problems encountered during parallel implementation of serial algorithms on Spark. However, in the process of obtaining frequent itemsets, it requires multiple traversals of the original dataset, wasting considerable time

on repeatedly traversing unnecessary data, resulting in less than ideal algorithm efficiency.

Eclat is a frequent itemset mining algorithm with good performance in practical applications. This paper proposes a Spark-based Eclat algorithm (SPEclat). The basic idea is: a) transform the original dataset' s horizontal structure into a vertical structure and use BitSet to represent itemset transaction sequences, aiming to improve frequent itemset generation methods and enhance computational efficiency; b) group data by prefix and distribute it to different computing nodes to achieve algorithm parallelization and fully utilize the multi-node, high-performance characteristics of distributed computing environments. Experimental results show that SPEclat has good scalability and speedup, and is more efficient than the MREclat algorithm proposed in reference [10].

1.1 Spark Parallel Computing Framework

Spark is currently one of the most active open-source projects for parallel programming models in the big data domain. Due to its ability to perform rapid calculations through memory-based processing and its features such as RDD persistence and high fault tolerance, the Spark cloud computing platform has become the most suitable tool for iterative computations on large-scale datasets.

The foundation of Spark' s ability to perform memory-based computations lies in its abstraction of in-memory data as Resilient Distributed Datasets (RDDs). RDD is a parallel data structure based on distributed memory that can store user data in memory, control partitioning to achieve data distribution, and possesses dataflow characteristics: automatic fault tolerance, location-aware scheduling, and scalability. Spark also provides rich built-in operations for RDDs, implementing not only conventional map and reduce functions but also more comprehensive operators such as foreach, groupBy, and join. These operators can transform one RDD into another through specific operations, which is the fundamental mechanism of Spark' s memory-based computing. However, Spark' s most important feature is the persistence operation for RDDs. Persisting data in memory eliminates the need for repeated loading into memory or storage to disk for data that requires multiple iterations, greatly accelerating data processing speed. Therefore, the Spark programming framework is particularly suitable for implementing iterative frequent itemset mining algorithms, and parallelization yields substantial performance improvements.

1.2 Frequent Pattern Mining Algorithms

1) Apriori Algorithm: Apriori [2] uses a level-wise complete search iterative algorithm that leverages the anti-monotonicity property of itemsets—based on the principle that "if an itemset is infrequent, all its supersets are also infrequent" —to generate candidate itemsets through join and prune operations, thereby compressing the number of candidate itemsets. This algorithm requires multiple traversals of the original dataset and repeatedly searches through some

redundant transactions in the original dataset without eliminating them, resulting in increased I/O consumption and reduced computational efficiency.

2) Eclat Algorithm: Eclat [4] is a frequent itemset mining algorithm based on vertical data structures (item name; set of transaction identifiers containing the item, Tidset). Using vertical data representation avoids repeated traversals of the original dataset. Itemset support counting is simply the length of the TidSet. Based on concept lattice theory, the algorithm partitions the search space using prefix equivalence relations, generates candidate itemsets through unions of two itemsets meeting certain conditions, and calculates support through intersection of corresponding TidSets to obtain frequent itemsets.

The Eclat algorithm stores and processes data using vertical structures, requiring preprocessing of data organization. The original data is saved in RDDs in the form (Tid, Items), where Items represents all items contained in the transaction. In the map phase, all items in each transaction and their corresponding transaction identifiers form key-value pairs. In the reduce phase, transaction identifiers corresponding to each item are merged. Items with transaction identifier counts exceeding the minimum support threshold become frequent 1-itemsets, and the item along with its transaction sequence are stored.

The Eclat algorithm' s bottom-up search approach is similar to Apriori, but due to different data storage structures, Eclat does not require repeated dataset traversal. However, when calculating candidate itemset support, if the TidSet size is massive, the intersection operation on TidSets consumes substantial time, affecting algorithm efficiency. This paper makes the following improvements: 1) use BitSet to represent transaction sequences, simplifying support counting to bitwise AND operations on corresponding BitSets; 2) group data by prefix and distribute it to different computing nodes to achieve parallel computation.

