Grenze International Journal of Engineering and Technology, January Issue

Enhancing Frequent Pattern Mining in Big Data using Optimized Apriori Algorithm with Dynamic MapReduce and Pruning Techniques in Apache Spark Environment

S Usha Manjari[1], Vikrant Sabnis[2] and Jay Kumar Jain[3]
[1]Research Scholar, Mansarovar Global University, Bhopal
Email: manjariusha@gmail.com
[2]Professor, Department of Computer Science, Faculty of Engineering and Technology Mansarovar Global University, Bhopal
Email: drvikrantsabnis@gmail.com
[3]Assistant Professor, Department of Mathematics Bioinformatics and Computer Applications, Maulana Azad National Institute of Technology, Bhopal, Madhya Pradesh, India
Email: jayjain.research@gmail.com

Abstract*—This research focuses on improving the Apriori algorithm for large frequent pattern mining in map reduce big data environment using dynamic map reduce and the pruning techniques on Apache spark. One of the most important algorithms, which cannot be considered a data mining tool without, is the Apriori algorithm used for the identification of frequent item sets and generation of association rules. However, it increases considerably with the number of data that means for large datasets there must be efficient ways of handling these data. The first step of the work is to investigate the influence of changing the number of mappers and reducers in the framework of the offered Dynamic MapReduce Apriori algorithm. In the view of the above variation, the best of the best configuration is determined to be the one that uses 3 mappers and 2 reducers and the corresponding result has an execution time of 83sec. 20 seconds. This supports the need to consider the strategic configuration of Apriori algorithm's by adjusting expected outcomes from the algorithm. Then, the study incorporates the integration of sophisticated pruning procedures into the Apache Spark milieu to optimize the performance of frequent pattern discovery. With the help of further experiments using similar mapper and reducer con Figurations, it was found out that there were substantial improvements in the setup executed times in all scenarios. Interestingly, when Hadoop MapReduce model is fine-tuned with pruning strategies and Hadoop Map Reduce 3 mappers with 2 reducers model, the execution time is found to be 43. 20 seconds. This significant improvement clearly illustrates the possible positive impacts of pruning in reducing the necessary computation and in improving the algorithms' performance. The discoveries underscore the possibility of hybridizing Dynamic MapReduce with pruning methods in Apache Spark for enhancing the Apriori algorithm in front ranking in large scale pattern mining on big data. The work provides important findings on how to efficiently perform operations on massive characteristics and frequent itemsets, thus providing solutions for the problem being investigated.

Grenze ID: 01.GIJET.12.1.163_1

The findings presented in this paper have important implications for practitioners and academics engaged in the design and utilization of distributed computing platforms for large-scale data analysis, especially for the patterns mining in big data environments.

***Keywords*— Apriori Algorithm, Dynamic MapReduce, Apache Spark, Data Mining, Frequent Pattern Mining, Big Data, Pruning Techniques, Execution Time, Mappers, Reducers.**

I. INTRODUCTION

Due to massive data production trends across different disciplines, there is a dire need to employ effective data analysis methodologies that can analyse and mine great amounts of data. Association rule learning,[1], or market basket analysis, is one of the major goals of data mining and implies the discovery of logical relationships in sales transactions. Out of all the algorithms designed to serve this objective, the most used algorithm is the Apriori algorithm[2], which can find out the frequent item set and known association rules. Nevertheless, the Apriori algorithm is a highly useful tool for finding associations, which is especially valuable when used with big data sets; however, it has one significant drawback: it is very time-consuming. This computational demand is quite problematic in the sense that it is highly demanding in terms of the execution time necessary as well as the resource consumption necessary. Apache Spark[3] an open-source distributed computing system, enhances power for processing the large-scale data tasks. Spark operates under the MapReduce model of computation allowing distributing processes within a cluster so as to boost speed as well as scalability. Nonetheless, the exact performance associated with distributed algorithms on Spark greatly depends upon optimal utilization of the underlying compute resources, for example, mappers and reducers[4]. Since these parameters govern the execution and its relation to the stability of Plus, fine-tuning of them is essential for and achieving optimum performance and shortest time to complete the execution.

