DF Eclat: An Eclat Algorithm Based On Deep Pruning Optimization

Peng Xiao

***Abstract*—The Eclat algorithm is one of the most widely used methods for mining frequent itemsets. In the ordinary Eclat algorithm and its variants, calculating the intersection size of project sets through sequential comparison of elements is inefficient, especially for large-scale transactions. We propose a DF Eclat algorithm based on deep pruning optimization and Flink distributed computing framework. This algorithm overcomes the problems of poor adaptability to big data, weak parallel computing ability, and the generation of a large number of invalid candidate itemsets during the iteration process when calculating frequent itemsets, wasting time and memory. This article introduces the idea of DF Eclat algorithm and studies its performance. The experimental results show that this algorithm effectively solves the problems existing in Eclat algorithm and has good acceleration performance.**

***Index Terms*—Deep pruning optimization, Flink framework, Frequent itemsets, Eclat algorithm**

I. Introduction

The mining of frequent itemsets is a fundamental problem in many data mining applications [1].

Algorithms for mining frequent itemsets can be basically divided into two categories: one is algorithms based on horizontally arranged datasets,For example, the Apriori [2] algorithm and the FP Growth [3] algorithm;Another algorithm is based on vertical layout of datasets, such as Eclat [4]. Eclat has more advantages than algorithms based on horizontal layout databases. It saves a lot of time because it does not require scanning the entire database [4-6].

With the exponential growth of data volume, traditional serial Eclat algorithms are no longer sustainable, with problems such as low execution efficiency, high resource consumption, low scalability, and poor fault tolerance [6]. The emergence of distributed parallel computing has effectively solved the above problems and become the best choice for processing massive data.

Faced with the problem of 'big data' [7], it is feasible and practical to use parallel or distributed algorithms to mine frequent itemsets from massive data. The current proposed parallel algorithms include CD (Count Distribution), CaD (Candidate Distribution), and DD (Data Distribution) [8]. These algorithms are all based on Apriori and have significant limitations in communication and synchronization [9]. However, these algorithms have certain guiding significance for parallel frequent itemset mining. Zaki [10] proposed four parallel algorithms based on DEC memory channel technology, but due to the inability to put a large amount of data into shared memory, they are not suitable for mining massive datasets. Based on this, this article proposes the DF Eclat algorithm, and experimental results show that this algorithm effectively solves the aforementioned problems and has good acceleration performance.

The rest of this article is organized as follows: Section 2 introduces background knowledge. Section 3 discusses the design concept of the DF Eclat algorithm. We presented our experimental results in Section 4. The fifth part summarizes this article.

Manuscript received December 13, 2023

Peng Xiao, School of Computer Science and Technology, TianGong University, TianJin, China)

II. Background Knowledge

A. Dataset Style

The traditional horizontal data format consists of transactions consisting of transaction identifiers (Tid) and items (items). Transactions are uniquely identified by Tid and can contain one or more items. The Eclat algorithm incorporates the idea of inversion, using a vertical data format to represent data, where a record consists of a list of one item and all transaction records (Tidset table) [11]. The specific data format types are shown in Table 1, with horizontal data structures used by algorithms such as Apriori and FP-Growth on the left and vertical data structures used by Eclat on the right.

Table 1 Two Dataset Distribution Formats

Horizontal dataset		Vertically dataset	
Tid	Item	Item	Tid
T1	A;B;D	A	T1
T2	B;C;E	B	T1;T2;T3
T3	B	C	T2;T4;T5
T4	C;D;E	D	T1;T4
T5	C;E	E	T2;T4;T5

B. Frequent Itemsets

Let I={i1, i2,..., im} it is a set. Let D={t1, t2... tm} be a transaction database, where each transaction has a unique

identifier (TID) and contains a set of collections. A set of k items is called a k-item set. Given a matter set X and a matter ⊤ , Make $X \subseteq$ ⊤ , be ⊤ Contains X. We call the support level (X) | X |/| D | the support level of project set X; Here, | X | represents the number of items containing project set X, and | D | represents the total number of transactions. When and only when the support of itemset X is greater than the minimum support (minSup) values specified by the user [12], [13], itemset X is considered frequent.

C. *Eclat Algorithm*

The Eclat algorithm is based on a vertical data format. It only needs to scan the dataset once, and then it can enumerate all frequent itemsets through simple set intersections.

