

FP-Growth on Spark for Big Data Pattern Mining: Applications in Data Science

Cassiano Rista¹, Marco Barbosa², Marcelo Teixeira^{1,3}, Mauro Fonseca¹

¹ Federal University of Technology – PR (CPGEI), Curitiba – Brazil

² Federal University of Technology – PR (PPGEPS), Pato Branco – Brazil

³ Federal University of Technology – PR (PPGEEC), Pato Branco – Brazil

rista@utfpr.edu.br

ABSTRACT

Frequent pattern mining is essential for uncovering associations and trends within transactional data. However, traditional algorithms often face scalability issues when processing large datasets. The FP-Growth algorithm addresses this challenge by mining frequent itemsets and association rules efficiently, without relying on the computationally intensive candidate generation step. In this work, we develop a distributed application using Apache Spark and the FP-Growth algorithm to process large-scale datasets. Using the parallelism of the MapReduce paradigm, our implementation achieved significant performance gains, including a speed-up factor of 9.5 and efficiency of up to 90% in certain configurations. These findings demonstrate that the FP-Growth algorithm, when deployed in a distributed environment, is a robust and scalable solution for big data analytics. Furthermore, the results emphasize the practical applicability of such distributed techniques in common Data Science tasks such as product recommendation, customer segmentation, and behavioral analysis.

KEYWORDS. Data Science, Big Data, Pattern Mining, Apache Spark.

Paper topics (indicate in order of PRIORITY the paper topic(s))

- 1. Data Science and Analytics**
- 2. OR Analytics in Statistics and Machine Learning**
- 3. Simulation**

1. Introduction

Frequent pattern mining is an essential technique in data mining that aims to uncover interesting relationships between items within large transactional datasets. A classic example is the “beer and diapers” rule, which famously revealed an unexpected correlation between purchasing these two seemingly unrelated products. Pioneered by [Agrawal et al., 1993], association rule mining has gained significant importance with the rise of “big data”. Modern datasets’ sheer volume and complexity necessitate efficient and scalable algorithms to extract meaningful patterns. One such algorithm, FP-Growth [Han et al., 2000], stands out due to its innovative approach. FP-Growth employs a more efficient strategy, unlike traditional methods like Apriori, which rely on multiple scans of the dataset to generate and test numerous candidate itemsets. It constructs a compact data structure called the frequent pattern tree (FP-tree), which hierarchically represents the frequent itemsets and their support counts. This tree-based structure enables the direct extraction of frequent patterns without explicit candidate generation, leading to significant performance improvements. However, applying FP-Growth to big data presents substantial challenges. The memory requirements for constructing and traversing the FP-tree can become prohibitively high for massive datasets. Moreover, traditional single-machine implementations may struggle to process data within reasonable time frames.

To address these challenges, distributed computing frameworks like Apache Spark offer a compelling solution. Spark’s parallel processing capabilities and machine learning library (MLlib) provide an ideal environment for implementing and scaling FP-Growth. By distributing the data and computations across a cluster of machines, Spark enables efficient processing of large datasets, significantly accelerating the mining process. This work focuses on implementing a frequent pattern mining application leveraging the FP-Growth algorithm on a Spark cluster. The application allows users to define key parameters, such as minimum support and confidence thresholds, to control the mining process.

The implementation process begins with Data Loading and Preprocessing, where the transactional dataset is loaded and prepared to ensure data quality and readiness for the FP-Growth algorithm. Next, the FP-Tree Construction step involves distributing the data across the Spark cluster to construct the FP-tree in a parallel and efficient manner. Following this, the Frequent Pattern Mining phase extracts frequent itemsets from the constructed FP-tree, utilizing the parallelism inherent in the Spark framework. Subsequently, Association Rule Generation is performed to derive strong association rules from the frequent itemsets, adhering to user-defined confidence and support thresholds. Finally, a comprehensive Performance Evaluation is conducted to assess the application’s execution time, resource utilization, and scalability, providing insights into its overall effectiveness. In summary, this paper presents the following contributions:

- The study demonstrates the FP-Growth algorithm’s effectiveness in handling large-scale data within a distributed Spark environment, achieving a near-linear scalability and reduced execution time.
- The article leverages Amdahl’s Law to evaluate the implementation’s performance, focusing on speed-up and efficiency.
- By employing the FP-Growth algorithm, the study identifies frequent itemsets and generates association rules from a large transactional dataset, showcasing its applicability in market basket analysis and other big data scenarios.