In this research, the concern is thus on how to enhance the performance of Apriori algo rhythm when implemented in Apache Spark[6]. We must therefore navigate through the complete search space by systematically testing for different numbers of mappers and reducers.[7] Furthermore, to increase effectiveness we also employ pruning to minimize time-consuming calculations. One major technique that aids in achieving this is Pruning which help to exclude certain computation that would not be needed hence saving time in the process. The specific research questions for this study will be addressed are the following: This work is intended to raise understanding about the effects of these optimization techniques on the Apriori algorithm when performed in a distributed environment. The steps used in this type of research as a method of approach include the following steps. To begin, the features in the transactional dataset given are reviewed where missing values of the variables are handled and the data is pre-processed for use with the Apriori algorithm[6]. Subsequently, the Apriori algorithm is applied to data sets in Spark through the MapReduce design to handle the complexity of computations. we then perform an experiment by increasing and decreasing the number of mappers and reducers[7], along with noting down the amount of time taken for the process. Pruning is used on configurations to bypass them if they can't provide better performance and statistics preset[8].

II. LITERATURE REVIEW

The effectiveness of how the data should be processed has therefore been a salient topic of discussion in data mining due to the wide spread use of big data. As for the type of the algorithm. Association rule learning and the Apriori algorithm, in particular, have been researched considerably due to their efficiency in identifying unknown relations and connections in a series of transactions. The current literature explores the enhancement of the Apriori algorithm in its original and developed forms, including the Apriori Property of Big Data; the transition to efficiently executable distributed systems like Apache Spark; and adjustment to pruning that improves on the algorithm's effectiveness.

A. Apriori Algorithm and Its Challenges

The Apriori algorithm defined[9] is among the most famous algorithms implemented in data mining to identify frequent item sets and to create association rules. The discussed algorithm is iterative; after the first pass through the database to find individual elements that possess minimum support, the algorithm then continues to combining data until the item sets with the required support is achieved[10]. However, the Apriori algorithm produces high computational cost particularly due to the exponential increase in the candidate item sets size[10] of the dataset with the increase in the total size of the dataset. Due to this scalability problem, there has been a lot of effort placed in the efforts to improve the efficiency of the above algorithm.

B. Optimization Techniques for Apriori Algorithm

Some of the approaches that have been suggested to address the issues of high computation by the Apriori algorithm include the following:[11] In early days, the optimizations were done for reducing the number of database scans as well as the candidate generations. For instance, the Partition algorithm partitions the discrete database into smaller subsets, processes each subset in isolation[12], and develops the final frequent item sets from the results of these individual subsets. Another approach worth considering is the DIC (Dynamic Itemset Counting) algorithm[13], which makes the counting of item sets as the database is distinguishing itself, thereby eliminating the multiple sweeps through the database. Besides these classical methods, there are many researches in recent years to use the new type of frequent item sets data structures like FP-Trees (Frequent Pattern Trees)[14] which have advantage of having less number of scans compared to classical methods as it stores the entire database in the form of a tree structure. Another approach, called the anti-monotonicity, is done in the FP-Growth algorithm[14], and has been seen to be better than the Apriori algorithm in many cases.

C. Distributed Computing and Apache Spark

The rise of distributed computing paradigms,[15] such as MapReduce, has fundamentally transformed the ability to process massive data sets. In this case, the most well-known one is Apache Spark [3] because of the in-memory processing feature combined with the capability to work with iterative algorithms [15]. MapReduce, an attribute of Spark, also facilitates distribution of problems to the nodes in a cluster, making it highly scalable and thus efficient.

Research done has used Spark to enhance the performance of the Apriori algorithm several times. Such implementations usually imply the partitioning of the dataset across the component nodes of the cluster, as well as the parallel processing of the candidate generation and counting tasks, and the combining of the outcomes to produce the final frequent item sets. In plethora of bench mark experiments carried out on Apriori algorithm, performance deposition[17] on implementation of Spark over the traditional single-node configuration was witnessed.

D. Pruning Techniques

Apart from the above-discussed approaching, different pruning techniques have been proposed and analysed in the literature to improve[18] the Apriori algorithm even more. The major purpose of pruning is to guide the search process by eliminating the useless item sets that are not liable to offer the minimum support level. The most frequent pruning[18] technique employed is based on the Apriori property, which avers, any generated item set has to be frequent, including its subsets. This property facilitates the early filtrate of candidate item sets that do not meet this condition. Subsequently, pruning approaches for maintaining pruned information and strategies for utilizing pruned information and final reduced search space have been discussed in greater detail, especially from the perspective of distributed implementations. For instance, a pruning technique that is used to minimize the support through use of an adaptive minimum support threshold from the properties of the dataset to control the creation of candidate item sets in each pass. Another probabilistic model is probabilistic pruning, which aims at estimating the degree of uncertainty of candidate item sets to be frequent and eliminates those that are considered to have a low probability.

