A Scalable Distributed Approach for Exploration Global Frequent Patterns

Original Scientific Paper

Houda Essalmi*
Laboratory of Engineering Sciences, Polydisciplinary Faculty of Taza,
University of Sidi Mohamed Ben Abdellah
Fez, Morocco
houda.essalmi@usmba.ac.ma

Anass El Affar
Laboratory of Engineering Sciences, Polydisciplinary Faculty of Taza,
University of Sidi Mohamed Ben Abdellah
Fez, Morocco
anass.elaffar@usmba.ac.ma

*Corresponding author

Abstract *– Finding patterns in transactional databases regularly is an essential part of data mining since it makes it simpler to identify significant connections and reoccurring patterns in datasets. Scalable, high-performance computing solutions that employ parallel computing systems to optimize resource efficiency and data analysis as data volumes continue to grow are necessary for efficiently processing large databases. To solve these issues, this paper presents Exploration Global Frequent Patterns (EGFP), a new parallel algorithm designed to generate global frequent patterns in different distributed datasets. By facilitating the distribution of workloads and data partitioning, the approach reduces communication costs and ensures efficient parallel execution. Our approach uses two prefix-tree structures to generate a significantly compacted and structured representation of frequent patterns. The first structure local-tree serves to store local support values to effectively collect and arrange transaction data. Global prefix counts are then aggregated and ranked to improve frequency-based analysis and provide a more organized and useful representation of frequent patterns. To find the globally prevalent patterns, a Master site develops a second structure global-tree for each prefix based on this arranged data. Experimental results on large-scale benchmark datasets show that EGFP outperforms other existing methods including CD and PFP-tree in terms of execution time and scalability, while incurring considerably less communication cost.*

Keywords: *Data mining, Parallel Processing, Frequent Patterns tree, Communication costs*

Received: March 2, 2025; Received in revised form: April 25, 2025; Accepted: April 28, 2025

1. INTRODUCTION

The great progress in technology and research in recent years has greatly affected the increasing data volume. Datasets including many complex attributes usually grow exponentially. Distributed data mining is the method of evaluating large datasets maintained across several linked sources or servers, therefore supporting decision-making and revealing hidden information inside the distributed database that calls for specific knowledge. Essential in Data Mining are classification, association rule mining, sequential pattern detection, and other activities [1]. In a transaction database, the interactions among data values are complicated and many of those relationships are effectively implicit. In the discipline of data mining, association rule mining [2] is a rather popular method, it aims to find relationships among itemsets contained in transaction databases or other data sources [3]. Effective counting of all frequent patterns depends on Apriori methods, which produce appropriate rule sets. To find regular patterns inside a transactional database, Apriori algorithms [4] require two main phases candidate generation and pruning, i.e., the elimination of uncommon itemsets is used in an iterative approach in the process. Initially, it finds individual frequent items with values above a minimum support threshold; then by combining them with other frequent itemsets, it generates more combinations. The candidates are further evaluated using the set support threshold. This process continues till no more frequent itemsets can be generated.

The sequential Apriori technique is essentially an essential component of both parallel and distributed algorithms. Association rule mining and optimization depend much on parallel and distributed techniques, thereby improving load distribution and accelerating computation execution. Candidate Distribution (CD) [5] among these approaches assigns the produced candidates to several sites to reduce computational repetition. By using transaction allocation, the Distributed Mining Algorithm (DMA) [6] improves distributed data management. Fast Distributed Mining (FDM) [7] maintains result accuracy while reducing communication costs across nodes, improving efficiency. Optimal Distributed Association Mining (ODAM) [8] is interested in mitigating load imbalance and improving association rule computation's efficiency. The Distributed Decision Miner (DDM) [9] addresses distributed data analysis to boost the decision-making process in vast settings. Distributed Decision (DD) [5], which strategically distributes tasks based on resource availability, and Intelligent Data Distribution (IDD) [10], which flexibly impacts data distribution to maximize processing performance, are alternative techniques. Employing a hash-based approach, hash-based Parallel Association Rule Exploration (HPA) [11] increases the effectiveness of parallel association rule exploration. Integration of CD and DD approaches by Candidate Distribution (CaD) [5] helps to effectively manage candidates and reduce computing costs. Skew Handling (SH) [12] solves data distribution differences to distribute the load. For enhanced performance in a distributed environment, hybrid distribution (HD) [10] combines several distribution techniques.

Through tree-based approaches, such as FP-Growth (Frequent Pattern Growth), Apriori-based methods presently facilitate the analysis of frequent patterns. opposed to the Apriori approach, which needs both the generation and assessment of many candidates, tree-based [13-16] solutions develop this process by grouping the data in a simple and organized hierarchical structure.

Although the FP-Tree (Frequent Pattern Tree) [16] reduces searches for patterns and eliminates excessive candidate generation, therefore decreasing searches for patterns and avoiding unnecessary candidate generation, it also presents many drawbacks when used with very large databases. Building the FP-Tree requires maintaining all transactions in memory as a hierarchical structure retained in a record. This structure can grow excessively large and surpass RAM limits for large databases; therefore, it's ineffective for operation. Building an FP-Tree demands multiple processes, including organizing frequently occurring elements and incorporating transactions into the tree. Regarding time and resources, this approach could be very costly, particularly if the database is large and contains several different components. Mostly operating in memory, the FP-Growth approach, which utilizes the features of the FP-Tree, makes implementation difficult in distributed systems. Different versions, such as Parallel FP-Growth [17] and Load Balancing FP-tree (LFP-tree) [18], have been developed to advance scalability, even though they typically involve complex changes.

LFP-Tree intelligently distributes FP subtrees and transactions among compute sites to maximize load balancing, avoiding bottlenecks and minimizing the processing time. Developed for distributed systems such as Hadoop and Spark, PFP-Tree partitions data into subsets handled separately before the final results are combined. Applied to large databases, both the LFP-Tree and PFP-Tree approaches have weaknesses. LFP-Tree maximizes load balancing; however, the issue can find it challenging to distribute subtrees dynamically in the presence of wildly different transactions, thus generating residual problems with balance and overloading some sites. Moreover, subtree operations and coordination could contribute to higher computing costs. Although PFP-Tree is suitable for distributed environments, it causes major communication expenses between sites during the aggregation of final results, therefore influencing general performance.

In most cases, parallel systems in distributed environments improve scalability and performance; yet, they also have some limitations. The communication overhead between sites represents a main issue that may become a limiting factor in the case of frequent essential data exchanges. Moreover, the control of synchronizing across operations may ultimately result in significant latencies, especially when some activities need close coordination. Load imbalance poses a major problem since certain sites are unused while others show too much demand, therefore limiting the general performance of the system. Especially when the number of data needed to be evaluated is significant, these algorithms often require many database scans, therefore optimizing processing time and resource requirements.

Designed to solve the above-mentioned problems and efficiently uncover global frequent patterns inside distributed datasets, this paper presents a new parallel approach called Exploration Global Frequent Patterns (EGFP). Our approach is based on two tree structures, local and global, including the prefix data of the global database.