The rest of the paper is organized as follows: Section II presents the related work; Section III discusses the background on the technologies involved in the work; Section IV presents

the methodology used; Section V presents and discusses the experiments performed; Section VI presents some conclusions and future work.

2. Related Work

[Essam et al., 2022] introduced enhancements to the parallel implementation of FP-Growth in Spark, reducing memory consumption and execution time through optimized data structures and load-balancing strategies. [Makanju et al., 2016] proposed DPPFP-Growth, which employs a parent-child mapping for granular partitioning, increasing parallel efficiency in the MapReduce model. [Chaturvedi et al., 2023] utilized Spark for pattern mining in social network data, demonstrating its effectiveness for processing large-scale datasets in a distributed manner. [Huang et al., 2021] introduced parallel architectures that minimize data transfer and enhance scalability. In contrast, [Shawkat et al., 2022] proposed an optimized version of FP-Growth, focused on reducing memory and execution time on dense datasets. [Feng et al., 2019] improved recommendation accuracy on large datasets by combining the Pearson correlation coefficient with FP-Growth. [Bagui et al., 2020] tackled performance issues in MapReduce environments with imbalanced datasets through proposed load-balancing heuristics. [Ragaventhiran and Kavithadevi, 2020] optimized FP-Growth using Hadoop and CAN trees, achieving efficient partitioning and reducing redundant calculations.

Although these advancements exhibit significant improvements in scalability and efficiency, there remains a gap in the detailed evaluation of scalability in distributed environments, particularly in practical configurations and hardware limitations. Moreover, studies integrating performance analysis with classical metrics, such as Amdahl's Law, are limited. This work contributes by filling this gap, applying and evaluating FP-Growth on a Spark cluster, and analyzing performance based on well-established metrics.

3. Background

This section introduces key concepts, highlighting the FP-Growth algorithm for frequent pattern mining, the Apache Spark framework for efficient distributed processing, and Spark MLlib's optimized machine learning algorithms for data mining.

3.1. Apache Spark

Apache Spark [The Apache Software Foundation, 2024] is a unified, flexible analytics engine for large-scale data processing widely used in big data applications. Although primarily developed in Scala, it offers Java, Python, and R APIs, making it accessible to many developers, as illustrated in Figure 1.

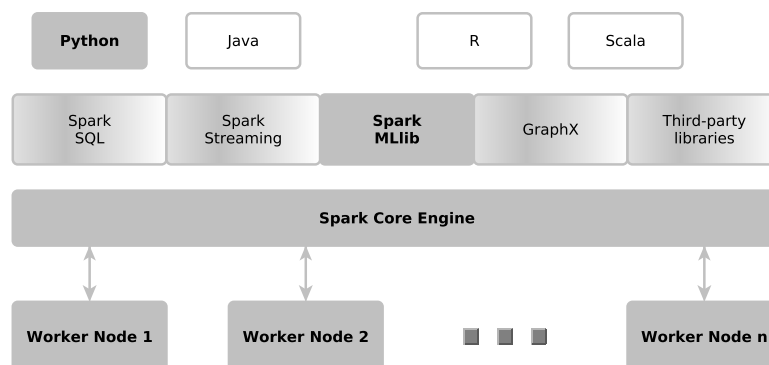


Figure 1: Spark architecture and libraries.