2 SPEclat Algorithm Based on Spark

The SPEclat algorithm proposed in this paper is a Spark-based implementation of Eclat. The algorithm consists of three steps: a) read data from HDFS, modify data storage structures, filter to obtain frequent 1-itemsets, and store results as RDDs in the memory of cluster nodes; b) iteratively partition frequent itemsets according to identical prefixes; c) in each computing node, generate frequent (k+1)-itemsets from frequent k-itemsets in a bottom-up manner until no more frequent itemsets are produced. In the first phase, this paper uses BitSet to store transaction sequences and replaces transaction set intersection operations with BitSet AND operations to accelerate frequent itemset generation. The second phase distributes obtained data to different computing nodes by prefix. The third phase uses an improved Eclat algorithm on each computing node to solve for all frequent itemsets in its corresponding partition.

2.1 Phase One

First, data stored in HDFS is directly read and saved as Spark RDDs. Map tasks read the data and process it line by line, where each transaction consists of a Tid and all items contained in that transaction. Since the Eclat algorithm stores and processes data using vertical structures, preprocessing of data organization is necessary. The original data is saved in RDDs in the form (Tid, Items). In the map phase, key-value pairs are formed from all items in each transaction and their corresponding transaction identifiers. In the reduce phase, transaction identifiers corresponding to each item are aggregated. Items with transaction identifier counts exceeding the minimum support threshold are filtered to produce frequent 1-itemsets.

The traditional Eclat algorithm stores transactions in the form (ItemSet, TidSet). However, when calculating candidate itemset support, massive data volumes result in large TidSets, and intersection operations on TidSets consume substantial time, affecting algorithm efficiency. To address this problem, the SPEclat algorithm changes the data storage method by storing data as (ItemSet, BitSet), enabling support counting through BitSet AND operations instead of set intersection, thereby avoiding large data volume operations.

The primary task of Phase One is to obtain frequent 1-itemsets by splitting items in transactions to form (Tid, Items) key-value pairs, aggregating transaction identifiers containing each item using the transaction item as the key to form (ItemSet, BitSet) key-value pairs, and finally filtering itemset key-value pairs with support greater than the minimum threshold to obtain frequent 1-itemsets. The pseudocode for this algorithm is shown in Algorithm 1.

Algorithm 1: Obtaining Frequent 1-Itemsets

```
For each transaction t(Tid,Items) in T
  flatMap(line,t)
    Foreach item I in t
      out(I,Tid)
  ReduceByKey(I,Tid)
    while(Item I in partition)
      BitSet+=Tid
    filter( |BitSet| >=support)
    L1+=(I,BitSet)
```

The storage structure conversion is illustrated in Figure 1 [Figure 1: see original paper]. In the converted storage structure, the value indicates whether contains , with 1 if it does and 0 otherwise. At this point, to calculate the occurrence count of a k-itemset in the entire dataset, one only needs to perform AND operations on the BitSets corresponding to these k items in the stored data. The operation result is the transaction identifiers containing this k-itemset, and counting yields its occurrence frequency.

2.2 Phase Two

This phase primarily implements partitioning of frequent itemsets by prefix. Using the generation and partitioning of frequent 2-itemsets as an example, first, frequent 1-itemsets are intersected pairwise to obtain 2-item candidates. Then, the corresponding two BitSets are ANDed, and the newly generated BitSet is counted to obtain frequent 2-itemsets. Frequent 2-itemsets are stored in the form (items, BitSet) and distributed to different computing nodes according to their first-item prefix.

Since BitSet AND operations can be completed in constant time, the overhead for frequent itemset generation is minimal. The pseudocode for data partitioning is shown in Algorithm 2.

Algorithm 2: Partitioning Data by Prefix

```
Read L(2) from RDD
map(items,BitSet)
  get the profix p from items
  out(p, (items, BitSet))
ReduceByKey(p, (items,BitSet))
  get all from items
  Eclat(
    out(frequent Itemset with profix p)
  )
```

To mine frequent itemsets in parallel using the SPEclat algorithm on each node, iterative map/reduce tasks are required, where the map() method implements data partitioning and the reduce phase implements the improved Eclat algorithm. The partition process must be rewritten between mapper and reducer to ensure that columns corresponding to prefixes belonging to the same group are partitioned to the same computing node.

2.3 Phase Three

In this phase, itemsets with identical prefixes are processed in parallel across computing nodes in a bottom-up iterative manner, generating frequent (k+1)-itemsets from frequent k-itemsets. In each node, itemsets assigned to that node are self-joined to generate candidate k-itemsets, followed by support counting.