Unlike most parallel approaches (peer to peer), we first construct the Master-Slave paradigm by distributing the workload among several Slave sites, improving execution time and system scalability. This architecture constitutes a conscious decision in distributed systems when efficient management of resources and centralized control are required. The slave sites principally aim at building the first local tree structure based on prefix items depending on the defined transaction sequence in the local database. Our method simultaneously develops an ancestor table for each prefix to rearrange the locale-tree structures for all Slave sites in descending order by executing a single scan at each local database. Then, depending on the ancestry information of

the initial localized tree structure, the Master site constructs a global tree for all prefixes, iteratively generating frequent global patterns without requiring Slave site communication.

Our EGFP reduces the communication load across several sites of the system by limiting data processing to a single pass (one scan), a crucial consideration in distributed architectures where an excessive number of exchanges can negatively impact performance. Compared to techniques requiring multiple reads and synchronizations, our method facilitates all Slave sites operating their processing independently, hence reducing the need for synchronizing and temporarily storing. We examine our EGFP algorithm's performance against PFP-Tree and CD on actual increasing data quantities. To develop and investigate a tree structure, PFP-Tree and CD both need several data readings. For real-time analytics and large-scale systems, our method maximizes processing by reducing the amount of data accessed to an alone pass.

The structure of the work is as follows: Section 2 contains a review of the existing works. Section 3 presents the notation and problem definition. In Section 4 our proposed method of algorithm is explained. the results with discussion are given in Section 5. Then the paper is concluded this work in Section 6.

2. RELATED WORK

Several research works have been proposed to enhance the efficiency, scalability, and flexibility of pattern mining algorithms in large-scale data settings. In this section, we review seminal contributions in the field of distributed frequent pattern mining.

Fernandez-Basso et al. [19] presented an original method to improve the extraction of prevalent patterns and association rules inside a distributed system by using Apache Spark. The study aims to address in the framework of large datasets the drawbacks of conventional algorithms such as Apriori and FP-Growth. The authors search for several optimizations, including using Spark's distributed design to provide efficient in-memory parallel computing, hence reducing scans and improving productivity. This work simplifies the consumption of resources and execution time while improving the scalability and applicability of knowledge extraction for large datasets.

The authors in [20] provided a novel method for sequential pattern extraction in databases using a tree structure (SP-Tree structure). This approach applies an optimal tree for better sequence structure, hence reducing data redundancy and database scans. The method advances performance on speed and memory use by applying an effective structure; consequently, pattern extraction becomes more suitable for large datasets. By establishing upgraded efficiency and scalability of sequential pattern analysis, this work progresses data mining methods.

Van and Josef [21] introduced a new approach for extracting frequent itemsets inside a distributed and parallel architecture. The FPO-Tree (Frequent Pattern Ordered Tree) structure that the authors provided improves memory compression and organization of transactions, hence reducing database scans. Moreover, they developed the DP3 (Distributed Parallel Preprocessing Pattern Mining) method, indicated to efficiently apply distributed architectures and parallelize the frequent pattern extraction. This approach reduces data transfers between nodes and maximizes workload distribution, thus boosting the scalability and efficiency of Frequent Itemset Mining (FIM). By reducing computational costs and improving analytical performance, the work presents a successful method for handling large data sets.

In [22], the researchers suggested a selective and adaptive approach for extracting frequent patterns in distributed transactional databases. In contrast to conventional methods that generate all frequent patterns, this particular approach operates on demand by extracting only the relevant patterns depending on user requests. In a distributed system, this greatly reduces computing costs and improves resource economy. This concept depends on the quick identification of patterns at the instant the analyst needs them. They offer DDSampling, a new pattern sampling technique. This program chooses a pattern at random from a distributed transactional database such that the selection probability corresponds to its degree of interest.

The study in [23] investigated the issues and possible solutions for parallelizing frequent itemset mining algorithms in large data settings. Their discussion is centered on enhancing the efficiency of conventional mining methods by their execution in parallel computing architectures. In particular, they suggested an approach that splits large datasets and shares computational tasks across several processing units. This method reduces redundancy in candidate generation and enhances overall efficiency by load balancing and better data access patterns. The implementation, evaluated on cloud computing setups, showed that parallelization substantially speeds up mining with result accuracy intact. These observations highlight the effectiveness of parallel mining techniques in managing the growing volume and velocity of data that are characteristic of contemporary big data systems.

In [24], the authors suggested a distributed association rule mining method named Mine-first Association Rule Mining that solves data decentralization and privacy issues in distributed networks. Their method allows each node to mine local frequent patterns without revealing raw data, thereby ensuring data confidentiality and conserving communication overhead. The incorporation of local patterns into global rules is facilitated by an effective aggregation mechanism that considers support and confidence measures, thereby guaranteeing the integrity and validity of the rules discovered.

The distributed framework is extremely effective when used with decentralized data, providing a scalable and privacy-conscious solution that is especially applicable to multi-source environments, including cloud-based analytics and federated platforms.

In another work authors [25] designed a parallel frequent itemset mining algorithm named STB_Apriori, which is specifically tailored for big data environments using the Apache Spark framework. The algorithm overcomes the drawbacks of conventional Apriori algorithms, especially the computational burdens resulting from repetitive candidate generation and processing of large datasets. The suggested approach employs a BitSet-based compression technique to preserve transactional data in compressed Boolean matrices to accelerate bitwise calculation and lower memory usage. Further, the system cleverly takes advantage of Spark's distributed computing framework for parallel processing of the mining task across several nodes. Experimental evaluations show STB_Apriori to be considerably better than state-of-the-art algorithms about execution time and scalability, positioning it in a favorable position for mining frequent patterns in large-scale distributed data.

Rochd and Hafidi [26] suggested DSSS (Distributed Single Scan on Spark), a distributed version of SSFIM (Single Scan Frequent Itemset Mining) [27], to efficiently mine frequent itemsets in big data environments with Apache Spark. DSSS conducts a single pass over the data by broadcasting a compressed dataset to nodes via RDDs and broadcast variables, unlike conventional multi-scan approaches. It includes early elimination of infrequent items and pruning of unpromising candidates to enhance memory and communication overhead. The experimental results confirm its ability for scalability and efficiency, showing its suitability for cloud and streaming environments.

3. NOTATION AND PROBLEM DEFINITION

Let $I= \{x_1, x_2, x_3,..., x_n\}$ be a collection of n distinct elements. A pattern or a collection of elements constitutes a subset of this set, defined by $X \subseteq I$. A database DB is essentially a collection of transactions, where each transaction T is a subset of I and is uniquely identified by its transaction identifier (TID). The number of database transactions presenting pattern X expressed as sup(X), signifies its level of support value. The equation of support is calculated as follows [2]:

$$sup(X) = (count(X))/N \quad (1)$$

where $count(X)$ represents the occurrence of X in the database and N defines the entire number of transactions. A pattern is considered frequent if its level of support surpasses the minimal support threshold, marked by the symbol ξ. Frequent pattern mining essentially is interested in identifying each pattern that frequently appears in a transaction database while maintaining the specified support threshold ξ.

