An Algorithm to Optimize Frequent Pattern Mining in Parallel and Distributed Environment

Anshu Singla
Faculty of Computer Applications, Manav Rachna International Institute of Research and Studies (MRIIRS), Faridabad, Haryana, India | KCCITM, Greater Noida, India
singlaanshu2016@gmail.com (corresponding author)

Parul Gandhi
Faculty of Computer Applications, Manav Rachna International Institute of Research and Studies (MRIIRS), Faridabad, Haryana, India
parul.sca@mriu.edu.in

Received: 4 December 2024 | Revised: 25 December 2024, 14 January 2025, 1 February 2025, 3 February 2025 | Accepted: 5 February 2025

 DOI: https://doi.org/10.48084/etasr.9830

ABSTRACT

Frequent Pattern Mining (FPM) is an important data mining task that involves identifying recurrent patterns or correlations in datasets. The main purpose of FPM algorithms is to find sets of items that frequently appear in transactional or relational databases. This study presents a Parallel and Distributed Recursive Elimination (PDReLim) algorithm, a novel FPM technique designed for parallel computing to improve efficiency compared to existing parallel FPM algorithms. PDReLim recursively deletes infrequent items on each node while using the capabilities of parallel and distributed systems or clusters. Its performance was evaluated on well-known datasets, namely Chess, Mushroom, and Connect, available in the UCI repository, with a focus on the lowest support threshold, which causes computational bottlenecks for many FPM algorithms. PDReLim, implemented in PySpark, outperforms standard MapReduce for iterative algorithms. Spark's execution is optimized for large databases by utilizing its proficient capabilities, such as the RDD data structure, in-memory processing, and shared variables. The results show that PDReLim was significantly faster than PApriori, PFP-Growth, and PFP-Max.

Keywords-PySpark; Frequent Pattern Mining (FPM); parallel FPM; Spark; association rule mining; apriori; eclat

I. INTRODUCTION

Basic association rules are a serial strategy that was utilized by mining algorithms but is inefficient for larger datasets. As databases expand in size, their accuracy starts to decline slowly. As a result, parallel algorithms came into play to manage massive datasets. Various cluster-based techniques are available to manage large datasets, but they introduce several issues such as redundant data, synchronization, etc. As a result, the parallel approach was replaced by the MapReduce approach. Apriori and FP-Growth are the best algorithms for finding frequent itemsets. The MapReduce approach is the best platform for implementing the Apriori algorithm. A Resilient Distributed Dataset (RDD) in Spark aids in the resolution of such problems. The key benefit of using these attributes is that the outcome of the iteration is preserved in a local cache and is available for subsequent iterations. Spark focuses on analytics and large-scale data processing. The capacity of Apache Spark to split data processing activities across a network of computers allows for scalable and parallel processing of large datasets, using different high-level APIs such as Python, R, Scala, and Java. PySpark is the Python programming interface for Apache Spark. RDD is the core data structure in Spark that denotes dispersed datasets that can be handled concurrently.

A. Research Problem and Motivation

Parallel frequent mining algorithms, such as PApriori and PFP-Growth, lack scalability and cannot use the capability of parallel processing and distributed systems of PySpark. Existing algorithms are inefficient due to inappropriate pruning strategies. In [1], a systematic review of parallel and distributed association rule mining algorithms was presented. This study provides a guide for researchers and suggests interesting research directions. In [2], different parallel and distributed algorithms were surveyed on various hardware platforms, focusing on the scalability of the algorithms to handle massive datasets. The DHP algorithm extends the Apriori approach by using a hash table to precompute approximate support of two itemsets. In [3], a comprehensive survey of the Parallel Association Rule Mining (PARM) and Distributed Association Rule Mining (DARM) approaches was presented, providing a systematic and rigorous evaluation of significant developments in the field. In [4], different FPM algorithms were analyzed and tested on four real-life datasets: Retail, Accidents, Chess, and

Mushrooms. The evaluation was based on execution time and memory consumption, and Pre-post+ outperformed the others on the Mushrooms dataset. Big data analytics is important for extracting knowledge from large datasets [5]. Frequent Itemset Mining (FIM) is a key data mining technique. However, FIM algorithms, such as Apriori and Eclat, suffer from efficiency issues with big data. SHFIM [5] is a hybrid Spark-based algorithm for FIM, utilizing both horizontal and vertical layouts and diffsets for efficiency, while the experimental results showed that it outperformed Eclat and dEclat in execution time. In [2], the use of Association Rule Mining (ARM) in data mining was discussed. ARM can be used to find subsets of items that frequently occur together. This study surveyed different parallel and distributed ARM algorithms proposed on various hardware platforms. In [6], the focus was on ARM and FIM, reviewing existing algorithms and proposing new efficient algorithms using Spark. The aim was to handle large amounts of data and compare algorithms in terms of speed, efficiency, and memory consumption.