The architecture is composed of several key components, each designed to handle specific data processing tasks, such as SQL and structured data processing (Spark SQL), machine learning (Spark MLlib), graph processing (GraphX), and stream processing (Spark Streaming) [The Apache Software Foundation, 2024]. This modular architecture enables Spark to handle many data processing workloads efficiently. This flexibility allows you to use several languages to write your Spark applications, leverage rich libraries to solve big data problems, and read/write data from various sources. One of the main motivations behind the creation of Spark was to address the complexity of using MapReduce/Hadoop to solve big data problems, which often required writing large amounts of low-level code, even for simple tasks. Spark stands out in this context for its simplicity and efficiency. By leveraging in-memory caching and optimized execution, it is significantly faster than Hadoop and supports a variety of use cases. Our primary interest lies in Spark MLlib. MLlib [Parsian, 2022] stands out as a powerful tool for large-scale data analysis. It offers a comprehensive collection of algorithms for classification, regression, clustering, frequent pattern mining, and more. By seamlessly integrating with the distributed computing capabilities of Spark, MLlib empowers data scientists to build and deploy sophisticated machine learning models on massive datasets. The paper uses PySpark, a Python API for Apache Spark that brings Spark's programming model to Python. By combining Python's accessibility with Spark's robust and expressive capabilities, PySpark provides an intuitive solution for our requirements. It allows the development of Spark applications using Python APIs and offers an interactive PySpark shell for distributed data analysis. PySpark offers two key data abstractions [Parsian, 2022; The Apache Software Foundation, 2024]: RDD (Resilient Distributed Dataset) and DataFrame. An RDD represents a distributed collection of elements of the same type, while a DataFrame is a table of rows with named columns. These abstractions allow you to structure distributed data, enabling higher-level operations. Once data is represented as an RDD or DataFrame, you can apply transformation functions like mappers, filters, and reducers to shape the data for ETL (Extract, Transform, Load) processes, analysis, and data-intensive computations.

Our interest lies in Spark due to its integration with PySpark and the Spark MLlib library, which will enable us to utilize the FP-growth algorithm, as detailed in the following subsection.

3.2. FP-Growth

Frequent pattern mining, involving identifying recurring patterns such as frequent itemsets and subsequences, is a crucial initial step in analyzing large datasets and remains a significant research area in data mining. The FP-growth algorithm, for instance, is particularly effective in discovering association rules from these frequent patterns, enabling the identification of frequently co-occurring items and valuable insights for decision-making. The FP-growth algorithm is described in the literature [Tan et al., 2019; Han et al., 2022], Mining frequent patterns without candidate generation, where "FP" stands for the frequent pattern. The algorithm begins by calculating the frequency of items in a dataset of transactions to identify frequent items. Unlike Apriori-like methods, FP-growth uses a suffix tree structure, called the FP-tree, in its second step to encode transactions directly, avoiding the costly generation of candidate sets. Once the FP-tree is built, frequent itemsets are efficiently extracted from it.

In this paper, we use Spark MLlib [The Apache Software Foundation, 2024], which includes a parallel implementation of the FP-growth algorithm, known as PFP (Parallel FP-growth), as described by [Li et al., 2008]. This implementation is designed to improve scalability by distributing the construction of FP-trees based on transaction suffixes and, hence, is more scalable than a single-machine implementation. It is important to highlight that the FP-growth algorithm operates on item sets, which are unordered collections of unique items. Since Spark lacks a dedicated set type, itemsets are represented using arrays. The algorithm accepts the following parameters [Tan

et al., 2019; Han et al., 2022; The Apache Software Foundation, 2024]:

- **minSupport:** Specifies the minimum support threshold required for an itemset to be considered frequent. The support of an itemset is defined as the proportion of transactions in which the itemset appears. For example, if an itemset occurs in 3 out of 5 transactions, its support is computed as $3/5 = 0.6$;
- **minConfidence:** Determines the minimum confidence threshold for the generation of association rules. Confidence measures the conditional probability of the consequent given the antecedent. For instance, if itemset X appears in 4 transactions and both X and Y appear together in 2 of them, the confidence of the rule $X \Rightarrow Y$ is calculated as $2/4 = 0.5$. This parameter does not affect the frequent itemset mining process itself, but rather serves as a constraint for rule generation from the mined itemsets;
- **numPartitions:** Indicates the number of partitions to be used for parallelizing the computation across the distributed environment. If this parameter is not explicitly specified, the number of partitions defaults to that of the input dataset.

The FP-growth algorithm provides several functionalities for analyzing and extracting patterns from transaction data. One key output is the identification of frequent itemsets. These are represented in a DataFrame with two columns [The Apache Software Foundation, 2024]: `items` and `freq`. The `items` column lists the individual itemsets identified, while the `freq` column shows the number of times each itemset appears in the dataset based on the algorithm configuration parameters. Another significant output is the generation of association rules. These rules are derived from the frequent itemsets and filtered by a minimum confidence threshold (`minConfidence`). The resulting association rules are presented in a data frame with several columns. The antecedent column contains the itemsets that form the hypothesis of the rule, and the consequent column includes itemsets, each containing a single element representing the rule's conclusion. Additional metrics include confidence, which indicates the reliability of the rule, lift, which measures how well the antecedent predicts the consequent relative to random chance, and support, which quantifies how frequently the itemset occurs in the dataset.

