International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Discovery of Frequent Itemsets in Distributed Datasets with Reduced Communication Overhead

Houda Essalmi*[1], Anass El Affar[2]

Submitted:14/03/2024 **Revised**: 29/04/2024 **Accepted**: 06/05/2024

Abstract: The identification of frequent itemsets is a vital and complex task in data mining. Conventional approaches for mining frequent itemsets in distributed datasets typically involve considerable communication overhead. To address this, this paper introduces an effective strategy aimed at optimizing communication times in extensive datasets. We provide a novel scheme strategy to reduce the frequency of communications and synchronizations needed for computing global frequent itemsets. We propose an algorithm, named Efficient Frequent Itemsets Finding (EFIF), which uncovers frequent itemsets at slave nodes within distributed environments. Our algorithm efficiently produces significant frequent itemsets by employing an effective candidate pruning technique. Two datasets with varying characteristics and complexities were selected to evaluate the efficiency and effectiveness of the EFIF algorithm in generating distributed frequent itemsets. This comprehensive assessment demonstrates the algorithm's performance across different scenarios. We compared the performance of the EFIF algorithm with the Apriori and FP-growth algorithms using a novel scheme strategy. Experimental results indicate that EFIF surpasses both Apriori and FP-growth in terms of communication and computation costs.

Keywords: *Distributed data mining algorithm, Communication Scheme, Apriori and FP-growth algorithms, Performance Analysis*

1. Introduction

Association rules mining is integral to data mining, focusing on identifying connections between attributes that frequently co-occur in transactional databases [1]. An essential phase in this process involves discovering frequently appearing itemsets [1]. First introduced by Agrawal and Srikant in 1994 [2], the Apriori algorithm is widely recognized as a pioneering and popular technique for generating candidate itemsets that meet specified frequency criteria. Various adaptations of the Apriori algorithm aim to minimize the number of candidate itemsets generated or the number of database scans required.

A significant advancement in association rules mining is the FP-growth algorithm, introduced by Han et al. [3]. Unlike traditional approaches that involve generating candidate itemsets, FP-growth offers a more efficient method for extracting frequent patterns from transaction databases. This algorithmic innovation has revolutionized association rules mining, simplifying and improving the extraction of valuable insights from vast datasets.

The rapid growth of data and computational requirements in today's data-driven society has rendered traditional methods for discovering common itemsets inefficient. To meet the demands of large-scale data analysis, more effective parallel versions have been developed. The sequential Apriori algorithm has paved the way for several evolutionary paths, including the evolution of parallel and distributed algorithms.

One significant algorithm in this domain is the CD (Count Distribution) algorithm, introduced by Agrawal and Shafer [4]. This method represents a foundational approach to data parallelism by offering a streamlined parallelization of the Apriori algorithm. A key advantage of the CD algorithm is its ability to minimize inter-site communication by exchanging only local support values of candidate itemsets across different sites during each iteration.

Apart from the CD algorithm, several other parallel algorithms have made significant contributions to the field of data mining. For example, Park et al. [5] introduced the PDM (Parallel Data Mining) algorithm, Cheung et al. [6] developed the FDM (Fast Distributed Mining) algorithm, and Shintani and Kitsuregawa [7] proposed the NPA (Non-Partitioned Apriori) algorithm. These algorithms, while sharing similarities with CD, incorporate enhancements such as hashing or candidate pruning to improve efficiency and performance.

The evolution of parallel and distributed algorithms for identifying frequent itemsets has been crucial in addressing the challenges posed by the growing volume of data and computational requirements in today's data-centric environment. These algorithms play a pivotal role in enabling efficient and scalable data mining operations.

The foundational DD (Data Distribution) algorithm, introduced by Agrawal and Shafer [4], operates on the principle of task parallelism. In this approach, database partitions from each site are disseminasted to all other sites.