Since data has already been converted to (ItemSet, BitSet) format in the previous step, BitSet AND operations can be used to count candidate itemset support using two self-joined frequent itemsets in the same node. After the AND operation, the transaction sequence containing the candidate itemset is obtained. This process uses fast BitSet AND operations instead of set intersection operations, saving substantial time. The generated BitSet is then counted to determine if the value exceeds minimum support. If it does, the key-value pair of candidate itemset and corresponding BitSet is stored.

The generated frequent (k+1)-itemsets are then repartitioned, and the SPEclat algorithm is called iteratively to mine frequent itemsets until no more frequent itemsets are generated. Algorithm 3 shows the pseudocode for the algorithm used in Phase Three.

Algorithm 3: Generating Frequent (k+1)-Itemsets

```
Read L(k) from RDD
for each li in L(k)
  for each lj in L(k)
    ItemSet=(li[1],li[2],li[3]…li[k-1],li[k],lj[k])
    BitSet = li.BitSet   lj.BitSet
    if( |BitSet| >=support)
      L(k+1)+=(ItemSet,BitSet)
```

2.4 Time Complexity Analysis

This section verifies the data scalability of SPEclat by comparing its computational efficiency with the MREclat algorithm across different datasets. SPEclat represents the first implementation of the Eclat algorithm on Spark, while MREclat is a classic MapReduce-based implementation of Eclat with superior efficiency compared to other similar algorithms. Therefore, these two algorithms are compared to validate SPEclat' s data scalability.

Certain values must be assumed to express algorithm time complexity in mathematical formulas. Assuming the original dataset contains T transactions, N computing nodes, and t represents the number of frequent 1-itemsets, the original algorithm' s time complexity is $O(t^2 + T/N \times a^2)$. The improved SPEclat algorithm' s time complexity is .

3 Experiments and Analysis

The distributed computing platform for experiments consists of nine computers, with the master node configured as an i7 3770 processor, 4GB memory, and 1TB hard disk. Computing nodes are configured with Pentium e2140 processors, 1GB memory, and 250GB hard disks. The Spark cluster environment parameters are as follows: Ubuntu 14.04; Hadoop version 2.4.0; Spark version 1.6.1.

To verify the correctness of the SPEclat algorithm, its processing results were compared with MREclat on the T10I4D100K, Pumsb_star, and chess datasets, with experiments showing consistent results between both algorithms.

3.1 Speedup Test

To verify whether the SPEclat algorithm is scalable and capable of processing big data, three datasets of sizes 10 GB, 20 GB, and 30 GB with 80 items were artificially generated for subsequent experiments.

Speedup refers to the ratio of time consumed by the same task running on single-processor versus multi-processor systems, serving as an effective metric for measuring parallelization performance. To evaluate the algorithm' s parallelization performance, the algorithm was run on the above datasets with 4, 6, and 8 nodes respectively. The experimental results are shown in Figure 2 [Figure 2: see original paper]. For clear comparison, linear speedup is included.

The experimental results indicate that SPEclat achieves significant performance improvements over MREclat, with performance gains increasing as data volume grows. As shown in Figure 3 [Figure 3: see original paper], for the same dataset, the algorithm' s speedup increases with the number of nodes. Since increased node count gradually increases time consumption for inter-node communication and data transfer, the increasing trend is close to linear but shows some gap from linear speedup, with the gap widening as nodes increase. Comparing speedups across different datasets, larger datasets exhibit speedup closer to linear because computational time accounts for a larger proportion of total time consumption in larger datasets compared to smaller ones. The experiments demonstrate that SPEclat possesses certain scalability.

3.2 Data Scalability Test

To further verify the algorithm' s data scalability, multiple comparative experiments were conducted between SPEclat and MREclat. Public benchmark datasets T10I4D100K, Pumsb_star, and chess were used to ensure experimental objectivity. To ensure result correctness, values in the figures represent averages from 10 algorithm runs.

During scalability measurement, the number of nodes was set to eight, and test data was replicated to 2×, 3×, 4×, 5×, and 6× the original dataset size. Multiple experiments measured the data scalability of both parallel algorithms. The results are shown in Figure 4 Figure 4: see original paper~(c), where the x-axis represents replication multiples of the data and the y-axis represents program execution time. The figures show that for continuously growing data volumes, MREclat' s execution time grows approximately linearly, while SPEclat' s computation time grows more slowly, essentially remaining horizontal. Moreover, SPEclat consumes far less time than MREclat, completing computational tasks at approximately 20 times the rate of MREclat.