The FP-growth model also offers a transformation capability through its `transform` method. For each transaction in the specified column (`itemsCol`), this method evaluates whether the items match the antecedents of any association rules. If a transaction contains all the items in the antecedent of a rule, that rule is applied, and its consequents are added to the prediction result. The `transform` method aggregates the consequences from all applicable rules, creating a prediction column. This column retains the same data type as `itemsCol` but excludes items already present in the original transactions.

4. Deployment Methods

Based on a formalism level, we can consider the following approach for frequent pattern mining [Tan et al., 2019] [Bagui and Dhar, 2019]:

Let:

$$I = \{i_1, i_2, \dots, i_N\} \quad (1)$$

be a set of N unique items, and let D represent the dataset of transactions, where each transaction T can be an item or a set of items and is a subset of I . Each transaction is associated with a unique identifier. Let X and Y be items or itemsets. Thus, our association rule can be expressed in the following form [Bagui and Dhar, 2019]:

$$X \Rightarrow Y, \text{ where } X \subseteq I, Y \subseteq I \text{ e } X \cap Y = \emptyset \quad (2)$$

Support, confidence, and lift configurations determined the association rules for deploying our application. Support measures how frequently an itemset appears, confidence indicates the likelihood of rule validity, and lift evaluates the strength of the association by comparing observed and expected frequencies. These metrics were fine-tuned to optimize the relevance and performance of the association rules, ensuring valuable predictions for decision-making. The MovieLens dataset, publicly available for download on the Kaggle platform, was used for this study. The dataset is 928.45 MB and can be downloaded from [Kaggle, 2024]. It includes ratings and free-text tagging activities from MovieLens, a movie recommendation service. The dataset contains 20,000,263 ratings and 465,564 tag suggestions across 27,278 movies. These data were generated by 138,493 users between January 9, 1995, and March 31, 2015, with the dataset being created on October 17, 2016.

The implementation began with creating a new Spark session and loading the MovieLens dataset in CSV format. Libraries such as *SparkContext* and *SparkSession* are general PySpark libraries required for PySpark applications and were therefore imported during this initial phase. The specific function used for frequent pattern mining was deployment using the FP-Growth algorithm, which is included in Spark MLlib. An exploratory analysis was conducted to gain an initial understanding of the dataset. Data preprocessing was performed using the *DataFrame.summary()* function, which calculates statistics for numerical and string columns. Basic statistics such as count, mean, minimum, and maximum, among others, were generated. This provided a fundamental understanding for the subsequent steps. The initial exploratory analysis provided a foundational understanding of the dataset through essential statistical insights. Following this, the *DataFrame.printSchema()* function was employed to inspect the structure of the Spark data frame, ensuring the data was organized correctly for analysis. This step also facilitated examining the number of baskets per customer and their distribution. Such an evaluation is critical in determining the dataset's suitability for *market basket analysis*¹. The absence of baskets would indicate insufficient data for performing association analysis.

Building on this foundation, the FP-growth algorithm was chosen to identify frequent itemsets within the dataset. FP-growth identifies frequent itemsets in datasets without training, using the FP-tree structure [The Apache Software Foundation, 2024]. This approach is particularly advantageous for market basket analysis, as it directly uncovers item associations from transactional data, making it suitable for real-time and big-data insights. The baskets were collected from the datasets. Since the FP-Growth algorithm requires each basket to be transformed into an array, the result of this transformation is as follows:

$$Array = [item1, item2, item3, \dots] \quad (3)$$

The basket data frame uses a wide format rather than an extended format, with null values for baskets containing fewer than 10 items. Since there should be no null values in the array, they were removed using the *array_except()* function. The data frame has now been prepared for use with the FP-growth algorithm. Thus, the FP-Growth algorithm was employed to identify frequent association patterns among movies. The model was configured with a minSupport of 0.001 and a minConfidence of 0.001, values carefully chosen to balance the dataset's density and the need to capture significant association patterns, even among less frequent itemsets. The *lift* metric was

¹Market basket analysis is a technique used to uncover relationships between items in transactional datasets, commonly applied to retail scenarios to identify items frequently purchased together.

employed to assess the generated association rules' accuracy. The *lift* quantifies the predictive power of the antecedent concerning the consequent and is defined as [The Apache Software Foundation, 2024]:

$$S(A \cup C) / (S(A) \times S(C)) \quad (4)$$

Here, S represents the support, while A and C denote the antecedent and consequent.