[1] *Laboratory LSI, FPT, USMBA Fez, Morocco*
ORCID ID: 0000-0002-8865-7618
[2] *Laboratory LSI, FPT, USMBA Fez, Morocco*
ORCID ID: 0009-0009-9545-0373
** Corresponding Author Email: houda.essalmi@usmba.ac.ma*

To determine support, each site searches through the entire database, encompassing both local and remote partitions. While differing in implementation, DD shares similarities with both Han et al.'s [8] IDD (Intelligent Data Distribution) and Shintani and Kitsuregawa's [7] HPA (Hash-based Parallel Mining of Association Rules). HPA manages global reduction on a master site, whereas IDD focuses on splitting basic prefix candidates. According to Agrawal and Shafer [4], there are also CAD (Candidate Distribution) algorithms that distribute candidates and transactions based on prefixes, enabling each site to operate independently.

Numerous research initiatives have been dedicated to addressing the complexity of mining frequent itemsets in parallel and distributed settings.

In their research, Mudumba and Kabir [9] introduce a novel technique called Mine-First association rule mining. This approach integrates local and global association rule mining strategies across diverse data sources, consolidating frequent patterns identified in each source to reveal relevant patterns across distributed environments. Moreover, the model can be extended to generate rules tailored to specific objectives. It elucidates significant relationships, offering decision-makers insights into frequent patterns within individual data sources and across the entire distributed environment.

Samudrala et al. [10] proposed a novel distributed architecture for frequent pattern mining using the Spark Framework. Their approach encompasses three primary stages: categorizing and clustering customer data according to seasonal patterns, segmenting customers based on behavior, and leveraging the Apriori algorithm to extract frequent itemsets and association rules in a distributed environment. This method is highly effective in analyzing extensive datasets and delivering precise predictions on customer needs and preferences.

Martin-Prin et al. [11] introduced a distributed SAT-based framework specifically designed for solving the Closed Frequent Itemset Mining (CFIM) problem. Their framework targets minimizing communication overhead in distributed architectures and mitigating bottlenecks from shared memory. By effectively enumerating the complete set of closed itemsets, the approach significantly reduces processing time through a distributed computing paradigm.

Sahoo and Senapati [12] proposed an efficient approach based on load matrices for distributed frequent pattern mining. This method partitions the dataset vertically into multiple segments, distributing them across available system cores for concurrent processing. Experimental findings demonstrate superior performance compared to the conventional Apriori algorithm.

Many current algorithms aimed at mining frequent itemsets in distributed environments often suffer from performance degradation due to extensive data scans and frequent synchronization and communication stages. To address this challenge, we propose a novel algorithm named EFIF (Efficient Frequent Itemsets Finding), specifically designed for efficient mining of frequent itemsets in distributed contexts. Our main objective is to achieve accurate results across all data while minimizing communication and synchronization overhead among distributed sites where the database is located. By reducing candidate generation and communication costs, quantified by the volume of exchanged messages, our approach significantly improves performance.

The article proceeds as follows: Section 2 outlines the architecture scheme adopted in our approach. Section 3 details the EFIF algorithm. Section 4 analyzes the experimental results of our proposed algorithm. Finally, Section 5 presents the conclusion.

2. Proposed Communication Scheme

2.1. Reduction of Communication Cost

Within distributed parallel architectures, communication among processors across sites during frequent itemsets generation often imposes significant overhead (Tseng et al. [13]). However, adopting a Master/Slaves scheme can effectively mitigate this issue by reducing the frequency of communications and synchronizations needed for computing global frequent itemsets. For instance, Vasoya and Koli [14] demonstrated in their work that implementing such a system led to improved time and space complexity. Initially, the Master processor segments the entire database into clusters and allocates these clusters to Slave processors [15]. Each Slave processor then employs an enhanced Apriori algorithm to generate frequent itemsets and forwards the results to the Master processor.

Given that the database is horizontally fragmented across P sites, let's denote C_K as the number of candidate itemsets at pass K. In algorithms following the scheme of broadcast communication, at each pass k, each site Pi is required to broadcast the locally calculated support from site Pi to all other sites [4] , for example, The CD algorithm requires (Pi - 1 * $|C_K|$) communication overhead at each iteration k [16]. In the Master/Slave scheme, during each k iteration, the P Slave sites send a message to the master site, and the master site responds with a message to all P Slave sites. Hence, we can conclude that the total message broadcast size in the Master/Slaves scheme is less compared to the broadcast communication scheme.