In [7], the focus was on the parallel mining of association rules in streaming data, as traditional data mining algorithms are not suitable for large streaming data. This study proposed the SSPFP algorithm using Spark Streaming for real-time mining to mine association rules in marine Argo data. In [8], a parallel implementation of data mining techniques for FPM was presented, comparing the performance of parallel, concurrent, and serial implementations and highlighting the advantages of FPM. In [9], an improved version of the Apriori algorithm was presented, which offered superior performance compared to current methods on a spark framework with the aid of PySpark. In [10], three datasets were used to compare different approaches for log file pattern mining using the Apache Spark platform. A new algorithm was also proposed, based on Apriori, to optimize the extraction of association rules from databases. The proposed algorithm outperformed Eclat and LCM Freq in terms of speed and memory consumption, saving more than half of the memory on the Chess database. In [11], the Orange tool surpassed KNIME in effectively identifying cyberbullying cases in Arabic comments on Instagram. In [12], a novel approach was presented to optimize the extraction of association rules, showing notable gains in computing efficiency, significantly reducing the number of created itemsets and the overall execution time compared to more conventional techniques such as Apriori. In [13], a thorough analysis of Privacy-Preserving Data Mining (PPDM) methods was presented, highlighting how crucial they are to protecting private information while allowing insightful analysis of large datasets. The computational complexity of identifying maximally frequent itemsets can be high, especially when dealing with large or complexly patterned datasets. The time and memory requirements to mine these itemsets can become expensive as the dataset grows larger and more complex [14].

II. PROPOSED SYSTEM

In association rule mining, the goal is to find interesting relationships or associations between items in a dataset. The Recursive Elimination (ReLim) algorithm is designed to address some of the performance issues associated with Apriori using a recursive approach to eliminate infrequent itemsets efficiently. The basic idea is to identify the most frequent itemsets first and then recursively eliminate infrequent itemsets to reduce the search space. The association rule mining method in ReLim starts by initializing the full transaction database. It scans and counts the number of occurrences to identify frequent 1-itemsets. Using them, it creates candidate 2-itemsets, detects frequent 2-itemsets, and recursively eliminates infrequent itemsets while producing higher-order candidates. This technique is repeated iteratively until no more frequent item sets can be found. The key idea behind ReLim is to recursively eliminate infrequent itemsets at each step, reducing the search space and improving efficiency compared to other algorithms such as Apriori. The algorithm takes advantage of the fact that if a set of items is infrequent, any superset containing that set is also infrequent.

A. Proposed PDReLim Algorithm

Implementing the ReLim algorithm in parallel using PySpark is a good strategy to handle large datasets more efficiently. PySpark is a Python library for Apache Spark, a distributed computing framework that allows for parallel and distributed processing. The parallel implementation of the ReLim algorithm follows data preparation and data distribution. Parallelizing the ReLim algorithm involves distributing the workload across multiple processors or nodes to enhance the efficiency of frequent itemset mining on large datasets. The basic steps for the PDReLim are as follows.

1) Data Distribution

Divide the transaction dataset into smaller segments. In a parallel computing environment, each partition should be processed independently by individual processors or nodes. In PySpark, for example, this can be accomplished by repartitioning the DataFrame.

2) Parallel Frequent 1-Itemset Mining

Assign frequent 1-itemsets to different divisions for computation. Each partition counts the number of 1-itemsets that appear locally. Identify frequent 1-itemsets globally by combining local results.

3) Parallel Candidate Generation

Use the frequent 1-itemsets to generate candidate 2-itemsets simultaneously. Each division creates candidates locally, which are subsequently combined globally. Repeat the process for higher-order itemsets, generating (k+1)-itemsets from common k-itemsets.

4) Parallel Counting and Recursive Elimination:

Divide the counting and recursive elimination steps into partitions. Each partition counts the occurrences of candidate itemsets locally and discards infrequent itemsets. Combine the results to determine globally frequent itemsets.

5) Iterative Processing

Repeat iteratively until no more frequent itemsets can be formed or a predetermined maximum iteration is obtained.

6) Association Rule Generation

Generate association rules in parallel using frequent itemsets.

7) Finalize and Aggregate Results

Combine and finalize findings from many partitions to get a comprehensive set of frequent items.