From this point, the prediction model was created with the most frequent items and the association rules generated. This model was built based on the generated association rules. To achieve this, the *transform()* function from the Spark library was used, summarizing all the consequents for the applicable rules and generating the predictions. After implementing and deploying the application with the FP-Growth algorithm in the Spark cluster environment, a new question emerged: which performance metric should be used for the testbed? Amdahl's Law [Pacheco and Malensek, 2021] was then used as the metric for subsequent experiments. Traditionally used in HPC (High-Performance Computing), speed-up, as defined by Amdahl's Law, is the factor by which a program can be accelerated through parallelism. Given a program that supports parallelism, P (therefore, $1-P$ is the non-parallelizable part) and the number of available cores N , the maximum speed-up is calculated using the following formula:

$$\frac{1}{(1 - P) + P/N} \quad (5)$$

For example, if the serial part of the program ($1-P$) is $1/4$, the speed-up cannot be greater than 4, regardless of how many cores are used. Factors such as overhead generated by the interconnection network, latency, and I/O can also reduce the application's performance. The following experiments and tests are described in Section 5, along with the experimental environment and configurations.

5. Performance Evaluation

The experiments were conducted using two Dell PowerEdge R710 servers. Each server has two Intel Xeon E5506 CPUs (8 physical cores per node) and 48 GB of RAM. These servers run the Ubuntu Server 20.04.6 LTS 64-bit operating system, Spark 3.2, Jupyter Notebook 1.0, and Python 3.8. A Gigabit Ethernet switch was used to connect the servers. All tests were executed 10 times for each experiment, with a confidence interval of approximately 95%. The experimental environment was established using a standalone deployment, the most basic and uncomplicated method for configuring Apache Spark on a private cluster. This deployment strategy simplifies management and optimizes resource allocation, making it well-suited for testing and evaluating the performance of the implemented application within a controlled setting.

An optimization using Jupyter Notebook within the Spark environment with PySpark was implemented to execute the Python application. This was achieved by setting up SSH tunneling to redirect the default remote port 8889 to the local port 8890. This configuration enabled the notebook to be deployed and tests executed, allowing access from any laptop or workstation. After demonstrating the installation and configuration of the testing environment, the results from the parallel deployment of the FP-Growth algorithm and the performance tests that were conducted can now be presented. Table 1 shows a subset of the most frequent items. Note that only 10 rows are displayed to facilitate understanding, given the dataset size. These frequent itemsets exemplify the FP-Growth algorithm's ability to identify patterns in large-scale datasets, demonstrating its effectiveness in processing transactional data for association rule generation and subsequent performance evaluation.

Following the identification of frequent itemsets, the analysis extracted a subset of high-confidence association rules to illustrate the relationships between co-occurring items in the dataset.

These rules, derived using the FP-Growth algorithm, highlight its scalability and efficiency in handling large transactional datasets, providing key insights for evaluating the algorithm's performance in the Spark cluster.

Table 1: Subset of Frequent Items.

Items	Freq
Bourne Identity, The (2002)	20091
Bourne Identity, The (2002), 5418	20091
Bourne Identity, The (2002), 5418, Action—Mystery—Thriller	20091
Bourne Identity, The (2002), Action—Mystery—Thriller	20091
Star Wars: Episode VI - Return of the Jedi (1983)	46839
Star Wars: Episode VI - Return of the Jedi (1983), 1210	46839
Star Wars: Episode VI - Return of the Jedi (1983), 1210, Action—Adventure—Sci-Fi	46839
Star Wars: Episode VI - Return of the Jedi (1983), Action—Adventure—Sci-Fi	46839
344	38493
344, Comedy	38245

5.1. Results

Based on the association rules, a test was conducted to verify the accuracy of the predictions. Table 2 presents the prediction results after execution. Following the validation of these results, performance tests were performed in the Spark cluster environment using the FP-Growth algorithm and the dataset.

Table 2: Prediction Based on Association Rule Testing.