2.2. Eliminate the Redundancy of the Generated Candidates

In distributed algorithms that utilize broadcast for message exchange, a common issue arises known as redundant candidate count at each site. This occurs because, at every

iteration, the same set of frequent itemsets is identified, leading to redundant calculations of their global supports across all sites.

In our approach, we address this challenge by centralizing the candidate itemset generation phase at the Master site. This strategic decision enables us to circumvent redundant calculations of candidate itemsets at each Slave site and optimally leverage the distributed system's global resources. Consequently, the Master site takes on the sole responsibility of generating global candidate itemsets and distributing them to all Slave sites. This ensures efficient utilization of system resources while eliminating redundant computations across multiple sites.

3. Proposed Approach

3.1. Proposed method steps

We can represent "I" as a group of items, while the database consists of transactional data. In a distributed setting, the database is divided into partitions $\{DB_1, DB_2, \ldots, DB_P\}$ and distributed across P sites $\{S_1, S_2, \ldots S_p\}$. D represents the size of the entire database, while di represents the size of each partition DB_i.

Consider an example database β illustrated in Figure 1, where the alphabet I= {a, b, c, d, e} (with m = 5 elements). In this scenario, the database is fragmented and distributed across two Slave sites.

Fig. 1. Sample Database β

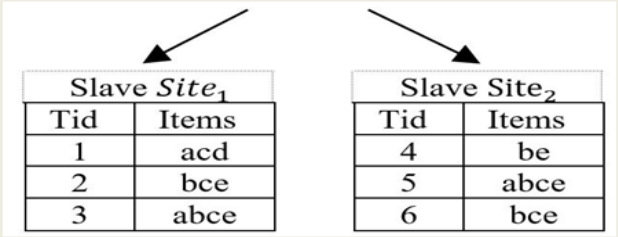

Fig. 2. Slave $Site_1$ and Slave $Site_2$ Databases

Given Itemset Y, Supp(Y) and $Supp_i(Y)$ represent the support of Y in the entire database DB and the subset DB_i, respectively. $Supp(Y) = card(Y)/T_G$ is calculated by dividing the number of transactions containing Y by the total number of transactions in DB, denoted as T_G. Similarly, $Supp_i(Y) = card(Y)/T_i$ is calculated by dividing the number of transactions containing Y by the total number of transactions in DB_i, denoted as T_i. Supp(Y) is referred to as global support, while $Supp_i(Y)$ is referred to as local support on the site of S_i.

An Itemset Y is considered globally frequent if Supp(Y) multiplied by the total number of transactions in the database DB is greater than or equal to the minimum support threshold Sup_{min} multiplied by D. Similarly, Y is locally frequent in the site S_i if $Supp_i(Y)$ multiplied by the number of transactions d_i is greater than or equal to Sup_{min} multiplied by d_i.

The Apriori sequential approach provides the foundation for our proposed algorithm EFIF (Efficient Frequent Itemsets Finding), which finds distributed frequent itemsets, it consists of three major steps as follows:

– Step 1: involves creating the CountList structure, which is a two-dimensional matrix with dimensions (m x m), where m represents the number of items in the database. This matrix serves as a projection of the database. Each cell (i, j) corresponds to the frequency of the itemset composed of elements y_i and y_j. Specifically, the cells on the diagonal represent the frequencies of 1-itemsets, while cells above the diagonal represent 2-itemsets, all with a minimum support threshold (Sup_{min} = 2).

CountList is implemented such that the Itemsets are ordered in lexicographic order at the implementation level, and redundant (symmetric) elements are eliminated. As a result, its size is reduced to ½(m^2+m) as illustrated in the following example:

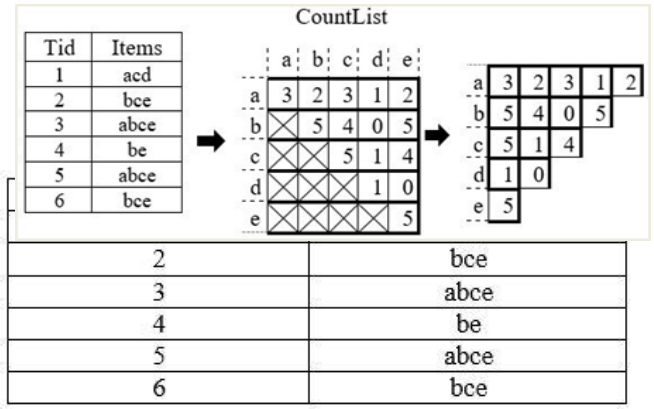

Fig. 3. Optimization of CountList

The CountList structure is created by scanning the β database for every transaction, increasing the frequency of the 1-Itemsets and various 2-Itemsets found in the transaction.