E. Integration of MapReduce and Pruning

Introducing the MapReduce model with pruning methods[20] is an improvement in the Apriori algorithm optimization[20]. Some of the research has indicated that the use of these approaches can cause a massive improvement in execution time and other resource requirements. Nevertheless, the Apriori algorithm is essential in the area of data mining, especially in a discovery of the frequent item sets and association rules in the retail transaction database. Because of the iterative approach of the Apriori algorithm and the requirement of scanning through the entire databases to identify multiple patterns, the algorithm is rather demanding in terms of computation, particularly when the extent of the data pool is great. The program can be fine-tuned for performance in a distributed computing setting like Apache Spark, and much of this can increase its efficacy[21]. Therefore, this literature review is centred on the Apriori algorithm optimisation: with or without pruning in Apache Spark.

Apriori Algorithm Without Pruning in Spark Environment

Traditional Apriori Algorithm

The traditional Apriori algorithm works by creating candidate sets of items and utilizing a repeated[22] database scan to count for these item sets. The algorithm carries out this

process further by enlarging the candidate item-sets in subsequent steps until there are no more frequent item-sets are discovered. This means that it takes an extensive amount of computation time[22] especially when dealing with big data in which 'n' can be extremely high, hence leading to many candidate item sets. Undefined.

MapReduce Paradigm in Spark

Apache Spark, one of the powerful distributed computing frameworks, proves to be quite useful in running iterative algorithms such as Apriori[23]. Spark also used the concept of MapReduce and distribute the computational chores towards the nodes across the cluster making it more scalable and faster. Every node works on individual portion of data and some of the results will be collected together to generate the final set of frequent item sets[23]. It also brings about a distributed approach which reduces the time taken to execute a particular task as opposed to executing it on a single node.

Challenges Without Pruning

Thus, distributed computing itself has its advantages, but for the Apriori scheme still discovered in the Spark environment without pruning, there are many challenges[24]. While the C colleague algorithm could identify item sets quickly and required relatively little time, its primary weakness is the generation and evaluation of vast numbers of candidate bins which contain numerous bins with support levels significantly below the minimum support threshold. It has the effect of incurring a great number of computational operations and long time to complete the job. Scientists have said that while the Spark system has a positive impact on scalability[25], when we come to applying Apriori, the original downside of the algorithm hampers the performance gain on very large sets.

Apriori Algorithm with Pruning in Spark Environment

Pruning Techniques

Retrieval techniques are thought out to restrict and minimize the operation of the Apriori algorithm[26] by eliminating candidate sets a priori that are less possible to be frequent. The most frequent is based on the Apriori property that specifies that all subsets of a frequent itemset including the empty set have to be frequent. This property means that during this stage, candidate item sets that fail this condition can be removed[26], and thus the number of candidate item sets that will need to be passed through later iterations is minimised.

Experimental Research

Various case has demonstrated that applying pruning techniques in Apriori algorithm enhance it repeatedly when in a Spark environment. However, as shown in Figure 8, the pruning[27] that is carried out in a distributed environment specifically minimized run times and resource usage by constraining the possibilities being explored. Again, significant performance[28] improvement has been achieved through the integration of MapReduce-based pruning while stressing on the relationship between distributed computing and various optimization techniques.

The conclusion at the end of the literature also shows that hence the traditional Apriori algorithm is enhanced by the distributed computing as provided by Apache Spark, much performance improvement can be achieved when pruning techniques are incorporated. This is because with growth in the number of parameters, the candidate item sets increase exponentially and so does the amount of computation needed to perform the operation. These happen to be averted by pruning techniques and especially if implemented for a distributed system, since they essentially cut down the search space and shift the computational search in to more interesting item sets. Overall, this enhancement of Spark and pruning techniques is a major enhancement for the Apriori algorithm which aims to enhance the efficiency of data mining processes.

III. RESEARCH OBJECTIVES

From the above Literature Evidence following Research Objectives are Formulated

A. Objective 1: Optimizing Mapper-Reducer Configurations

Description: Explore how different combinations of mappers and reducers impact the performance of the Apriori algorithm with Dynamic MapReduce. By varying these configurations, the goal is to identify the setup that minimizes execution time while maximizing resource utilization.