8) Optimizations

Optimize performance through data caching, and use efficiently broadcast variables and appropriate partitioning schemes.

The steps for implementing the PDReLim algorithm using PySpark are as follows:

- Step 1: Load the transaction dataset into a PySpark DataFrame.
- Step 2: Preprocess the data as required for the ReLim method, ensuring that it is spread among Spark jobs.
- Step 3: Repartition the DataFrame to divide the data among Spark partitions. This ensures that each division can be processed independently by many workers.
- Step 4: Define the logic of the ReLim algorithm using PySpark transformations and actions. This includes filtering, grouping, and aggregating data to find common itemsets.
- Step 5: Use PySpark's parallel processing capabilities. Many PySpark transformations are automatically parallelized.

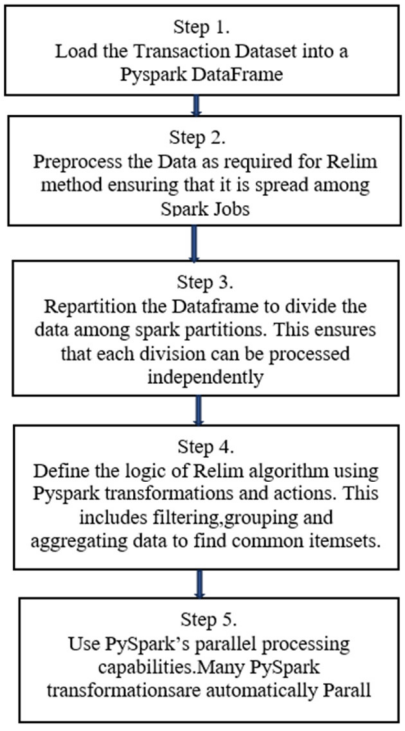

Fig. 1. Flowchart of the PDReLim algorithm.

III. RESULTS AND DISCUSSION

The results were based on the support value provided by the user during the execution. The procedure was carried out on a single workstation using three datasets from the UCI dataset repository, namely Mushroom [15], Connect [16], and Chess [17]. The performance of the PDReLim algorithm was evaluated based on execution times.

TABLE I. DATASET CHARACTERISTICS

Dataset	Items	Average length	Transactions	Size
Chess	75	35	3198	334K
Connect	127	36	1352	15.9M
Mushroom	119	21	8122	557K

A. Experimental Setup

A workstation equipped with an Intel i7 processor, 16 GB RAM, and a 2 TB SATA 2 hard drive served as the basis for the implementation configuration. The experiments were coded on Anaconda3-2023.09-0-Windows-x86_64 with Jupyter and Spark 3.4.2 with Python on Windows 10. The algorithms were tested on a single workstation.

B. Results on Mushroom Dataset

The Mushroom classification dataset [15] is a well-known benchmark. Table II and Figure 2 show the performance of the PDReLim algorithm using different minimum support and threshold values on this dataset. The support value is used for the frequency threshold and to discard infrequent itemsets. The results of the experimental execution time show that PDReLim was more efficient than PAriori, PFP-Growth, PFP-max, and Peclat. On average, the PDReLim algorithm was approximately six times faster than PApriori, three times faster than PFP-Growth, and 1.5 times faster than PFP-max.

TABLE II. EXECUTION TIME (MS) ON THE MUSHROOM DATASET

Support	PApriori	PFP-Growth	PFP-max	Peclat	PDReLim
10	4838	2278	1126	1089	846
20	648	372	325	440	325
40	325	190	140	120	114
60	282	120	113	106	97
80	240	102	92	84	68

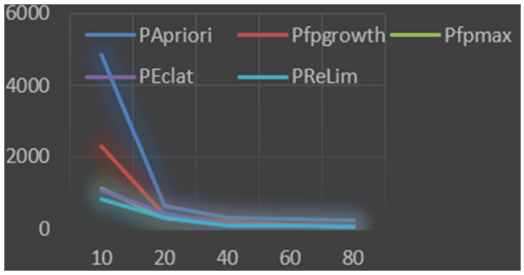

Fig. 2. Performance comparison on the Mushroom dataset.

C. Results on Connect Dataset

Each game state from the Connect-4 board game in the Connect dataset [16] corresponds to a particular board configuration. The aim variable specifies the game outcome (victory, loss, or draw) for the first player, while the

characteristics define the positions of the pieces on the board. Figure 3 and Table III show the execution times of PApriori, PFP-Growth, Peclat, PFP-max, and the proposed PDReLim on the Connect dataset. A high minimum support criterion was used since all algorithms ran out of memory when the minimum support was less than 95. The results show that PDReLim was faster than the other algorithms, being approximately five times faster than PApriori, four times faster than PFP-Growth, and two times faster than PFP-max on average.