Support (y_i) =CountList (i, 1) / n, $| y_i |=1$,

Support $(y_i y_j)$ =CountList (i, j-(i-1)) / n, $| y_i y_j |=1$, n=| β |.

In our approach, each Slave site computes its local CountList structure ($CountListL_n$) by traversing its respective portion of the local database. Subsequently, we determine the supports of the 1-itemsets by directly accessing the $CountListL_n$ and discard those with support values less than Sup_{min} . From these frequent 1-itemsets, we generate the candidates for 2-itemsets.

It's noteworthy that computing frequent 1-itemsets and frequent 2-itemsets requires just a single traversal of the local database, unlike the Apriori algorithm which necessitates two separate runs. Following this, the Slave sites transmit the content of their $CountListL_n$ to the Master

site. The Master site then calculates the global support in the $CountList_G$ structure by aggregating (summing up the values) all the $CountList_n$ received from the various Slave sites. This process is illustrated in Figure 4 below, depicting a scenario with a Master site and 2 Slave sites.

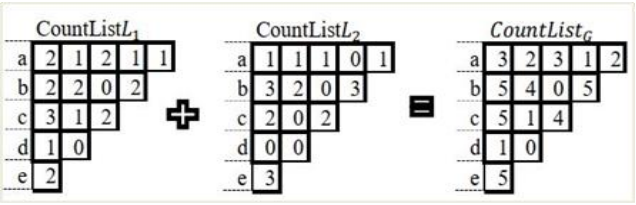

Fig. 4. CountList calculated the sum based on local 1 and local 2

The Master site concludes the list of global frequent 1-Itemsets and frequent 2-Itemsets from $CountList_G$ without database access or communication with the Slave sites. The list of 1-Itemsets includes {a :3, b:5, c:5, e:5}, and the list of 2-Itemsets includes {ab:2, ac :3, ae:2, bc:4, be:5, ce:4}.

– Step 2 Global extraction of frequently occurring K-Itemsets ($K \geq 3$):

The search for frequent k-itemsets ($K \geq 3$) is made more effective by the Master site, which uses a graphical structure to iteratively construct a list of global candidate k-itemsets ($K \geq 3$).

Let us take an example using transactions in Database β, a Master site, and two Slave sites to demonstrate this procedure. Global frequent 2-itemsets {ab, ac, ae, bc, be, ce} are used to start the graphical layout. These sets are organized lexicographically as nodes at level 1. Every node stands for a frequent element along with its backing. At level 2, global 2-itemsets from level 1 are self-joined to create global candidates for 3-itemsets.

The abc node is formed when nodes ab and ac share (k-2) prefixed members. A link is created between these two global frequent nodes and the newly constructed abc node. The least support value of the two global frequent nodes is allocated to the abc node to assess the global support for itemsets abc.

Likewise, the abe node arises from the nodes ab and ae, which have the same first item, 'a'. Between the two nodes, ab and ae, and the abe node, a link is established. The minimum support of the ab and ae nodes determines its approximate support.

Nodes like ab, bc, and be do not have a common prefix, preventing the generation of new nodes in the graphical structure. This process is then applied to other nodes such as ac, ae, bc, be, and ce until all frequent nodes are identified in level 2. This includes abc, abe, ace, and bce, as shown in Figure 5.

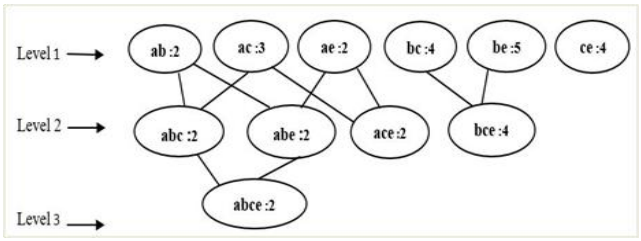

Fig. 5. Finding candidates for K-Itemsets (where K is greater than or equal to 3) using the graphical structure.