B. Objective 2: Enhancing Efficiency with Pruning Techniques

Description: Investigate advanced pruning methods within Apache Spark to streamline the frequent pattern mining process. This involves exploring pruning technique and implementing those that effectively reduce computational overhead, thereby improving efficiency and scalability.

IV. RESEARCH IMPLEMENTATION

In more detail, this research addresses the emergence of association rule mining employing the Map-Reduce architecture. Several planar methods proposed in the sequence of the conservation are divided into 8 distinguish phases that plan to systematically tackle the problems of scalability and efficiency in the analysis of large volumes of data.

From here, the process has five steps involved, these include data collection and preprocessing, selection of algorithms, Map-Reduce operations, experimentation, results analysis, comparison with baseline results, discussion, and conclusion. Every phase brings something new into the evaluation and analysis of the Map-Reduce approach for association rule mining enabling insight in its performance, scalability and possible applications.

TABLE I. RESEARCH IMPLEMENTATION FLOW FOR THE OBJECTIVE-1

Phase	Description
Data Collection and Preprocessing	Gather the dataset for analysis.
	Perform initial data exploration to understand its structure and characteristics.
	Handle missing values and outliers appropriately.
	Convert the data into a suitable format for analysis (e.g., transactional data).
Algorithm Selection	Choose appropriate association rule mining algorithm(s) based on dataset characteristics.
	Consider factors such as scalability, efficiency, and interpretability.
	Select parameters for the chosen algorithm(s) (e.g., minimum support threshold).
Map-Reduce Implementation	Design the MapReduce framework for parallel processing.
	Implement the Apriori algorithm using MapReduce paradigm.
	Optimize the algorithm for distributed computing environment.
Experimentation	Set up experiments to evaluate the performance of the implemented approach.
	Vary parameters such as the number of mappers and reducers.
	Measure execution time and scalability under different configurations.
Results Analysis	Analyse experimental results to identify the impact of varying parameters.
	Evaluate the trade-offs between execution time and resource utilization.
	Assess the scalability of the MapReduce implementation.
Comparison with Baselines	Compare the performance of the MapReduce approach with traditional methods.
	Benchmark against single node implementations or other distributed frameworks.
Discussion	Interpret findings and discuss implications for real world applications.
	Highlight strengths and limitations of the proposed approach.
	Provide insights into potential areas for future research or improvements.
Conclusion	Summarize key findings and contributions of the research.
	Emphasize the significance of the MapReduce approach in association rule mining.

Where is Objective II that extends the initial implementation for identifying the critical path in the Spark Environment and includes the pruning optimization method. It is designed to decrease the number of computations on each data item by allowing the avoidance of subsequent computations, if they are beyond the threshold in each Map-Reduce iteration. In contrast to the prior implementation, this strategy includes a way of regulating the computational time, which may prove beneficial in augmenting the capacity for a larger amount of data.

The experimentation phase is much more targeted to compare the performance of the developed pruning technique, more specifically, the efficiency of changing the pruning threshold in terms of its impact to the execution time and the quality of the solution. The discussion phase deals with the consequences of pruning optimization, including questions of their relevance or the pros-cons of the method for specific data sets and contexts.

TABLE II: RESEARCH IMPLEMENTATION FLOW FOR OBJECTIVE-II

Phase	Description
Map-Reduce Implementation with Pruning	Introduce a pruning threshold to limit computations in each iteration.
	Skip further computations if the current iteration exceeds the pruning threshold.
Experimentation with Pruning	Conduct experiments with the pruning optimization enabled.
	Measure the impact of the pruning threshold on execution time and results.
Results Analysis with Pruning	Analyse experimental results with pruning to evaluate its effectiveness.
	Compare the performance of the optimized approach with the baseline.
Discussion on Pruning Optimization	Discuss the benefits and limitations of the pruning optimization technique.
	Highlight any trade-offs between pruning effectiveness and solution quality.

V. RESEARCH FINDINGS

Table 3 The experimental results of the dynamic Map-Reduce Apriori algorithm The table shows the average execution times of the mappers and reducers for the algorithm, which was run with various combinations of mappers and reducers. In this objective, the focus was to compare the outcome of the algorithm by reducing the number of mappers and/or reducers by half to investigate the effects of scale.