TABLE III. EXECUTION TIME (MS) ON CONNECT DATASET

Support	PApriori	PFP-Growth	PFP-max	Peclat	PDReLim
95	6743	4318	2245	1552	1224
96	2131	1342	2048	1280	1143
97	1235	1022	984	1249	857
98	802	726	553	629	219
99	426	349	248	223	128

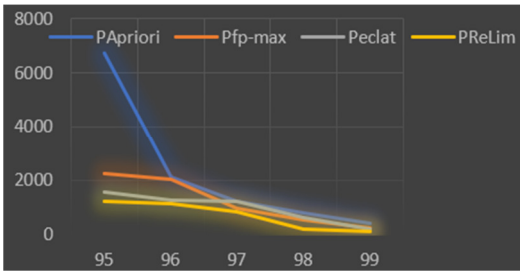

Fig. 3. Performance comparison on the Connect dataset.

D. Results on Chess Dataset

The Chess dataset [17] includes several chess-related datasets, including the King-Rook vs. King-Pawn and Chess Endgame datasets, recording board configurations and potential outcomes according to game rules. This dataset contains chess-related information, such as turns, results, or approaches, and it is useful for understanding chess-playing principles and techniques. Figure 4 and Table IV show the performance of PDReLim, which performed approximately four times better than PApriori, three times faster than PFP-Growth, and two times faster than PFP-max on average.

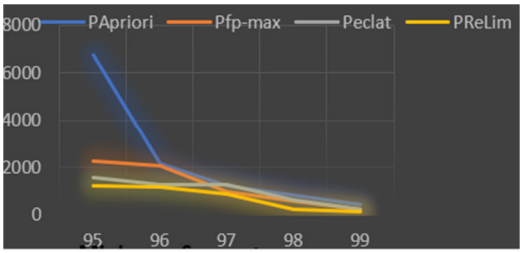

Fig. 4. Performance comparison on the Chess dataset.

TABLE IV. EXECUTION TIME (MS) ON CHESS DATASET

Support	PApriori	Pfp-growth	Pfp-max	Peclat	PDReLim
10	3245	2243	1549	1027	824
20	2732	895	547	478	274
40	1352	545	575	358	148
60	648	324	286	346	112
80	232	210	154	139	76

IV. CONCLUSION

This study introduces PDReLim, a novel FPM algorithm designed within the parallel computing framework to enhance efficiency by using the parallelization scheme of FPM algorithms. This method focuses on the simultaneous deletion of infrequent itemsets at each node, acquiring the advantage of the power of distributed systems or clusters with a large number of nodes. PDReLim was evaluated in terms of execution speed and accuracy using the frequently used datasets Chess, Mushroom, and Connect from the UCI repository [15-17]. On the Musroom dataset, PDReLim was around six times faster than PApriori, three times faster than PFP-Growth, and 1.5 times faster than Pfpmax. On the Connect dataset, PDReLim was about five times faster than PApriori, four times faster than Pfpgrowth, and two times faster than PFP-max. PDReLim outperformed PApriori, PFP-Growth, and PFP-max approximately four, three, and two times, respectively. The implementation of PDReLim in PySpark demonstrated higher performance for iterative algorithms. The execution for large datasets was optimized using Spark's advanced capabilities, such as the RDD data structure, in-memory processing, and shared variables. These results demonstrate that PDReLim improves both speed and accuracy, making it a potential solution for PFP mining. Compared to existing parallel FPM techniques, PDReLim is novel because of its highly effective parallelization and unique pruning method for infrequent itemsets. PDReLim redefines the state-of-the-art in parallel FPM by addressing major computing constraints and providing a scalable and efficient solution for large-scale data analysis. The next phase seeks to take advantage of GPUs' parallel processing capabilities, which could unleash even bigger performance advantages.

REFERENCES

[1] P. Gupta and V. Sawant, "A Parallel Apriori Algorithm and FP- Growth Based on SPARK," *ITM Web of Conferences*, vol. 40, 2021, Art. no. 03046, https://doi.org/10.1051/itmconf/20214003046.

[2] M. J. Zaki, "Parallel and distributed association mining: a survey," *IEEE Concurrency*, vol. 7, no. 4, Oct. 1999, Art. no. 14–25, https://doi.org/10.1109/4434.806975.