A distributed system consists of n sites, each labeled, S_1, S_2, S_3,..., S_n.Under this system, the database DB is horizontally divided into n parts, shown as DB_1, DB_2, DB_3,..., DB_n, where each part is assigned to a particular site for data processing (for $i = 1,..., n$). To measure the frequently occurring pattern X, we indicate $Sup_i(X)$ as its local support count, and $Sup(X)$ as its global support count in the whole distributed system. Globally frequent a pattern X satisfying the minimal support criteria ξ, determined using the formula [4]:

$$Sup(X) \geq \xi \times |DB| \quad (2)$$

In a distributed database setting, this requirement is essential since it ensures that the pattern shows consistently over several partitions.

4. PROPOSED METHOD

This section presents our proposed method of algorithm, centered on the Master/Slave paradigm in a distributed computing environment, which can efficiently enable parallel and distributed frequent pattern mining. Additionally provided is an extensive description of the mining process and its purposes.

4.1. MASTER-SLAVE PARADIGM

Communication in distributed frequent pattern mining is typically decentralized. At each phase, all of S_i shares its locally calculated support values with every other site, which produces a rapid increase in data exchanges [28]. Network performance may be impacted as a result of the higher levels of communication imposed by growing datasets and site interactions. An alternative is the Master/Slave paradigm, which, as described in [28], offers a centralized communication mechanism. The dataset is divided into clusters by a Master site, which then assigns each cluster to a slave site. The Master site coordinates the entire process and collects information from Slave sites. This operational method is highly beneficial for distributed computing systems because it minimizes time and space complexity and maximizes resource efficiency, according to previous research by Vasoya and Koli [29].

A distributed and parallel method is examined in this work, in which the database is split horizontally over multiple Slave sites, each of which handles an equal share of transactions on its site. The Master site reduces the overhead of inter-site communication by distributing and merging processed data, functioning as the coordinator. Each Slave site obtains unique prefixes and their associated support numbers from the dataset in a single database scan to generate a Local-Tree Structure (LTS). By collecting the values of local supports, the Master site ranks prefixes in descending order of global frequency. The Ancestor Table (AT), which is generated from this data, serves as an essential component for the Global-Tree Structure (GTS), which is built at the Master site. The FP-Tree technique processes a single structured tree recursively, while the Mining Global Frequent Patterns (EGFP) approach

splits the task into multiple smaller GTSs. By investigating patterns at different hierarchical levels without requiring an excessive amount of inter-node communication, this method increases computational efficiency. The following paragraph provides a detailed description of the EGFP algorithm and its computational elements.

4.2. CONSTRUCTION OF LOCAL-TREE STRUCTURE

At this stage, the Local-Tree Structure (LTS), a data structure designed to optimize processing and storage efficiency with a single database investigation, gets formed. The primary LTS structure consists of a root node, called null, and multiple child nodes, each of which represents a different prefix subtree. Every node contains crucial information, including the prefix name, which identifies the associated element, the support count, which records the frequency of occurrence of a path segment in local transactions, and a node-link, which connects related nodes, using a structure similar to the FP-tree [16].

A sequential three-step process is used to construct the LTS. The transactions from the local database are initially added to the tree in a predefined transaction sorting order to ensure consistency. They locate and store local frequency counters. Then, the local counters from several Slave sites are combined into a Global Counter Table, where their values are added up and then arranged in decreasing order of frequency. Finally, the LTS paths are reorganized according to these sorted global prefix counters, refining the tree's structure for optimal efficiency. The following example illustrates how an LTS is constructed from a local database.

Consider the database DB illustrated in Fig. 1(a), where transaction records are distributed across two Slave sites: DB_1 and DB_2, as depicted in Fig. 1(b). The minimum support threshold is set at ξ=2. The initial phase involves constructing a Local-Tree Structure (LTS) at each Slave site by organizing transaction prefixes in lexicographic order. To facilitate this process, an empty Counter Table (CT) is first created, listing all database prefixes alongside their respective frequency counts. The LTS is then built using the FP-tree methodology, inserting transactions based on the CT structure. Unlike the FP-tree's vertically structured Header Table, the CT provides a horizontal representation, mapping elements, and their counts to their first occurrences in the LTS. Each Slave site independently constructs its local LTS by processing transactions from its database segment. Every time a transaction is added, the CT updates the occurrence count for each prefix. The initial structures of LTS_1 and LTS_2, representing Slave sites 1 and 2, are illustrated in Fig. 1(c) and (d). Correspondingly, CT_1 and CT_2 maintain accurate prefix frequency counts within their respective Slave sites. Once the LTSs are built, the contents of CT_1 and CT_2 are transmitted to the Master site, which oversees system coordination and management.

In the second phase, the Master site computes the global support count for each prefix by aggregating the support values from all received CTs. After summing the local counts, the prefixes are sorted in descending order of frequency, prioritizing the most significant patterns. The prefix counter structure is shown in Fig. 1(e), while Fig. 1(f) details the computation process for global support values. Once the support values are consolidated, Fig. 1(g) presents the sorted global prefix counts. This information is then redistributed to all Slave sites, allowing them to reorganize their LTSs accordingly, thereby streamlining subsequent pattern analysis and extraction.

The final step aims to enhance tree efficiency by restructuring LTSs at each Slave site based on the sorted global prefix order, thus avoiding redundant database scans. Only the frequently occurring prefixes contribute to tree reorganization. At the start of this step, each Slave site generates an Ancestor Table (AT), which records prefix frequencies alongside their ancestor relationships. As paths are reorganized, prefix occurrences are updated within the AT, preventing data duplication. For example, in Slave site 1, the original transaction path (A, B, C, E) is reordered as (B, C, E, A). The ancestor relationships are updated as follows: Prefix B has no ancestors, as it serves as the root node. Prefix C has a single ancestor, {B:2}, where 2 represents the support count of B in this path and in the previous path (B, C, E). Prefix E has one ancestor consisting of two prefixes, {B, C:1}. Prefix A has two ancestors: The first ancestor is {C:1}, derived from the (C, A) path, and the second ancestor is {B, C, E:1}. This restructuring allows for more efficient pattern extraction in subsequent analyses.

Fig. 1(h) and (j) illustrate the information repository at Slave sites 1 and 2, detailing each prefix, its support count, and ancestor relationships. Meanwhile, Fig. 1(i) and (k) depict the optimized LTS structures following reorganization at both sites.

After completing the final phase, the local LTSs attain a highly compact structure, preserving all essential details from their respective databases. This restructured format ensures that the ancestry of each prefix node can be accurately retrieved via the Ancestor Table (AT). Once the LTSs are finalized, the collected node data is transmitted to the Master site for global frequent pattern extraction. For instance, if a node X has N ancestors, its representation in the local AT takes the following form: {($X.ancestor_1$: sup ($X.ancestor_1$)),…,($X.ancestor_N$: sup ($X.ancestor_N$)), sup (X)} where each $X.ancestor_1$ is distinct from $ancestor_N$.