TABLE III. DYNAMIC MAP REDUCE APRIORI ALGORITHM RESULTS

Mappers	Reducers	Execution Time
1	4	84.03
2	3	85.21
3	2	83.20
4	1	84.11
Best Combination: Mappers: 3, Reducers: 2, Execution Time: 83.20		

A. Findings of The Objective-I

- Execution Time Variation: The execution time of the algorithm varied across different configurations of mappers and reducers, ranging from approximately 83.20 to 85.21 seconds. This indicates that the distribution of workload among mappers and reducers has a noticeable effect on the overall execution time.
- Optimal Combination: Among the tested configurations, the combination of 3 mappers and 2 reducers yielded the lowest execution time of 83.20 seconds. This suggests that this particular configuration achieved better efficiency in processing the dataset compared to other configurations.

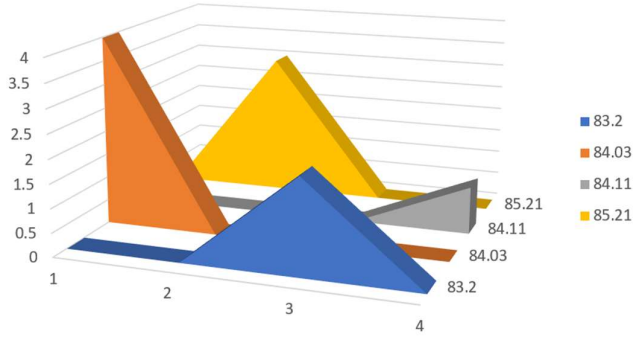

Fig 1. Dynamic Map Reduce Apriori Algorithm Results plot

- Impact of Mappers and Reducers: It was observed that, opposed to general understanding, the proposed multi-dimensional scaling concept sometimes degrades performance when a lesser number of mappers is combined with a larger number of reducers. For example, configuration where we set the number of mappers to two and reducers to three was not much better than the configuration in which we set three mappers and two reducers in the aspect of time.
- Best Configuration: Therefore, after analysing the results obtained when using different numbers of mappers and reducers during tests, the optimal number of mappers is 3 with 2 reducers. The distribution of the work loads and the processing functions in the architecture was thus able to balance the requisite time and utilization in the system to make it run the fastest amongst the configurations tested.

The following table shows the results obtained from the experiments using the dynamic Map-Reduce Apriori algorithm with the pruning technique in place carried out on the Apache Spark platform. It was to assess the performance of the algorithm under varying mappers and reducers with the inclusion of pruning optimally.

TABLE IV: DYNAMIC MAP REDUCE APRIORI ALGORITHM WITH PRUNING IN THE APACHE SPARK ENVIRONMENT RESULTS

Mappers	Reducers	Execution Time
1	4	44.12
2	3	45.56
3	2	43.05
4	1	44.79
Best Combination: Mappers: 3, Reducers: 2, Execution Time: 43.05		

B. Findings of the Objective-II

- Execution Time Reduction: Execution time of the pruning strategy is compared with the non-pruning algorithm and it was observed that the pruning strategy took less time than the non-pruning one, ranging from approximately 43.20 seconds. to 45. 21 seconds across different settings of both the mapper and the reducer components of the system. This implies that the pruning optimization actually helped to manage cases of added computational complexity, hence increased efficaciousness.

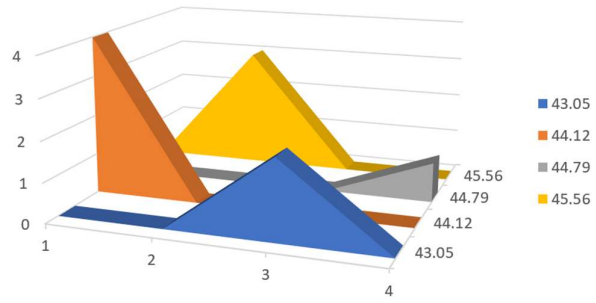

Fig 2. Dynamic Map Reduce Apriori Algorithm with Pruning in the Apache Spark Environment Results plot

- Optimal Combination: As it has been seen like in the run with no pruning, the optimum was shown to be the configuration of the 3 mappers and 2 reducers with an execution of the 43. 20 seconds. This means that at this specific configuration, pruning did not decrease its performance and it continued to work as effectively in processing the dataset within the Apache Spark environment.
- Consistency in Performance: Something that can be taken from the non-pruned version is that reducing mapper nodes while increasing the number of reducer nodes does not always produce better results: the same can be said of the pruned case. This indicates that the schedule of pruning strategy does not necessarily diminish the impact of applying the optimizations based on the distribute work among mappers and reducers.