[3] S. Biswas, N. Biswas, and K. C. Mondal, "Parallel and Distributed Association Mining: A Recent Survey," *Information Management and Computer Science*, vol. 2, no. 1, pp. 15–24, Sep. 2019, https://doi.org/10.26480/imcs.01.2019.15.24.

[4] R. Khajuria, A. Sharma, S. Sharma, A. Sharma, J. Narayan Baliya, and P. Singh, "Performance analysis of frequent pattern mining algorithm on different real-life dataset," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 29, no. 3, Mar. 2023, Art. no. 1355, https://doi.org/10.11591/ijeecs.v29.i3.pp1355-1363.

[5] M. R. Al-Bana, M. S. Farhan, and N. A. Othman, "An Efficient Spark-Based Hybrid Frequent Itemset Mining Algorithm for Big Data," *Data*, vol. 7, no. 1, Jan. 2022, Art. no. 11, https://doi.org/10.3390/data7010011.

[6] C. Fernandez-Basso, M. D. Ruiz, and M. J. Martin-Bautista, "New Spark solutions for distributed frequent itemset and association rule mining algorithms," *Cluster Computing*, vol. 27, no. 2, pp. 1217–1234, Apr. 2024, https://doi.org/10.1007/s10586-023-04014-w.

[7] L. Liu, J. Wen, Z. Zheng, and H. Su, "An improved approach for mining association rules in parallel using Spark Streaming," *International Journal of Circuit Theory and Applications*, vol. 49, no. 4, pp. 1028–1039, Apr. 2021, https://doi.org/10.1002/cta.2935.

[8] J. J. Flores *et al.*, "Parallel mining of frequent patterns for school records analytics at the Universidad Michoacana," in *2017 IEEE International*

Autumn Meeting on Power, Electronics and Computing (ROPEC), Ixtapa, Nov. 2017, pp. 1–6, https://doi.org/10.1109/ROPEC.2017.8261636.

[9] F. Gao, C. Bhowmick, and J. Liu, "Performance Analysis Using Apriori Algorithm Along with Spark and Python," in *Proceedings of the 2018 International Conference on Computing and Big Data*, Charleston, SC, USA, Sep. 2018, pp. 28–31, https://doi.org/10.1145/3277104.3277108.

[10] A. Satty, M. M. Y. Salih, A. A. Hassaballa, E. A. E. Gumma, A. Abdallah, and G. S. Mohamed Khamis, "Comparative Analysis of Machine Learning Algorithms for Investigating Myocardial Infarction Complications," *Engineering, Technology & Applied Science Research*, vol. 14, no. 1, pp. 12775–12779, Feb. 2024, https://doi.org/10.48084/etasr.6691.

[11] S. S. Alzahrani, "Data Mining Regarding Cyberbullying in the Arabic Language on Instagram Using KNIME and Orange Tools," *Engineering, Technology & Applied Science Research*, vol. 12, no. 5, pp. 9364–9371, Oct. 2022, https://doi.org/10.48084/etasr.5184.

[12] B. Bouaita, A. Beghriche, A. Kout, and A. Moussaoui, "A New Approach for Optimizing the Extraction of Association Rules," *Engineering, Technology & Applied Science Research*, vol. 13, no. 2, pp. 10496–10500, Apr. 2023, https://doi.org/10.48084/etasr.5722.

[13] D. J. I. Raj, V. S. Radhakrishnan, M. R. Reddy, N. S. Selvan, B. Elangovan, and M. Ganesan, "The Projection-Based Data Transformation Approach for Privacy Preservation in Data Mining," *Engineering, Technology & Applied Science Research*, vol. 14, no. 4, pp. 15969–15974, Aug. 2024, https://doi.org/10.48084/etasr.7969.

[14] M. Sinthuja, S. Pravinthraja, B. K. Dhanalakshmi, H. L. Gururaj, V. Ravi, and G. Jyothish Lal, "An efficient and resilience linear prefix approach for mining maximal frequent itemset using clustering," *Journal of Safety Science and Resilience*, vol. 6, no. 1, pp. 93–104, Mar. 2025, https://doi.org/10.1016/j.jnlssr.2024.08.001.

[15] "Mushroom." UCI Machine Learning Repository, 1981, https://doi.org/10.24432/C5959T.

[16] J. Tromp, "Connect-4." UCI Machine Learning Repository, 1995, https://doi.org/10.24432/C59P43.

[17] R. Quinlan, "Chess (King-Rook vs. King-Knight)." UCI Machine Learning Repository, 1983, https://doi.org/10.24432/C56W2G.