Basket	Prediction
[Star Wars: Episode VI - Return of the Jedi (1983)]	[1210, Action—Adventure—Sci-Fi]
[Jumanji (1995)]	[2, Adventure—Children—Fantasy]

In order to evaluate the scalability and performance of the FP-Growth algorithm in a Spark cluster, a series of tests were conducted using Amdahl's Law as the performance metric. Traditionally applied in high-performance computing, this metric was employed to calculate the speed-up achieved with varying core configurations. As illustrated in Figure 2, the spark cluster demonstrated a nearly tenfold performance improvement, reducing execution time from 2160 seconds to 228 seconds when utilizing 16 cores, highlighting the effectiveness of parallel processing.

Figure 3 compares the calculated and theoretical speed-up values, showing that the achieved speed-up closely follows the theoretical trend. With 16 cores, the implementation achieved a speed-up of 9.5, confirming near-linear scalability within the tested configuration.

Efficiency values, presented in Figure 4, show how effectively the computational resources were utilized. Efficiency started at 100% with one core and gradually decreased to 60% with 16 cores, as expected in parallel systems due to communication overhead and synchronization. The algorithm exhibited exceptional efficiency with two cores, reaching a peak of 90%, indicating strong performance even with limited computational resources.

The results demonstrate that the algorithm can scale efficiently, utilizing up to the 16 available cores. The speed-up curve in Figure 3 suggests further scalability potential if more nodes were added to the cluster. This observation is critical for handling even larger datasets in future

applications. In summary, the FP-Growth algorithm demonstrated excellent performance and scalability within the Spark cluster environment, achieving significant reductions in execution time and maintaining robust efficiency metrics.

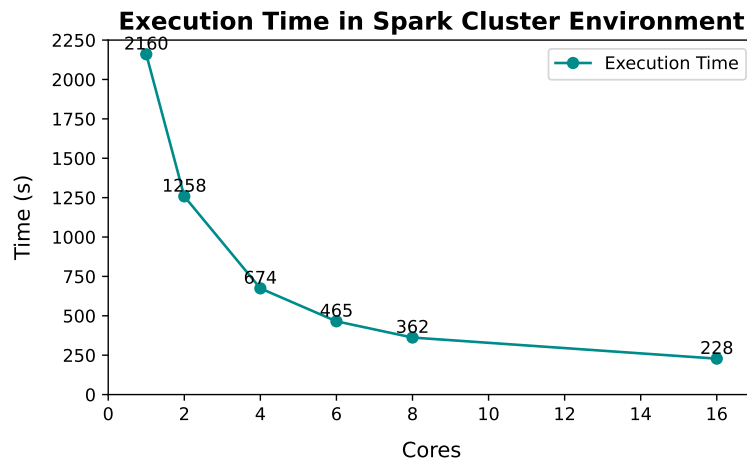


Figure 2: Execution time measured for a two-node Spark cluster environment with different core configurations.

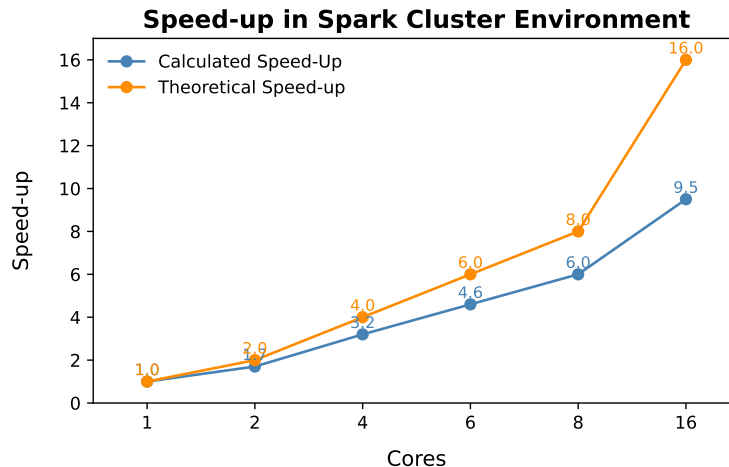


Figure 3: Theoretical and calculated speed-up values in a two-node Spark cluster environment.