When an ancestor appears multiple times across subsequent pattern analysis and extraction.

The final step aims to enhance tree efficiency by restructuring LTSs at each Slave site based on the sorted global prefix order, thus avoiding redundant database different paths, its frequency values must be consolidated to maintain an accurate count. Instead of storing redundant entries, the cumulative support count is

computed as follows: {($X.\ ancestor_1$: sup ($X.\ ancestor_1$) + sup ($X.\ ancestor_N$), sup (X)}. Applying this principle to our example, the computed values for *Slave Site*$_1$ are: B: {Ø,2}, C: {B :2,3}, E: {(B, C :2),2}, A: {(C :1), (B, C, E :1),2}. For *Slave Site*$_2$, the results are: B: {Ø,2}, C: {B :2,2}, E: {(B :1), (B, C :2),3}, A: {(B, C, E :1),2}.

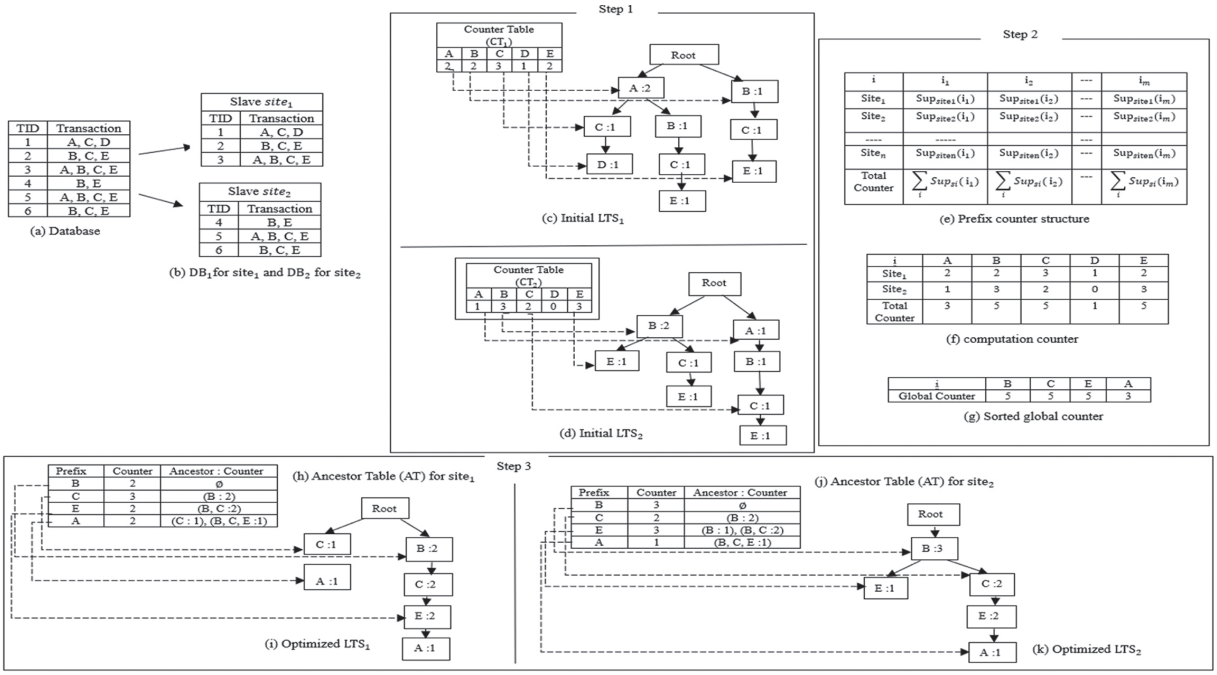


Fig. 1. Construction of Local-Tree Structure

4.3. CONSTRUCTION OF GLOBAL-TREE STRUCTURE

In this section, the pattern exploration process is carried out at the Master site, where we introduce a new frequent pattern extraction technique based on a hierarchical tree structure called the Global-Tree Structure (GTS). Our EGFP algorithm utilizes the GTS through multiple iteration levels K. Instead of relying on a single tree, the approach employs multiple hierarchical GTSs, each designed to organize frequent patterns at different levels of granularity. Each GTS level captures increasingly generalized patterns (e.g., pairs of elements, triplets, etc.) and facilitates targeted pattern exploration. This structured approach reduces the search space while efficiently organizing frequent patterns across various levels of abstraction.

The GTS construction is primarily based on information extracted from the Ancestor Table (AT), which is received from the Slave sites. For each prefix X, all ancestor information gathered from the different Slave sites is combined to build the GTS. The nodes within a GTS contain: all ancestor prefixes, the count of each prefix, node connections, and pointers to a table named Global Counter Table (GCT). The GCT serves as a central repository for aggregated support information from all ancestor prefixes. If the same ancestor elements appear in multiple Slave sites, their support values are accumulated within the corresponding GCT elements. The GTS exploration procedure follows a methodology similar to that of conditional FP-Trees but with a reversed traversal direction. Instead of exploring elements bottom-up, as in an FP-Tree, the GTS traversal moves from top to bottom within the Global Counter Table (GCT). By executing this recursive exploration process, all frequent global itemsets associated with X are efficiently derived.

The GTS construction process is carried out iteratively at each level. At level K=1, a GTS is created for each prefix of size m=1 received from the local Slave sites, as previously described. At level K=2, a GTS is built for each global frequent pattern derived from the first-level patterns of size (m=2). This is achieved by intersecting the paths of frequent global pattern subsets to construct the GTS. At level K≥3, the process continues by identifying the common paths shared among subsets of X of size (m−1) that contain the same (m−2) prefixes at the start of frequent global patterns X. This iterative procedure continues until no further GTSs can be constructed, meaning that all global frequent sets in the distributed database have been identified, and no additional frequent patterns can be generated. If a prefix X has no ancestors, its GTS cannot be constructed. Similarly, if X belongs to a subset of a global frequent pattern XY and X has no ancestors, then the GTS for XY cannot be created, preventing any further global frequent pattern generation.

Fig. 2 illustrates the GTS construction process for each global frequent pattern at different iteration levels K, based on the example provided in Fig. 1(i) and (k). For instance, at level K=1, the GTS extraction process for the global element A proceeds as follows: The global pattern A shares a common ancestor (B, C, E:2) from *Slave Site*$_1$

and *Slave Site*$_2$, as well as a single ancestor (C:1) from *Slave Site*$_1$. Therefore, the GTS is constructed, forming nodes B, C, and E, while accumulating support information for all prefixes in the Global Counter Table (GCT). Since these elements are frequent, a set of frequent item combinations associated with A is generated: {AB:2, AC:3, AE:2}. Next, a GTS is constructed for each derived global frequent pattern (AB, AC, AE) enabling recursive exploration at iteration K=2. At level K=2: The global element AB contains the prefix B, which has no ancestors, so its GTS cannot be built. The global element AC has two prefixes, A and C, meaning an intersection can be made between the paths of GTS$_A$ and GTS$_C$ to construct associated with X are efficiently derived.