Eclat utilizes the theory of equivalence classes and iteratively calculates the candidate (k+1) itemset by combining two frequent k-itemsets. The Tidset table of the candidate (k+1) itemset is obtained by intersecting the Tidset tables of the two frequent k-itemsets. By calculating the support of the candidate (k+1) itemset Tidset table, it is determined to generate frequent (k+1) itemsets, iterate until union or intersection is empty, and the algorithm ends [14]. The logical code is:

```
Eclat (F_k) {
   for all x_i∈ F_k
      F_{k+1} = Ø
      for all x_i∈ F_k i < j
         C_{k+1} = x_i ∩ x_j
      if sup ( x_i ∩ x_j ) >min_sup
         F_{k+1} =F_{k+1}∪ C_{k+1}
   end
end
if (F_{k+1}≠ Ø)
   Eclat(F_{k+1})
}
```

III. Implementation of DF Eclat Algorithm

A. *Deep pruning optimization*

The Eclat algorithm does not require multiple scans of the database during the mining process, effectively reducing I/O consumption. However, unrelated operations can generate a large number of unrelated itemsets, increasing memory and time consumption. Therefore, through Pre-pruning optimization and Post-pruning optimization, the size of the candidate set is reduced and the calculation of irrelevant items is reduced.

a. Deep pruning optimization

The pre pruning idea utilizes the following characteristics: when generating a k-itemset from the union of two (k+1) itemsets, if the first k-2 terms of the two itemsets are different, the new itemset is the previously generated repeated or non frequency itemset [15].

Based on this property, frequent k-term sets are extracted during the algorithm implementation process, with the first k-1 term as the prefix, and the data is distributed to different computing nodes according to different prefixes. The mining process only requires each computing node to jointly generate a candidate (k+1) itemset internally, effectively reducing the size of the candidate set and reducing the network load of data exchange between nodes in big data environments.

b. Post-pruning optimization

In order to further reduce the size of the candidate set, post pruning optimization was performed on Ck. The pruning idea uses the downward closure attribute of frequent itemset prior constraints. It is a non frequent itemset, so any (k+1) term of x is also a non frequent itemset.

Based on this property, in the algorithm implementation, we sort the dataset of each computing node in ascending support order, and then use a double-layer loop to traverse the dataset. Among them, the outer loop Lj traverses the dataset from low to high, using attributes to delete infrequent itemsets and compress the dataset size in real time. The inner loop Li (i>j, where i, j are loop variables) traverses the dataset from high to low, and the union of the outer loop results generates a high-order candidate set. The properties are used again to delete infrequent itemsets and further compress the dataset size.

B. *Flink framework*

The Flink platform regards batch processing as an extreme case of stream processing and uses the concept of stream processing to solve batch processing problems. In other words, the Flink platform treats all tasks as streams. The parallelism of stream processing is easy to implement, and the Flink platform has a concept of parallelism, which can achieve the goal of parallel execution of programs, tasks, and threads by setting the parallelism value to be greater than 1.

A Flink program consists of multiple tasks, each of which is executed by multiple parallel threads. The number of threads executing a task in parallel is called the parallelism of the task. The parallelism setting on the Flink platform can be specified at multiple levels, including the operator, execution environment, client, and system.

As shown in Figure 1, there are four operators in the current data stream: source, map, window, and sink. The parallelism of the sink operator is 1, while the parallelism of the other operators is 2. So the parallelism of this stream processing program is 2.

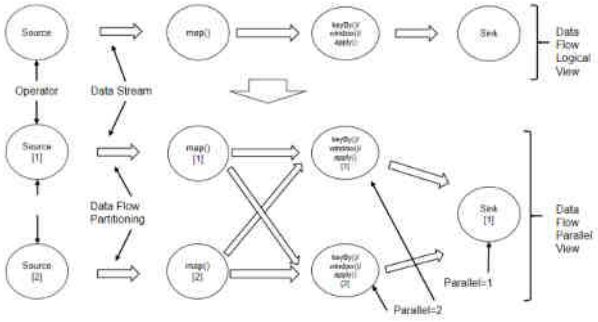

Figure 1.Parallel Subtasks and Parallelism of Flink Framework

IV. Performance Evaluation

To verify the effectiveness and feasibility of the improved

algorithm, experiments were conducted in a cluster environment consisting of four nodes (1 master node, 3 slave nodes) on a laptop computer with 8GB of memory, AMD Ryzen 7 5800H processor with Radeon Graphics 3.20 GHz, and 1T hard drive. The master node is configured with 2GB of memory and 100G of hard disk; The configuration of the three slave nodes is the same, with 1GB of memory and 50GB of hard disk. The operating system is Ubuntu 20.04, JDK version is JDK1.8, Hadoop version is Hadoop 2.6.4, and Flink version is Flink 1.17. All algorithms are implemented using the Scala language. The dataset used in the experiment is shown in Table 2:

Table 2 Dataset Introduction

Dataset name	Number of transactions/piece	Average physical length/piece	Dataset size/kb
Mushroom	8124	23	558

By setting different minimum support thresholds, a line comparison chart of the running time between the original Eclat algorithm and the DF Eclat algorithm proposed in this article was obtained under the dataset in Table 2. The x-axis represents the minimum support threshold, and the y-axis represents the running consumption time. The results are shown in Figure 2.