5.2. Discussion

The results presented in this study provide critical insights into the performance, scalability, and efficiency of the distributed FP-Growth algorithm implemented in a Spark cluster environment. These results highlight the effectiveness of parallelization in handling large-scale frequent pattern mining tasks and offer a detailed evaluation of performance metrics such as execution time, speed-up, and efficiency. Results reveal a significant reduction in execution time as the number of cores increases, underscoring the impact of parallel computing on the FP-Growth algorithm's performance. With a single core, the execution time was measured at 2160 seconds, which dropped to just 228 seconds when utilizing 16 cores. This nearly tenfold improvement demonstrates the ability of the Spark-based implementation to scale effectively with additional computational resources.

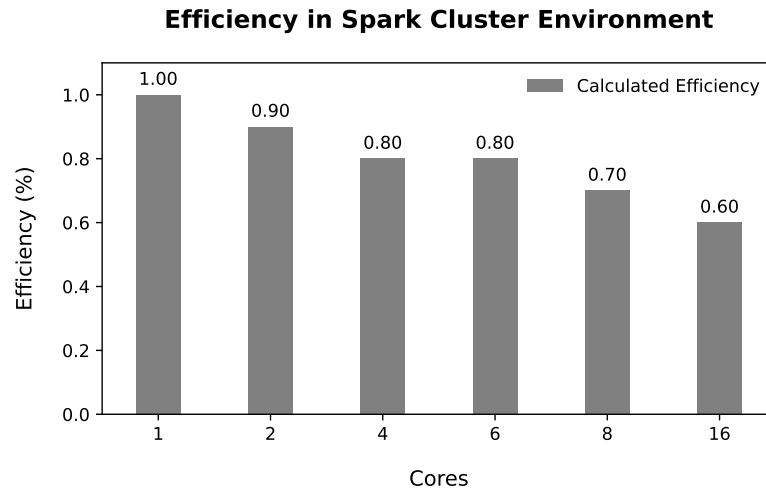


Figure 4: Efficiency calculated for a two-node Spark cluster environment with different core configurations.

The consistent decrease in execution time as cores increase confirms the algorithm's capacity to leverage parallel processing for efficient data mining tasks. The speed-up values illustrate the algorithm's scalability. The observed speed-up closely aligns with theoretical expectations, reaching a value of 9.5 when using 16 cores. This indicates that the implementation approaches near-linear scalability under the tested configurations. The alignment of actual and theoretical speed-up values in Figure 3 reflects the effectiveness of the Spark framework's distributed architecture in minimizing bottlenecks and maximizing resource utilization. The speed-up curve shows a consistent upward trend, suggesting that the implementation can scale further with additional cores and nodes. This observation is critical for real-world applications where the volume of transactional data continues to grow, demanding ever-increasing computational capabilities. The efficiency values reported provide additional context for understanding the resource utilization of the FP-Growth implementation. Efficiency starts at 1.0 with a single core and gradually decreases as more cores are added, reaching 0.6 with 16 cores. This decline in efficiency is typical in parallel computing due to overheads such as data communication and synchronization among nodes. However, the efficiency of 90% observed with two cores is particularly noteworthy, as it highlights the algorithm's ability to maintain high performance in low-core configurations. The gradual decline in efficiency also indicates that the implementation's performance is not heavily impacted by overhead, making it a robust solution for distributed data mining. In addition, Figures 3 and 4 provide a graphical representation of the relationship between speed-up, theoretical speed-up, and efficiency. The close overlap between the calculated and theoretical speed-up lines reinforces the experimental setup's validity and the results' accuracy. Besides, the efficiency bar chart illustrates how resource utilization decreases as the number of cores increases, which is consistent with expectations in distributed systems.

5.3. Lessons Learned

First, the near-linear scalability observed up to 16 cores underscores the distributed FP-Growth algorithm's effectiveness in leveraging parallel processing. This capability is crucial for applications in domains such as market basket analysis, where transactional data volumes can be immense. Second, the high efficiency observed with fewer cores highlights the algorithm's suitability for environments with limited computational resources. This makes it a versatile solution for diverse deployment scenarios. Finally, the slight divergence between calculated and theoretical speed-up at higher core counts points to opportunities for further optimization. Addressing

inter-node communication latency and workload balancing could enhance performance even further. The results demonstrate the robust performance, scalability, and efficiency of the Spark-based FP-Growth implementation. By leveraging Spark's distributed architecture, the algorithm achieves significant reductions in execution time and demonstrates strong scalability, making it a powerful tool for frequent pattern mining in big data environments. The insights gained from these results provide a foundation for further research and optimization, ensuring that the FP-Growth algorithm remains a relevant and effective solution for future data mining challenges.