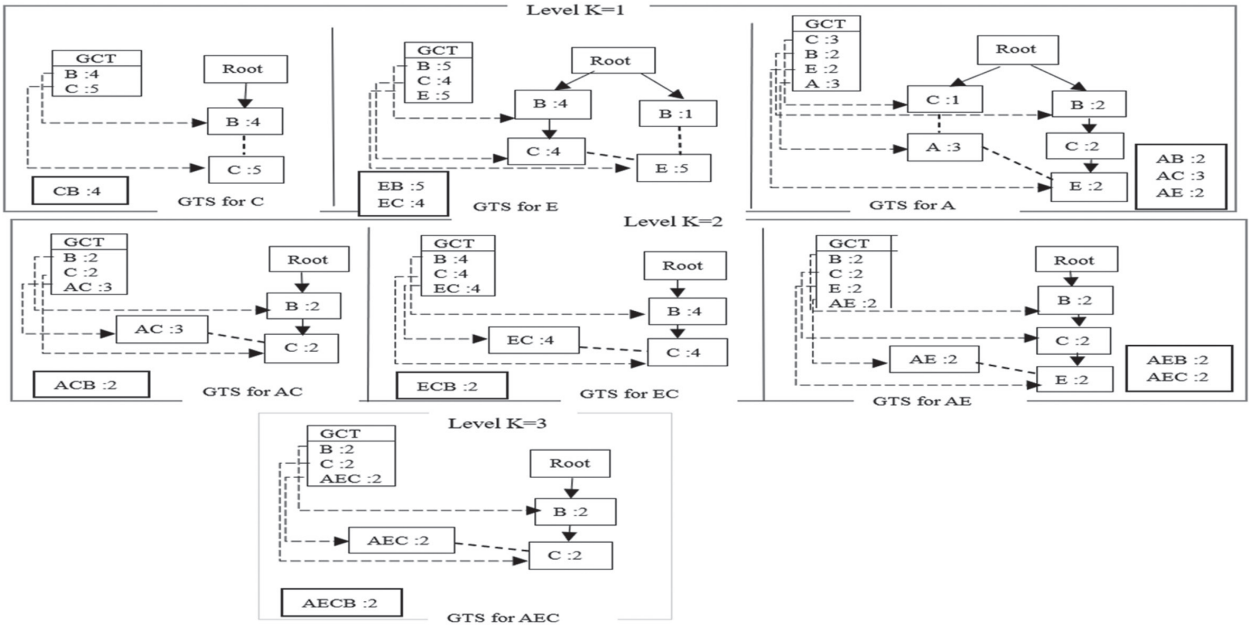


Fig. 2. Construction of Global-Tree Structure

GTS$_{AC}$, leading to the derivation of frequent elements associated with AC: {ACB:2}. The global pattern ACB contains the prefix B, which lacks ancestors, preventing the construction of GTS$_{ACB}$.

This procedure is repeated for the global patterns C and E until all frequent global patterns are extracted. In this example, the process converges in three iterations.

The efficiency of our global frequent pattern exploration procedure lies in its ability to construct a highly compact and optimized GTS, especially in iterations where $k \geq 3$. At this stage, the approach focuses only on shared paths among subsets of a global pattern of size $(m-1)$ that contain the same $(m-2)$ prefixes at the beginning of frequent patterns X. This method significantly enhances the extraction of highly frequent global patterns.

To execute our EGFP (Exploration Global Frequent Patterns) algorithm, we implement a Master/Slave communication model within a fully distributed environment. This intelligent data distribution strategy minimizes communication overhead between sites. Unlike traditional distributed frequent itemset mining algorithms, which require extensive inter-site communication, leading to high network costs, our approach optimizes synchronization while reducing complexity. For comparison, the CD (Count Distribution) algorithm follows the Apriori logic, employing an all-to-all broadcasting approach, which necessitates multiple database scans and explicit candidate generation. While effective, this method can become computationally expensive when handling large-scale data, as it results in increased communication and synchronization overhead.

Applying the above example, the CD algorithm operates as follows: In the first iteration, each Slave site computes the local support of 1-itemsets: *Slave Site*$_1$: {A:2, B:2, C:3, E:2}, *Slave Site*$_2$: {B:3, C:2, E:3}. By exchanging local support counts for each candidate itemset, the two sites can synchronize and calculate the global support: {B:5, C:5, E:5, A:3}. The second iteration computes the local support for 2-itemsets using Apriori-based steps, producing the following outcomes: {AB, AC, AE, BC, BE, CE}. The sites exchange these frequent itemsets with each other. This iterative process continues for iterations 2, 3, and 4, leading to the discovery of: {ABC, ABE, ACE, BCE}, and finally {ABCE}.

The EGFP algorithm leverages two fundamental tree structures: LTS and GTS. Unlike conventional methods, EGFP optimizes database scanning by requiring just a single pass to compute prefix frequencies, significantly enhancing efficiency. In contrast, the parallelized FP-Growth (PFP-Tree) algorithm operates within a fully distributed Peer-to-Peer framework, necessitating two separate scans: the first to compute local frequency counts and the second to reconstruct the local FP-Tree, where items are ordered based on their local frequency. Once local FP-Trees are built, they are progressively merged

into a global FP-Tree. Unlike a Master-Slave model, where coordination is centralized, PFP-Tree requires direct exchanges between sites, increasing communication costs and synchronization complexity especially as FP-Trees grow larger. After the global FP-Tree is formed, it is partitioned into subtrees, and each site executes FP-Growth independently. The resulting frequent patterns are later aggregated to generate the final global set. In contrast, EGFP optimizes communication by reducing it to only two rounds between the Master and Slave sites. Global frequent patterns of size (m=1) are computed in the first round, and Ancestor Table (AT) data is sent to the Master site in the second. To avoid inter-site exchanges during frequent pattern extraction, EGFP uses a highly compressed GTS structure, which reduces the redundancy of frequent elements. However, FP-Growth relies on a single tree to recursively investigate frequent sets without the need for subtree division. EGFP has several benefits, including simplified frequent pattern extraction without extra processing steps and less communication overhead. To maximize overall efficiency and remove reliance on external techniques, the GTS employs an optimized iterative strategy to find frequent global patterns.

4.4. PROCESS OF EGFP ALGORITHM

A detailed explanation of our methodology is provided in Fig. 3. Its purpose is to extract the set of frequently occurring global patterns in a distributed setting. A whole database must first be horizontally divided into local databases that are assigned to various Slave sites by a Master site. Then, using a Counter Table (CT) that determines the elements' support numbers, an initial LTS including local transactions will be constructed for every slave site after calculating each prefix element's support number. To generate the Ancestor Table (AT) and rebuild the paths of each LTS, a global aggregating phase of the local counters is required. The GTS that corresponds to each frequent pattern is subsequently generated, which provides the basis for effectively extracting global frequent patterns.