This is because SPEclat employs memory-based computing, enabling rapid multiple iterations on datasets in memory and eliminating substantial I/O operations. Simultaneously, when counting support for candidate itemsets, changing the data storage structure enables fast BitSet AND operations to replace set intersection operations, optimizing the support counting method and improving computational efficiency. Compared to MREclat, SPEclat' s performance improvements become increasingly significant as data volume grows.

3.3 Node Scalability Test

To verify SPEclat' s node scalability, experiments were conducted using public benchmark datasets T10I4D100K, Pumsb_star, and chess. The cluster node count was adjusted to 4, 6, and 8 while keeping dataset sizes constant, and SPEclat execution times were recorded. The results are shown in Figure 5 Figure 5: see original paper, (b), and (c), where the x-axis represents the number of nodes in the cluster and the y-axis represents time consumption. Analysis of the three experimental figures shows that as cluster node count increases, SPEclat' s time consumption decreases approximately linearly, demonstrating that the algorithm possesses good node scalability.

4 Conclusion

This paper proposes SPEclat, a Spark-based frequent itemset mining algorithm. Addressing massive data generated in the information age, this paper improves the original serial Eclat algorithm and parallelizes it on the Spark platform, fully leveraging Spark' s memory-based and iterative computation-friendly characteristics to achieve breakthrough performance for mining massive data and solving problems encountered in parallelization such as data partitioning and extensive cross-computations during frequent itemset generation. Experiments using both artificial and public datasets demonstrate that these algorithmic improvements are effective, with algorithm performance becoming increasingly prominent as data volume continues to grow.

References

[1] Han Jiawei, Kamber M, Pei Jian, et al. Data Mining: Concepts and Techniques [M]. Beijing: Mechanical Industry Press, 2012.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases [C]// Proc of International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc., 1994: 487-499.

[3] Zaki M J, Parthasarathy S, Ogihara M, et al. Parallel algorithms for discovery of association rules [J]. Data Mining and Knowledge Discovery, 1997, 1 (4): 343-373.

[4] Zaki M J. Scalable algorithms for association mining [J]. IEEE Trans on Knowledge & Data Engineering, 2000, 12 (3): 372-390.

[5] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [C]// Proc of the 6th Conference on Symposium on Operating Systems Design & Implementation. Berkeley: USENIX Association, 2004: 137-150.

[6] Deng L, Lou Y. Improvement and research of FP-growth algorithm based on distributed Spark [C]// Proc of International Conference on Cloud Computing and Big Data. Washington DC: IEEE Computer Society, 2015: 141-148.

[7] Yang S, Xu G, Wang Z, et al. The parallel improved Apriori algorithm research based on Spark [C]// Proc of the 9th International Conference on Frontier of Computer Science and Technology. 2015: 354-359.

[8] Wang S, Wu P, Liu T, et al. P-WLPA algorithm research on parallel framework Spark [C]// Proc of Information Technology and Artificial Intelligence Conference. 2014: 437-441.

[9] 李伟卫, 赵航, 张阳, 等. 基于 MapReduce 的海量数据挖掘技术研究 [J]. 计算机工程与应用, 2013, 49 (20): 112-117.

[10] 章志刚, 吉根林, 唐梦梦. 并行挖掘频繁项目集新算法——MREclat [J]. 计算机应用, 2014, 34 (8): 2175-2178.

[11] 曹博, 倪建成, 李淋淋, 等. 基于 Spark 的并行频繁模式挖掘算法 [J]. 计算机工程与应用, 2016, 52 (20): 86-91.

[12] Qiu H, Gu R, Yuan C, 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.

[13] 陈明洁. 分布式频繁项集挖掘算法 [J]. 计算机应用与软件, 2015 (10): 1-4.

[14] 赵焱德. 基于 SPARK 的海量数据频繁模式挖掘算法研究 [D]. 哈尔滨: 哈尔滨工业大学, 2016.

[15] 郭进伟, 皮建勇. 基于 MapReduce 的 SON 算法实现 [J]. 计算机应用, 2014 (a01): 100-102.

[16] 马可, 李玲娟, 孙杜靖. 分布式并行化数据流频繁模式挖掘算法 [J]. 计算机技术与发展, 2016 (7): 75-79.

[17] 唐颖峰, 陈世平. 一种基于后缀项表的并行闭频繁项集挖掘算法 [J]. 计算机应用研究, 2014, 31 (2): 373-377.

Note: Figure translations are in progress. See original paper for figures.