6. Conclusion and Future Work

This work investigates the application of the FP-Growth algorithm for frequent pattern mining within a Spark cluster. The proposed solution effectively demonstrates its capability to efficiently extract frequent patterns and generate association rules from large datasets, showcasing its scalability and suitability for big data scenarios. The results highlight the advantages of using Spark for distributed processing, achieving significant reductions in execution time while maintaining high efficiency even with limited computational resources. These findings underscore the robustness and scalability of the proposed approach to address the challenges of large-scale data analytics. The experiments carried out reinforce the applicability of the distributed FP-Growth algorithm in Spark as a powerful tool for large-scale data analysis, with direct applications in e-Commerce, personalized marketing, and data-driven decision support. Future research will explore the integration of Petri Net-based approaches, as inspired by the work of [Rista et al., 2018], to further improve the modeling, evaluation, and optimization of big data applications.

References

- Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, p. 207–216, New York, NY, USA. Association for Computing Machinery. ISBN 0897915925.
- Bagui, S., Devulapalli, K., and Coffey, J. (2020). A heuristic approach for load balancing the fp-growth algorithm on mapreduce. *Array*, 7:100035.
- Bagui, S. and Dhar, P. C. (2019). Positive and negative association rule mining in hadoop's mapreduce environment. *Journal of Big Data*, 6:SP – 75.
- Chaturvedi, S., Saritha, S. K., and Chaturvedi, A. (2023). Spark based parallel frequent pattern rules for social media data analytics. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*, p. 168–175.
- Essam, A., Abdel-Fattah, M. A., and Abdelhamid, L. (2022). Towards enhancing the performance of parallel fp-growth on spark. *IEEE Access*, 10:286–296.
- Feng, W., Zhu, Q., Zhuang, J., and Yu, S. (2019). An expert recommendation algorithm based on pearson correlation coefficient and fp-growth. *Cluster Computing*, 22(3):7401–7412.
- Han, J., Pei, J., and Tong, H. (2022). *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann.
- Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12. ISSN 0163-5808.
- Huang, P.-Y., Cheng, W.-S., Chen, J.-C., Chung, W.-Y., Chen, Y.-L., and Lin, K. W. (2021). A distributed method for fast mining frequent patterns from big data. *IEEE Access*, 9:135144–135159.

- Kaggle (2024). Movielens 20m dataset. url <https://www.kaggle.com/code/pavitrasprabhu/movie-recommender-using-pyspark-20m-grouplens/data>.
- Li, H., Wang, Y., Zhang, D., Zhang, M., and Chang, E. Y. (2008). Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys '08*, p. 107–114, New York, NY, USA. Association for Computing Machinery.
- Makanju, A., Farzanyar, Z., An, A., Cercone, N., Hu, Z. Z., and Hu, Y. (2016). Deep parallelization of parallel fp-growth using parent-child mapreduce. In *2016 IEEE International Conference on Big Data (Big Data)*, p. 1422–1431.
- Pacheco, P. and Malensek, M. (2021). *An Introduction to Parallel Programming*. Elsevier Science.
- Parsian, M. (2022). *Data Algorithms with Spark: Recipes and Design Patterns for Scaling Up using PySpark*. O'Reilly Media.
- Ragaventhiran, J. and Kavithadevi, M. (2020). Map-optimize-reduce: Can tree assisted fp-growth algorithm for clusters based fp mining on hadoop. *Future Generation Computer Systems*, 103: 111–122.
- Rista, C., Teixeira, M., Griebler, D., and Fernandes, L. G. (2018). Evaluating, estimating, and improving network performance in container-based clouds. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, p. 00514–00520.
- Shawkat, M., Badawi, M., El-ghamrawy, S., Arnous, R., and El-desoky, A. (2022). An optimized fp-growth algorithm for discovery of association rules. *The Journal of Supercomputing*, 78(4): 5479–5506. ISSN 1573-0484.
- Tan, P., Steinbach, M., Kumar, V., and Karpatne, A. (2019). *Introduction to Data Mining eBook: Global Edition*. Pearson Education.
- The Apache Software Foundation (2024). Apache spark - a unified engine for large-scale data analytics. URL <https://spark.apache.org/>.