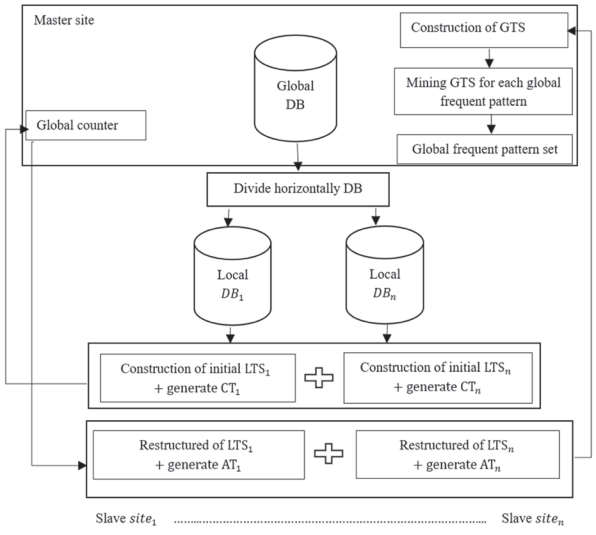


Fig. 3. Process of the Proposed EGFP Algorithm

5. RESULTS AND DISCUSSION

In order to evaluate the performance of our EGFP, we conducted extensive experiments on two kinds of datasets with different characteristics, as shown in Table 1. T10I4D100K and Kosarak are sparse large-scale datasets obtained from FIMI [30]. The T10I4D100K dataset contains a Max TL (Maximum Tree Length) of 29 and an Avg TL (Average Tree Length) of 10.1, showing relatively balanced transaction sizes. Kosarak, on the other hand, has a significantly higher Max TL of 2,498 and an Avg TL of only 8.10, representing a dataset composed of predominantly short transactions with some outliers drastically contributing to tree depth.

We compared EGFP with some previously known algorithms such as PFP-Tree, and CD. The experiments were performed on a system with an Intel® Core™ i7-10875H CPU running at 2.80 GHz, 16 GB of RAM, and operating on Windows 11. To assess scalability and efficiency, the datasets were distributed across 3, 5, and 7 Slave sites. All the programs are implemented in Java using the NetBeans IDE. Communication between sites is facilitated through MPJ Express, a Java-based message-passing library specifically designed for executing parallel applications on multicore processors.

Table 1. Dataset Characteristic

Dataset	Transaction	Items	Max TL (Maximum Tree Length)	Avg TL (Average Tree Length)
T10I4D100K	100000	870	29	10.1
Kosarak	990002	41270	2498	8.10

5.1. ANALYSIS OF RESULTS

The comparative analysis of the PFP-Tree algorithm, CD algorithm, and our proposed EGFP algorithm reveals significant differences in performance, scalability, and efficiency across the T10I4D100K and Kosarak datasets. Fig. 4 compares the execution time of EGFP, PFP-Tree, and CD algorithms for the T10I4D100K dataset, showing how performance scales with the number of Slave sites and minimum support thresholds (MinSupp).

EGFP: Demonstrates superior performance with low execution times across all configurations. For example, with 5 Slave sites and a MinSupp% of 3,5%, EGFP achieves an execution time of 14 seconds, compared to 21 seconds for CD and 19 seconds for PFP-Tree.

CD: Suffers from high communication overhead and redundant computations, leading to significantly higher execution times, especially for lower MinSupp% values.

PFP-Tree: Performs better than CD but is outperformed by EGFP due to the cost of merging FP-Trees and communication overhead.

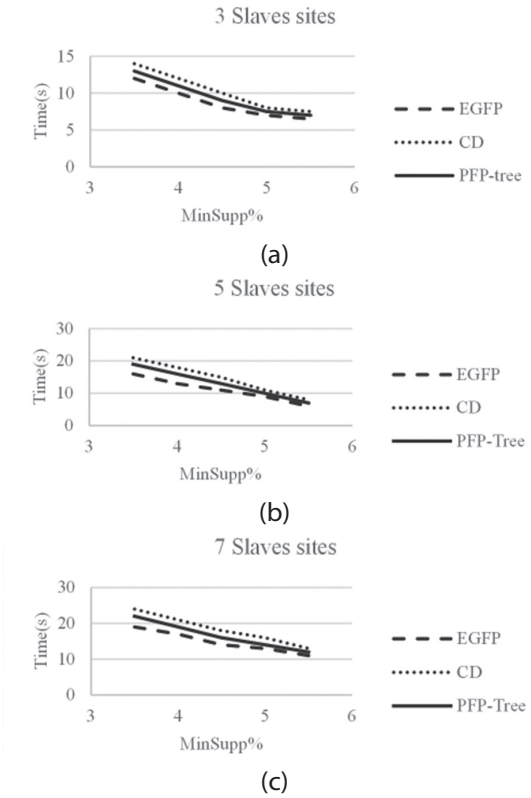


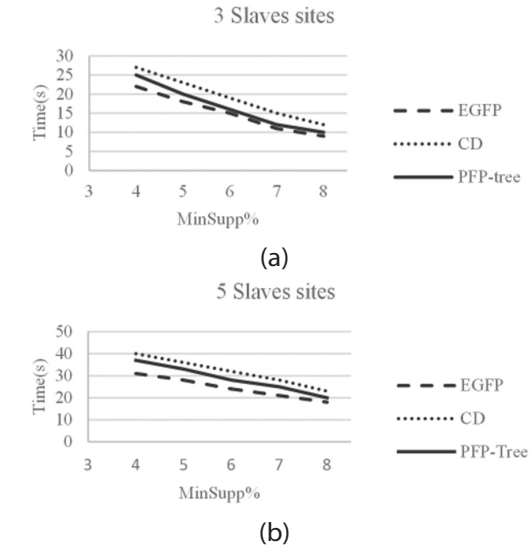
Fig. 4. The running time of T10I4D100K with (**a**) 3 numbers of Slave sites, (**b**) 5 numbers of Slave sites, (**c**) 7 numbers of Slave sites

Fig. 5 compares the execution time of EGFP, PFP-Tree, and CD algorithms for the Kosarak dataset, highlighting the impact of dataset density on scalability.

EGFP: Maintains efficient performance even with the sparse and large Kosarak dataset. For instance, with 7 Slave sites and a MinSupp% of 4%, EGFP achieves an execution time of 40 seconds, compared to 60 seconds for CD and 51 seconds for PFP-Tree.

CD: Struggles with scalability, showing poor performance as the number of Slave sites increases.

PFP-Tree: Performs moderately but is less efficient than EGFP, especially for larger numbers of Slave sites.