At level 3, we identify the 4-itemset nodes by combining the global frequent nodes "abc" and "abe" from level 2, which have a common prefix "ab" of size (k-2). This merging process creates the node "abce" with an approximate support that is the minimum of the "abc" and "abe" global frequent nodes. However, the global frequent nodes "abc" and "ace" do not share the same (k-2) items in common, so no link is established between them. This process continues until no additional nodes are generated. As a result, a list of K-itemset global candidates ($K \geq 3$) is produced along with their approximate support values, without the need for exchanges with Slave sites.

– Step 3: Refining Global Frequent Itemsets:

To reduce the number of global frequent itemsets, a validation step is needed during this stage of the algorithm. The list of potential K-itemsets, where K is larger than or equal to 3, is sent to the Slave sites by the Master site once it is formed in the preceding stage.

The Slave sites then examine the local database sections to determine the actual supports of the received K-itemsets. The Master site receives the computation results after that.

By eliminating non-globally frequent K-itemsets, the Apriori algorithm's Master site determines which K-itemsets are globally frequent. It should be noted that 1-itemsets and 2-itemsets are not included in this procedure since the global $CountList_G$ structure was already used in Step 1 to determine their actual supports.

The Slave sites calculate the local supports of 1-itemset candidates in the first iteration of the Apriori process, which involves a Master site and two Slave sites. They then transmit this data to the Master site. Next, the Master site finds the 1-frequent itemsets by combining the candidates it obtained from the Slave sites. The 2-itemset candidates are computed by the Slave sites in subsequent iterations and sent back to the Master site. For iterations two, three, and four, the same iterative procedure is continued.

Therefore, the EFIF algorithm requires 2 database accesses, one in Step 1 and another in Step 3. In comparison, the Apriori algorithm requires 4 exchanges. By utilizing the graphical structure in Step 2, our algorithm reduces the number of candidate itemsets generated. This not only

decreases the cost of computing frequent itemsets but also minimizes the accesses to local databases of Slave sites for calculating the supports of local candidate itemsets.

3.2. Process of EFIF algorithm

A suggested layout for the methodology of the proposed EFIF algorithm is presented in Figure 6.

Here is a refined version of the process for the EFIF algorithm:

- Database Fragmentation: Partition the database horizontally into a Master site and multiple Slave sites.
- Local Support Calculation: Calculate local support for 1-itemsets and 2-itemsets using the CountList structure at each Slave site, removing those with support below Sup_{min} .
- Transmitting Local CountList: Send the local $CountListL_n$ contents to the Master site.
- Global CountList Calculation: Determine the global $CountList_G$ by aggregating the local CountList structures.
- Generating Frequent k-Itemsets: Identify frequent k-itemsets ($K \geq 3$) from the frequent 2-itemsets list extracted from $CountList_G$, based on an initial graphical structure consisting of a sorted set of global frequent 2-itemsets.
- Approximate Support Calculation: Determine the approximate support at each level of the graphical structure created in Step 2 for each candidate in the frequently occurring k-itemsets list ($K \geq 3$).
- Sending Candidate List to Slave Sites: Distribute the generated list of frequent k-itemsets ($K \geq 3$) to all Slave sites.
- Real Local Support Calculation: Each Slave site computes the actual local supports of the received candidate k-itemsets.
- Returning Results to Master: Send the results back to the Master site to identify frequent k-itemsets that exceed Sup_{min}.
- Extraction of Distributed Frequent Itemsets: Extract distributed frequent itemsets based on the determined frequent k-itemsets.