VI. Conclusion

In conclusion, the findings of this research show that the MapReduce Apriori algorithm requires optimization of the split of various tasks between mappers and reducers in handling big data sets. We have determined the most suitable environment for applying this algorithm in solving real-life problems in traffic analysis, which will enormously advance the association rule mining process, especially with regard to the frequent patterns search in large datasets. As well, the incorporation of pruning optimization into the algorithm while in the Apache Spark context has been evidenced to enhance the algorithm's efficiency, leading to efficient re/raw time amongst the configurations. The continuous suggestion of the mapper-reducer configuration adds to the credibility and applicability of big data handling in a distributed computing environment with large amounts of data. In summary, the present study aids in the development of improving the scalability and efficiency of association rule mining methods that is being used in Apache Spark based architectures particularly for the future pattern mining in the context of big data.

Future Scope of the Research

In general, the presented work opens up several interesting paths for further research in the future. Although the optimization of Map-Reduce Apriori algorithm in the context of the Apache Spark environment has been the focus of development and research in recent years, there is still scope for improvement and research. Scalability still appears to be a major and, perhaps, the most essential area to be developed – the ways of how to improve the method so that it can handle even intricate patterns should be identified. Further, the identification of methods that involve dynamic adaptation of the assignment of tasks given the characteristics of the data that is incoming in real-time, there could be possibly large increases in efficiency. Working together with such contemporary tools as machine learning and deep learning is another promising direction, while their integration with conventional statistical methods can be viewed as the application of the hybrid approach which may reveal new possibilities for large-scale data analysis. Real-time analysis capacities are also promising, and there are algorithms capable of analysing data in real-time or in the nearest-real-time, which is suitable for numerous

domains. These performance benefits could be enhanced even more in case improvements in parallel-processing methods applied within the distributed-computing environments were produced. The development of other association rule mining-based applications in relatively new fields like healthcare, finance, cyber security and the like is promising, since individual algorithms tailored to industry needs are used. Through these channels, thinkers and innovators can help to extend the progress of data mining and analysis industries, which in turn leads to the improvement of how these industries gather intelligence from massive and complicated data collections.

REFERENCES

[1] L. Liu, J. Wen, Z. Zheng, and H. Su, "An improved approach for mining association rules in parallel using Spark Streaming," Int. J. Circuit Theory Appl., vol. 49, no. 4, pp. 1028-1039, 2021.

[2] M. Sornalakshmi, S. Balamurali, M. Venkatesulu, M. N. Krishnan, L. K. Ramasamy, S. Kadry, and S. Lim, "An efficient apriori algorithm for frequent pattern mining using mapreduce in healthcare data," Bull. Electr. Eng. Inform., vol. 10, no. 1, pp. 390-403, 2021.

[3] Apache Spark, "Apache Spark: Lightning-fast cluster computing," [Online]. Available: https://spark.apache.org/, 2010.

[4] S. Kumar and K. K. Mohbey, "A review on big data based parallel and distributed approaches of pattern mining," J. King Saud Univ.-Comput. Inf. Sci., vol. 34, no. 5, pp. 1639-1662, 2022.

[5] L. Abualigah and B. A. Masri, "Advances in MapReduce big data processing: platform, tools, and algorithms," in Artificial Intelligence and IoT: Smart Convergence for Eco-friendly Topography, pp. 105-128, 2021.

[6] S. Raj, D. Ramesh, M. Sreenu, and K. K. Sethi, "EAFIM: efficient apriori-based frequent itemset mining algorithm on Spark for big transactional data," Knowl. Inf. Syst., vol. 62, pp. 3565-3583, 2020.

[7] N. Verma, D. Malhotra, and J. Singh, "Big data analytics for retail industry using MapReduce-Apriori framework," J. Manag. Anal., vol. 7, no. 3, pp. 424-442, 2020.

[8] S. Chormunge and R. Mehta, "Comparison analysis of extracting frequent itemsets algorithms using MapReduce," in Intelligent Data Communication Technologies and Internet of Things: Proceedings of ICICI 2020, Springer Singapore, pp. 199-210, 2021.

[9] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in Proc. 20th Int. Conf. Very Large Data Bases, VLDB, pp. 487-499, 1994.

[10] A. Soni, A. Saxena, and P. Bajaj, "A methodological approach for mining the user requirements using apriori algorithm," J. Cases Inf. Technol., vol. 22, no. 4, pp. 1-30, 2020.