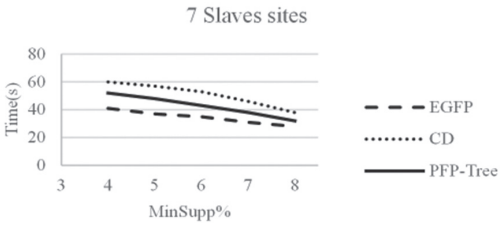


Fig. 5. The running time of Kosarak with (**a**) 3 numbers of Slave sites, (**b**) 5 numbers of Slave sites, (**c**) 7 numbers of Slave sites

The results highlight the strengths and weaknesses of each algorithm in distributed frequent pattern mining:

- EGFP: The use of a Master/Slave architecture and bidirectional communication significantly reduces communication overhead and redundant computations. As a result, EGFP is very scalable and efficient, especially for big datasets Kosarak.
- CD: The broadcast communication approach leads to high communication costs and redundant calculations, making it less efficient for large-scale datasets.
- PFP-Tree: The combination of FP-Trees and communication overhead restricts its scalability and efficiency, even if it outperforms CD.

a) Interpretations

Efficiency of EGFP: EGFP is the most efficient algorithm due to its minimization of redundant computations and concentrated pattern production at the Master site. This is especially evident in its capacity to manage greater numbers of slave sites and capture global frequent patterns using lower minimum support.

Scalability Challenges for CD and PFP-Tree: The communication and computation overheads make CD and PFP-Tree unsuitable for large-scale applications. When the number of slave sites grows, these difficulties become more prominent.

Dataset Impact: The performance gap between EGFP and the other algorithms is more pronounced for the large Kosarak dataset, highlighting EGFP 's ability to handle complex datasets efficiently.

b) Scalability Analysis

The scalability analysis, as shown in Fig. 6, evaluates the performance of EGFP and CD and PFP-Tree as the number of nodes increases.

1. Kosarak Dataset:

- EGFP: Execution time increases gradually with the number of nodes, indicating good scalability. For example, with 5 nodes, EGFP achieves an execution time of 15 seconds, compared to 19 seconds for CD and 18 seconds for FFP-Tree.
- CD and PFP-Tree: Execution time increases significantly with the number of nodes, indicating poor scalability.

2. For T10I4D100K Dataset:

- EGFP: Execution time increases gradually with the number of nodes, maintaining efficient performance.
- CD and PFP-Tree: Execution time increases significantly with the number of nodes, reflecting high communication overhead.

The Key Insights of EGFP Demonstrate excellent scalability, making it suitable for large-scale distributed systems. CD and PFP-Tree Struggle with scalability, particularly for large numbers of nodes and sparse datasets.

Impact of Node Expansion: With an increasing number of Slave nodes, the EGFP algorithm maintained a clear performance edge over PFP-Tree and CD. Even with 7 nodes, EGFP continuously produced faster execution speeds. Although CD and PFP-Tree experienced obvious delays as a result of the growing influence of communication overhead. Efficient communication minimization is a key component of EGFP 's exceptional scalability, enabling it to maintain performance as the system grows.

Performance Trends at Higher Node Counts: As nodes grew, the differences in algorithm efficiency became more evident. At 7 nodes, the increasing communication overload was significantly affecting the performance of PFP-Tree and CD. EGFP provided a distinct advantage, as it benefited from optimized data management and fewer synchronization requests. This capacity allowed it to maintain its efficiency even under high parallelism conditions.

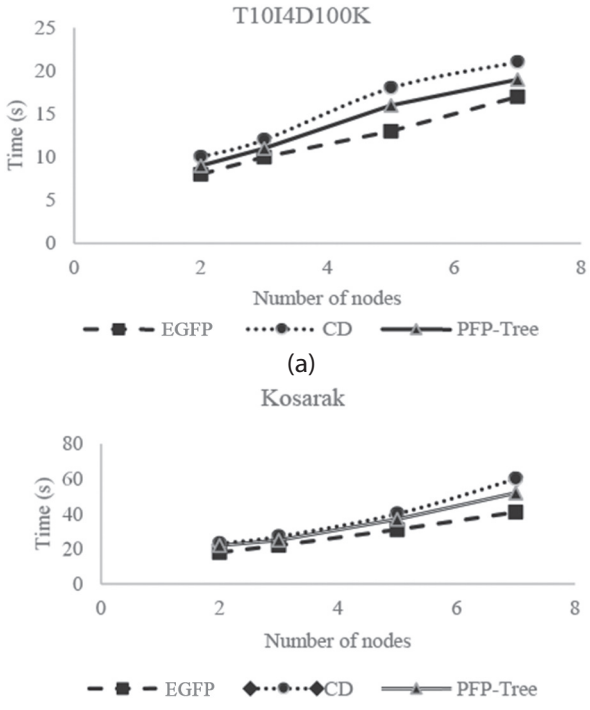


Fig. 6. Scalability of EGFP by various number of nodes for (**a**) T10I4D100K and (**b**) Kosarak with MinSupp = 4%

5.2. DISCUSSION

Our approach employs an iterative process to generate global patterns while leveraging a highly compressed and optimized GTS, ensuring efficient pattern discovery. On the other hand, PFP-Tree relies on conditional sub-tree construction, which reduces redundancy but does not naturally minimize execution overhead. As a result, its computational cost remains significantly higher than that of our algorithm. Additionally, PFP-Tree involves multiple computational steps, such as merging local trees, partitioning the global FP-Tree into subtrees, and executing the FP-Growth algorithm. On the other hand, EGFP removes these complications, which significantly increases execution time and resource efficiency. The CD algorithm exchanges local support values using a straightforward communication paradigm based on all-to-all broadcasting. This strategy generates a lot of candidate sets, which greatly raises the computational and communication overhead even though it is effective in distributing data. A low support threshold causes a rapid increase in the number of generated patterns. Even if parallelization speeds up processing, a significant number of patterns are still extracted overall. EGFP is not a requirement for inter-site communication because it generates all global candidates openly using the GTS. Through the use of the GTS's compression mechanism, this method significantly reduces duplication and communication overhead.

According to the analysis, EGFP performs better on both datasets in terms of efficiency, scalability, and performance than both CD and PFP-Tree. EGFP is the best algorithm for distributed frequent pattern mining because it uses a Master-Slave architecture and bidirectional communication, which significantly reduces the communication cost and unnecessary calculations. With better scalability and faster calculation times than sequential approaches, our algorithm represents a significant advancement in parallel frequent pattern mining. We could focus on investigating fault tolerance strategies to improve EGFP 's resilience in distributed situations and further optimize it for even larger datasets.

6. CONCLUSION

This paper presents a new EGFP algorithm for exploring the discovery of frequent patterns inside a distributed system. Our EGFP remedies the important problems of previous parallel algorithms, including the computational time and communication costs, typically obtained via message exchanges among different sites. The EGFP method compacts and compresses the database using two prefix tree structures. Each Slave site builds its preliminary structure (LTS) to determine the supports of all prefixes, which are subsequently aggregated at the Master site to rearrange the LTS paths for each Slave site by establishing a table including all the ancestors of the local tree nodes. For each element in the local database, the Master site includes ancestral data required for building the Global Tree structure (GTS). The process is car-

ried out iteratively to generate the Global Tree Structure (GTS) for exploring all global patterns. Comparatively to the PFP-Tree and CD algorithms, the performance evaluation of our method on real-world datasets showed its efficiency in speed and scalability.