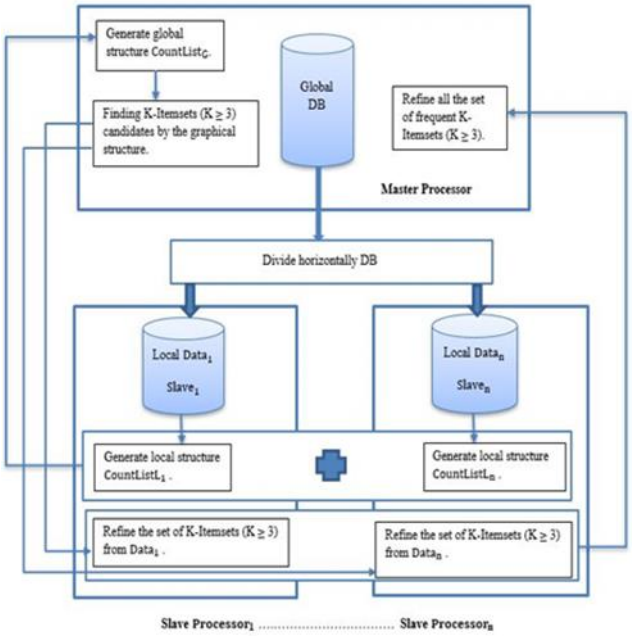

Fig. 6. Architecture of the proposed EFIF

4. Experimental Results

This section employs the T40l10D100K and Chess datasets for experimental evaluation of the EFIF algorithm's performance. A detailed description of these datasets is provided in Table 1.

Table 1. Datasets Description

Dataset Name	***Description***
T40l10D100K	A dataset is available at FIMI [17] and studied by Fournier et al. [18]. It comprises transactions related to a certain domain, with 1000 distinct items, and a total size of 100,000 transactions.
Chess	Another dataset is available at FIMI [17] and utilized in research by Fournier et al. [18]. This dataset represents chess games and consists of transactions with 75 items and 3196 transactions. It serves as a benchmark for evaluating the performance of the EFIF algorithm.

The experimental setup took place within a local network environment. The dataset was horizontally partitioned and distributed across several slave sites. The experiments were conducted on a computer equipped with an Intel® Core™ i7 CPU running at 2.80 GHz, 4GB of RAM, and operating on the Windows 10 platform. The system was tested with varying numbers of slave sites, specifically 3, 5, and 7. The implementation and evaluation of the EFIF algorithm were

carried out using the Java programming language and the NetBeans IDE development environment.

These datasets are chosen to assess the efficiency and effectiveness of the EFIF algorithm in generating distributed frequent itemsets. They offer varying characteristics and complexities, providing a comprehensive evaluation of the algorithm's performance across different scenarios.

We assess the performance of the EFIF algorithm against Apriori and FP-growth algorithms using the Master/Slaves scheme. Researchers Fournier et al. [18] have implemented Apriori and FP-growth in Java. The figures below illustrate the execution time for different support thresholds across datasets T40l10D100K and Chess. The x-axis represents the minimum support, while the y-axis indicates the execution time.

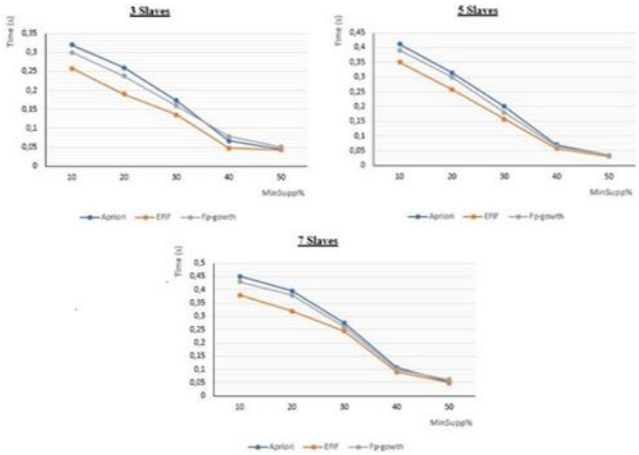

Fig. 7. Assessment of Chess datasets at runtime.

The Chess database has fewer transactions than the other two, and the analysis reveals that the EFIF algorithm performs better, particularly for lower support values, than both the Apriori and FP-growth algorithms. Compared to Apriori and FP-growth algorithms, the EFIF algorithm can generate fewer candidate itemsets, which accounts for its superiority. Consequently, compared to Apriori and FP-growth algorithms, this decrease has a direct influence on communication costs and results in less messages being transferred between sites. Furthermore, compared to the Apriori and FP-growth algorithms, the EFIF algorithm requires at least one fewer iteration, which saves one communication phase.