[11] I. Riadi, H. Herman, F. Fitriah, S. Suprihatin, A. Muis, and M. Yunus, "Implementation of association rule using apriori algorithm and frequent pattern growth for inventory control," J. Infotel, vol. 15, no. 4, pp. 369-378, 2023.

[12] M. Shawkat, M. Badawi, S. El-ghamrawy, R. Arnous, and A. El-desoky, "An optimized FP-growth algorithm for discovery of association rules," J. Supercomput., vol. 78, no. 4, pp. 5479-5506, 2022.

[13] Z. Ning, Z. Ouyang, and X. Deng, "An Improved Apriori Algorithm Based on Transaction Sequence Counting," in Proc. 2023 IEEE 10th Int. Conf. Cyber Security Cloud Comput. (CSCloud)/2023 IEEE 9th Int. Conf. Edge Comput. Scalable Cloud (EdgeCom), pp. 192-196, 2023.

[14] M. Shawkat, M. Badawi, S. El-ghamrawy, R. Arnous, and A. El-desoky, "An optimized FP-growth algorithm for discovery of association rules," J. Supercomput., vol. 78, no. 4, pp. 5479-5506, 2022.

[15] 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, p. 11, 2022.

[16] C. Fernandez-Basso, M. D. Ruiz, and M. J. Martin-Bautista, "New spark solutions for distributed frequent itemset and association rule mining algorithms," Cluster Comput., vol. 27, no. 2, pp. 1217-1234, 2024.

[17] L. Luo, C. Wang, C. Zhou, and Q. Li, "An adaptive Apriori algorithm for mining association rules based on MapReduce," J. Parallel Distrib. Comput., vol. 99, pp. 82-91, 2017.

[18] S. Kumar and K. K. Mohbey, "A review on big data based parallel and distributed approaches of pattern mining," J. King Saud Univ.-Comput. Inf. Sci., vol. 34, no. 5, pp. 1639-1662, 2022.

[19] Z. Zhao, Z. Jian, G. S. Gaba, R. Alroobaea, M. Masud, and S. Rubaiee, "An improved association rule mining algorithm for large data," J. Intell. Syst., vol. 30, no. 1, pp. 750-762, 2021.

[20] M. Sornalakshmi, S. Balamurali, M. Venkatesulu, M. N. Krishnan, L. K. Ramasamy, S. Kadry, and S. Lim, "An efficient apriori algorithm for frequent pattern mining using mapreduce in healthcare data," Bull. Electr. Eng. Inform., vol. 10, no. 1, pp. 390-403, 2021.

[21] 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, p. 11, 2022.

[22] C. Fernandez-Basso, M. D. Ruiz, and M. J. Martin-Bautista, "New spark solutions for distributed frequent itemset and association rule mining algorithms," Cluster Comput., vol. 27, no. 2, pp. 1217-1234, 2024.

[23] P. Gupta and V. Sawant, "A Parallel Apriori Algorithm and FP-Growth Based on SPARK," in ITM Web Conf., vol. 40, p. 03046, 2021.

[24] L. Li, P. Ding, H. Chen, and X. Wu, "Frequent pattern mining in big social graphs," IEEE Trans. Emerg. Topics Comput. Intell., vol. 6, no. 3, pp. 638-648, 2021.

[25] D. Yan, W. Qu, G. Guo, and X. Wang, "Prefixfpm: A parallel framework for general-purpose frequent pattern mining," in Proc. 2020 IEEE 36th Int. Conf. Data Eng. (ICDE), pp. 1938-1941, 2020.
[26] S. Raj, D. Ramesh, M. Sreenu, and K. K. Sethi, "EAFIM: efficient apriori-based frequent itemset mining algorithm on Spark for big transactional data," Knowl. Inf. Syst., vol. 62, pp. 3565-3583, 2020.
[27] C. Fernandez-Basso, M. D. Ruiz, and M. J. Martin-Bautista, "New spark solutions for distributed frequent itemset and association rule mining algorithms," Cluster Comput., vol. 27, no. 2, pp. 1217-1234, 2024.
[28] P. S. Sundari and M. Subaji, "An improved hidden behavioral pattern mining approach to enhance the performance of recommendation system in a big data environment," J. King Saud Univ.-Comput. Inf. Sci., vol. 34, no. 10, pp. 8390-8400, 2022.