For distributed systems, real-time data streams, and conditions needing rapid execution, our method, with its optimized approach that limits data processing to a single pass, is extremely fast. In the future work, we plan to concentrate on developing our algorithm with association rule mining to promote efficiency, scalability, and adaptation to complex datasets for the discovery of beneficial patterns and insights.

7. REFERENCES

[1] H. Kargupta, C. Kamath, P. Chan, "Distributed and Parallel Data Mining: Emergence, Growth, and Future Directions", Advances in Distributed and Parallel Knowledge Discovery, AAAI/MIT Press, 2000, pp. 409-416.

[2] R. Agrawal, T. Imieliński, A. Swami, "Mining Association Rules Between Sets of Items in Large Databases", Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington, USA, 25-28 May 1993, pp. 207-216.

[3] P.-N. Tan, M. Steinbach, V. Kumar, "Association Analysis: Basic Concepts and Algorithms", Introduction to Data Mining, Pearson Addison Wesley, 2005, pp. 327-386.

[4] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases", Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, 12-15 September 1994, pp. 487-499.

[5] R. Agrawal, J. C. Shafer, "Parallel Mining of Association Rules", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, 1996, pp. 962-969.

[6] D. W. Cheung, V. T. Ng, A. W. Fu, Y. Fu, "Efficient Mining of Association Rules in Distributed Databases", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, 1996, pp. 911-922.

[7] D. W. Cheung, J. Han, V. T. Ng, A. W. Fu, Y. Fu, "A Fast Distributed Algorithm for Mining Association Rules", Proceedings of the 4th International Conference on Parallel and Distributed Information Systems, Miami Beach, FL, USA, 18-20 December 1996, pp. 31-42.

[8] M. Z. Ashrafi, D. Taniar, K. Smith, "ODAM: An Optimized Distributed Association Rule Mining Algorithm", IEEE Distributed Systems Online, Vol. 5, No. 3, 2004, pp. 1-18.

[9] A. Schuster, R. Wolff, "Communication-Efficient Distributed Mining of Association Rules", ACM SIGMOD Record, Vol. 30, No. 2, 2001, pp. 473-484.

[10] E.-H. Han, G. Karypis, V. Kumar, "Scalable Parallel Data Mining for Association Rules", ACM SIGMOD Record, Vol. 26, No. 2, 1997, pp. 277-288.

[11] T. Shintani, M. Kitsuregawa, "Hash Based Parallel Algorithms for Mining Association Rules", Proceedings of the 4th International Conference on Parallel and Distributed Information Systems, Miami Beach, FL, USA, 18-20 December 1996, pp. 19-30.

[12] L. Harada, N. Akaboshi, K. Ogihara, R. Take, "Dynamic Skew Handling in Parallel Mining of Association Rules", Proceedings of the 7th ACM International Conference on Information and Knowledge Management, Bethesda, MD, USA, 3-7 November 1998, pp. 76-85.

[13] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, Y.-K. Lee, "Efficient Single-Pass Frequent Pattern Mining Using a Prefix-Tree", Information Sciences, Vol. 179, No. 5, 2009, pp. 559-583.

[14] H. Huang, X. Wu, R. Relue, "Association Analysis with One Scan of Databases", Proceedings of the IEEE International Conference on Data Mining, Maebashi, Japan, 9-12 December 2002, pp. 629-632.

[15] G. Grahne, J. Zhu, "Fast Algorithms for Frequent Itemset Mining Using FP-Trees", IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 10, 2005, pp. 1347-1362.

[16] J. Han, J. Pei, Y. Yin, "Mining Frequent Patterns Without Candidate Generation", Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 16-18 May 2000, pp. 1-12.

[17] A. Javed, A. Khokhar, "Frequent Pattern Mining on Message Passing Multiprocessor Systems", Distributed and Parallel Databases, Vol. 16, 2004, pp. 321-334.

[18] O. R. Zaïane, M. El-Hajj, P. Lu, "Fast Parallel Association Rule Mining Without Candidacy Generation",

Proceedings of the IEEE International Conference on Data Mining, San Jose, CA, USA, 29 November - 2 December 2001, pp. 665-668.

[19] C. Fernandez-Basso, M. D. Ruiz, M. J. Martin-Bautista, "New Spark Solutions for Distributed Frequent Itemset and Association Rule Mining Algorithms", Cluster Computing, Vol. 27, No. 2, 2023, pp. 1217-1234.

[20] R. A. Rizvee, C. F. Ahmed, M. F. Arefin, C. K. Leung, "A New Tree-Based Approach to Mine Sequential Patterns", Expert Systems with Applications, Vol. 242, 2024, p. 122754.

[21] V. Q. P. Huynh, J. Küng, "FPO Tree and DP3 Algorithm for Distributed Parallel Frequent Itemsets Mining", Expert Systems with Applications, Vol. 140, 2020, p. 112874.

[22] L. Diop, C. T. Diop, A. Giacometti, A. Soulet, "Pattern on Demand in Transactional Distributed Databases", Information Systems, Vol. 104, 2022, p. 101908.

[23] C. Wu, H. Jiang, "Research on Parallelization of Frequent Itemsets Mining Algorithm", Proceedings of the IEEE 6th International Conference on Cloud Computing and Big Data Analysis, Chengdu, China, 23-25 April 2021, pp. 210-215.

[24] B. Mudumba, M. F. Kabir, "Mine-First Association Rule Mining: An Integration of Independent Frequent Patterns in Distributed Environments", Decision Analytics Journal, Vol. 7, 2024, p. 100434.

[25] D. Fan, J. Wang, S. Lv, "Optimization of Frequent Item Set Mining Parallelization Algorithm Based on Spark Platform", Discover Computing, Vol. 27, No. 1, 2024, p. 38.

[26] Y. Rochd, I. Hafidi, "Frequent Itemset Mining in Big Data with Efficient Distributed Single Scan Algorithm Based on Spark", International Journal of Intelligent Engineering and Systems, Vol. 18, No. 2, 2025, p. 101908.

[27] Y. Djenouri, M. Comuzzi, D. Djenouri, "SS FIM: Single Scan for Frequent Itemsets Mining in Transactional Databases", Advances in Knowledge Discovery and Data Mining, Springer, 2017, pp. 644-654.

[28] T. Tassa, "Secure Mining of Association Rules in Horizontally Distributed Databases", IEEE Transactions on Knowledge and Data Engineering, Vol. 26, No. 4, 2014, pp. 970-983.

[29] A. Vasoya, N. Koli, "Mining of Association Rules on Large Database Using Distributed and Parallel Computing", Procedia Computer Science, Vol. 79, 2016, pp. 221-230.

[30] B. Goethals, M. J. Zaki, "Advances in Frequent Itemset Mining Implementations", ACM SIGKDD Explorations Newsletter, Vol. 6, No. 1, 2004, pp. 109-117.