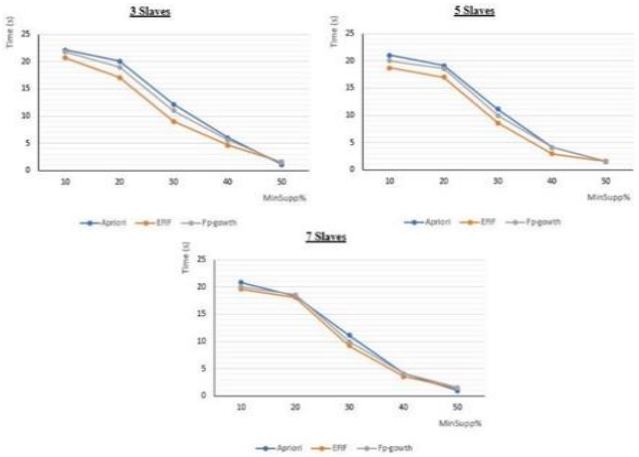

Fig. 8. Runtime assessment for T40l10D100K

To evaluate the EFIF algorithm's scalability, we grew the database size. The EFIF method works better than the Apriori and FP-growth algorithms, as shown by the findings shown in Figure 7. The primary reason for this improvement is that the computation of global frequent itemsets by the EFIF method requires fewer communication phases. Furthermore, we noticed that the algorithms' performance tends to converge as the number of sites (or slave nodes) rises. The convergence becomes particularly noticeable when using more slave nodes—07 slave nodes, for example. For the purpose of calculating global frequent itemsets, the fine granularity of data distribution results in a substantial communication cost.

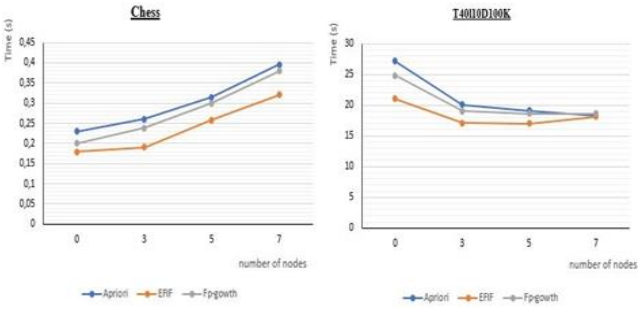

Fig. 9. Scalability of EFIF by number of nodes with Sup_{min} =20%

According to the experiments, the EFIF algorithm outperforms the Apriori and FP-growth algorithms in terms of efficiency and scalability. As the database size, number of nodes, and transactions increase, the EFIF algorithm accelerates significantly. In situations with more parallelism (higher number of nodes), the EFIF algorithm's performance is comparable to the Apriori and FP-growth algorithms. This highlights the EFIF algorithm's strength and flexibility in different data sizes and computational settings.

5. CONCLUSION

This research introduces EFIF, a novel distributed approach aimed at discovering common itemsets within distributed datasets. EFIF employs a Master/Slaves strategy to efficiently locate frequent itemsets across distributed systems, thereby reducing the number of candidate itemsets and minimizing communication overhead. The EFIF methodology consists of three primary steps: first, constructing the CountListG structure by scanning the database to tally the frequencies of 1-itemsets and 2-itemsets per transaction; second, identifying candidates of K-itemsets (where $K \geq 3$) using a graphical approach; and finally, refining the identified frequent itemsets. Experimental evaluations compared EFIF with the Apriori and FP-growth algorithms using datasets with varying minimum support values. The results demonstrate EFIF's superior performance in terms of computation time and communication efficiency, highlighting its scalability in distributed computing environments.

References

[1] R. Agrawal, T. Imieliński et A. Swami, " Mining association rules between sets of items in large databases ", ACM SIGMOD Rec., vol. 22, no. 2, pp. 207–216, 1993, doi: 10.1145/170036.170072.

[2] R. Agrawal, R. Srikant, " Fast algorithms for mining association rules," in Proc. 20th int. conf. very large databases, VLDB. Vol. 1215, 1994, pp. 487-499,doi:10.5555/645920.

[3] J. Han, J. Pei et Y. Yin, " Mining frequent patterns without candidate generation ", ACM SIGMOD Rec., vol. 29, no. 2, pp. 1–12, 2000, doi: 10.1145/335191.335372.