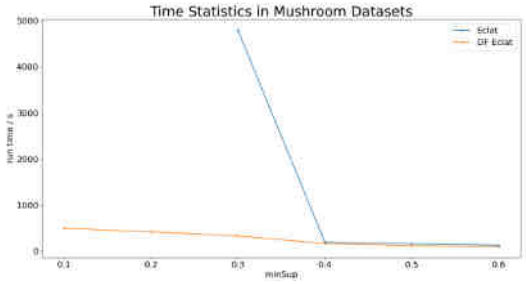

Figure 2.Time Statistics in Mushroom Datasets

Figure 2 shows a comparison of the runtime of two algorithms on a dense dataset called Mushroom. It can be seen from the graph that when the minimum support threshold value is smaller, both algorithms require longer runtime. Among them, the Eclat algorithm may experience memory overflow when the minimum support threshold value is less than 0.3, resulting in the program not running properly. The improved algorithm proposed in this article has a smaller minimum support threshold and higher efficiency in terms of runtime when facing this dataset.

By setting different minimum support thresholds, the memory consumption histogram of the original Eclat algorithm and DF Eclat algorithm under the dataset in Table 2 is obtained, and the results are shown in Figure 3.

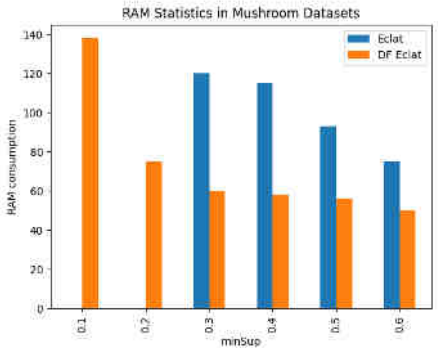

Figure 3.RAM Statistics in Mushroom Datasets

Figure 3 shows a comparison of memory consumption between two algorithms on the dense dataset Mushroom. It can be seen from the figure that for both algorithms, the smaller the minimum support threshold, the larger the required computer memory. As the minimum support threshold increases, the required computer memory decreases. The DF Eclat algorithm proposed in this article outperforms the original Eclat algorithm in memory consumption, and the smaller the minimum support threshold, the better the performance of the DF Eclat algorithm in memory consumption.

V. Conclusion

This article proposes a parallel algorithm DF Eclat based on deep optimization pruning and Flink framework. DF Eclat first transforms transactional datasets, dividing frequent 1-item sets into balanced groups, and then reassigns the records of the transformed dataset based on their prefixes. Records with prefixes in the same group will be distributed to the same computing node. DF Eclat uses an improved Eclat to process data with the same prefix. The experimental results indicate that DF Eclat has high scalability and good acce leration ratio.

References

[1] R. Agrawal, T. Imieliski,and A. Swami, " Mining association rules between sets of items in large databases," Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM, Vol. 22. No. 2, 1993, pp. 207-216.

[2] R. Agrawal, R. Srikant, " Fast algorithms for mining association rules, " Proceedings of the 20th International Conference on Very Large Data Bases, VLDB. Vol. 1215, 1994, pp. 487-499.

[3] J. Han, J. Pei, and Y. Yin, " Mining frequent patterns without candidate generation," ACM SIGMOD Record. ACM, Vol. 29, No. 2, 2000, pp. 1-12.

[4] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, " New algorithms for fast discovery of association rules," Proceedings of the 3th International Conference Knowledge Discovery and Data Mining,vol.97, 1997, pp. 283-286.

[5] Z. F. Li, X. F. Liu, andX. Cao. " A study on improved Eclat data mining algorithm," Advanced Materials Research, vol. 328, 2011. pp. 1896-1899.

[6] B. Kotiyal, A. Kumar, B. Pant, R. H. Goudar, S. Chauhan, and S. Junee, " User behavior analysis in web log through comparative study of Eclat and Apriori," Proceedings of 7th International Conference on Intelligent Systems and Control (ISCO). IEEE, pp. 421-426, 2013.

[7] D. Howe, M. Costanzo, P. Fey, T. Gojobori, L. Hannick, and W. Hide, et al., " Big data: The future of biocuration" , Nature, Vol. 455, No. 7209, 2008, pp. 47-50.

[8] M. Ashrafi, T. Zaman, S. David, and S. Kate, " ODAM: an optimized distributed association rule mining algorithm," Distributed Systems Online 1541-4922, IEEE, Vol. 5, No. 3, 2004, pp. 1-18.

[9] X. Y. Yang, Z. Liu, and Y. Fu, " MapReduce as a programming model for association rules algorithm on Hadoop," Proceedings of 3rd International Conference on Information Sciences and Interaction Sciences (ICIS). IEEE, pp. 99-102, 2010.

[10] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, " Parallel algorithms for discovery of association rules," Data Mining and Knowledge Discovery, Vol. 1, No. 4, 1997, pp. 343-373.

[11] J. Yang, Y. Zhang, Y. Wei, " An Improved Vertical Algorithm forFrequent Itemset Mining from Uncertain Database, " InternationalConference on Intelligent Human-Machine Systems and Cybernetics.IEEE, Aug.2017, pp.355-358, doi: 10.1109/IHMSC.2017.87.

[12] S. Aggarwal and V. Singal, " A survey on frequent pattern mining algorithms," Int. J. Eng. Res. Technol., 2014, pp. 86– 89.

[13] C. C. Aggarwal and J. Han, Frequent Pattern Mining. New York, NY, USA: Springer, 2014, pp. 1– 17.