[4] R. Agrawal et J. C. Shafer, "Parallel mining of association rules ", IEEE Trans. Knowl. Data Eng., vol. 8, no. 6, pp. 962–969, 1996, doi :10.1109/69.553164.

[5] J. S. Park, M.-S. Chen et P. S. Yu, "An effective hash-based algorithm for mining association rules ", ACM SIGMOD Rec., vol. 24, no. 2, pp. 175–186, 1995,doi:10.1145/568271.223813.

[6] D. W. Cheung, Jiawei Han, V. T. Ng, A. W. Fu and Yongjian Fu, "A fast distributed algorithm for mining association rules, " Fourth International Conference on Parallel and Distributed Information Systems, Miami Beach, FL, USA, 1996, pp. 31-42, doi: 10.1109/PDIS.1996.568665.

[7] T. Shintani and M. Kitsuregawa, "Hash based parallel algorithms for mining association rules, " Fourth International Conference on Parallel and Distributed Information Systems, Miami Beach, FL, USA, 1996, pp. 19-30, doi: 10.1109/PDIS.1996.568664.

[8] E.-H. Han, G. Karypis et V. Kumar, "Scalable parallel data mining for association rules ", ACM SIGMOD Rec., vol. 26, no.2, pp. 277–288, 1997,doi:10.1145/253262.253330.

[9] B. Mudumba et M. F. Kabir, "Mine-first association rule mining: An integration of independent frequent patterns in distributed environments ", Decis. Analytics J., pp. 100434, 2024, doi :10.1016/j.dajour.2024.100434.

[10] K. Samudrala, J. Kolisetty, A. S. Chakravadhanula, B. Preetham and R. Senapati, "Novel Distributed Architecture for Frequent Pattern Mining using Spark Framework, " 2023 3rd International Conference on Intelligent Technologies (CONIT), Hubli, India, 2023, pp. 1-5, doi: 10.1109/CONIT59222.2023.10205903.

[11] J. Martin-Prin, I. O.Dlala, N.Travers, and S. Jabbour, "A Distributed SAT-Based Framework for Closed Frequent Itemset Mining", In International Conference on Advanced Data Mining and Applications. Cham: Springer Nature Switzerland, November.2022, pp. 419-433, doi: 10.1007/978-3-031-22137-8_31.

[12] A. Sahoo and R. Senapati, "A Novel Approach for Distributed Frequent Pattern Mining Algorithm using Load-Matrix, " 2021 International Conference on Intelligent Technologies (CONIT), Hubli, India, 2021, pp. 1-5, doi: 10.1109/CONIT51480.2021.9498411.

[13] F. S. C. Tseng, Y.-H. Kuo et Y.-M. Huang, "Toward boosting distributed association rule mining by data de-clustering ", Inf. Sci., vol. 180, no 22, pp. 4263–4289,2010, doi:10.1016/j.ins.2010.07.020.

[14] A. Vasoya et N. Koli, "Mining of Association Rules on Large Database Using Distributed and Parallel Computing ", Procedia Comput. Sci., vol. 79, pp. 221–230, 2016, doi: 10.1016/j.procs.2016.03.029.

[15] T. Tassa, "Secure Mining of Association Rules in Horizontally Distributed Databases, " in IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 4, pp. 970-983, April 2014, doi: 10.1109/TKDE.2013.41.

[16] M. Z. Ashrafi, D. Taniar and K. Smith, "ODAM: An optimized distributed association rule mining algorithm, " in IEEE Distributed Systems Online, vol. 5, no. 3, 2004, doi: 10.1109/MDSO.2004.1285877.

[17] B. Goethals, M. J. Zaki, "FIMI'03: Workshop on frequent itemset mining implementations", In Third IEEE International Conference on Data Mining Workshop on Frequent Itemset Mining Implementations,2003, pp.1–13, doi: 10.1145/1007730.1007744.

[18] P. Fournier-viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu et al., "Spmf: a java open-source pattern mining library", The Journal of Machine Learning Research, vol.15, pp.3389-3393, 2014, doi : 10.1007/978-3-319-46131-1_8.