iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** III **Month of publication:** March 2024

DOI: https://doi.org/10.22214/ijraset.2024.59458

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue III Mar 2024- Available at www.ijraset.com

Distributed Approach for Efficiently Extracting Common Patterns from Large Datasets

Dr. P. Senthil[1], Ms. G. Srividya[2], Cherukuri Ashwith[3], Galipelli Soniya[4], Nethi Pranay[5]
[1, 2]*Assistant Professor,* [3, 4, 5]*B.Tech, Computer Science and Engineering*

Abstract: ***In recent years, advancements in database knowledge discovery have enabled it to find important and practical information with great force. Frequent pattern mining and association rule mining has been thoroughly investigated for a great range of real-world applications. Traditional mining algorithms use memory-resident, centralised data. The performance is not effective enough when applying these methods to remote databases because of the large amount of data, bandwidth limitation, and energy limitations, especially in this era of big data. Datamining in dispersed contexts has therefore become a significant field of study. In order to enhance performance, we provide the group of FP growth-based algorithms that identify FPs with the ability to deliver quick and scalable services in distributed computing settings, We also suggest a concise data structure to hold items and counts in order to reduce the amount of data that has to be transmitted over the network. DistEclat and BigFIM were taken into consideration for the experiment comparison in order to guarantee completeness and execution capacity. Experiments demonstrate that the suggested strategy performs well across a range of experiment settings and offers greater cost-effectiveness for processing large datasets. On average, the suggested approach only needed 33% of DistEclat's execution time and 45% of its transmission cost. Compared to BigFIM, the suggested approach necessitated, on average, 23.3% of the execution duration and 14.2% of the transmission expense of BigFIM***

Index Terms: ***Data mining, parallel algorithms, distributed computing, DistEclat, BigFIM.***

I. INTRODUCTION

Knowledge in databases offers a potent capability for uncovering valuable insights, significantly enhancing business operations' efficiency. Its implementation across various industrial domains have led to the development of numerous datamining methods such as association rule, classification and sequential pattern mining. Frequent pattern (FP) mining and association rule mining, crucial for many real-life applications, has received great attention in recent years.

Artificial intelligence (AI) has played a pivotal role in empowering machines with decision-making capabilities, often without human intervention. The collabration of AI with the Internet of Things (IoT) facilitates data collection, analysis, and automated decision-making, paving the way for the forthcoming era of AI of Things, characterized by the huge growth of data.

Two prominent algorithms, Apriori [3] and FP-growth [4], have been proposed for mining frequent itemsets and association rules. The Apriori [3] approach employs a generation-and-test strategy, generating numerous candidate datasets and repetitively scanning the database to identify frequently occurring patterns. In contrast, FP-growth [4], introduced by Hanet al., utilizes an FP-tree structure to effectively mine Common patterns with just two database scans. However, the time required for finding Common patterns escalates with large database sizes.

Traditional mining algorithms operate on centralized, memory-resident data, which poses challenges in distributed environments, particularly in the times of big data, characterized by vast amounts of data, bandwidth constraints, and energy limitations. Consequently, research interest has surged of data mining for distributed environments. Tehreem et al. also proposed leveraging field-programmable gate arrays (FPGAs) for implementing parallel computing, aiming to enhance performance in such distributed settings.

Previous research has demonstrated the superiority of FP tree-like algorithms over Apriori-like algorithms when leveraging distributed, parallel computing, and Hadoop techniques, as well as algorithms within the Apache Spark framework. These include PFP (parallelize the FP-Growth), PIFP-Growth (parallelized incremental FP-Growth), R-PFP (recursive-PFP), MR-PFP (MapReduce-based parallel frequent pattern growth), PBFP-Growth (parallel block FP-Growth algorithm), MISFP-growth (multiple item support frequent patterns), BigFIM (big frequent itemset mining), and S-FPG (Spark FP-Growth). Additionally, certain approaches have proposed for mining frequent itemsets from secondary memory to accommodate large databases or data structures too large to get into main memory.

©IJRASET: All Rights are Reserved

Two notable algorithms, DistEclat (distributed equivalence class clustering and bottom-up lattice traversal) and BigFIM for the MapReduce platform, have been introduced. DistEclat enhances workload balance among processing units by evenly partitioning the search space through prefixes, while BigFIM integrates principles from Apriori to initially mine frequent itemsets, supporting the DistEclat method for large datasets. Although DistEclat and BigFIM have demonstrated superior performance compared to PFP, they require additional parameters for completion, posing practical challenges for users.

To address scalability issues, the DP (database projection) and aggressive projection algorithms have been proposed, utilizing disk-based structures to manage large databases efficiently. However, these methods entail additional execution overhead due to disk operations, despite their improved memory efficiency.

While previous studies have focused on FP mining and proposed enhancements to execution time efficiency, challenges such as high data transmission time, memory cost, scanning cost, and redundant execution time persist. Neither Hadoop MapReduce nor Apache Spark adequately address these issues simultaneously. To mitigate these challenges, we propose a distributed and parallel computing method named DFP (distributed frequent pattern mining). DFP leverages FP growth in its initial stage while recording information on necessary memory sizes. If frequent patterns cannot be successfully extracted in the first stage, the algorithm estimates required nodes for parallel computing based on recorded information, distributing workload accordingly.

The primary contributions of our study include a set of algorithms based on FP growth optimized for distributed computing environments, along with a compact data structure to minimize network transmission overhead. Experimental comparisons with DistEclat and BigFIM demonstrate the superior cost-effectiveness and scalability of proposed method under various conditions.

This paper has organized as follows: Section II provides background information, Section III introduces our proposed algorithm DFP and explains its application for tree-based FP mining, Section IV presents an analytical evaluation of the proposed method's complexity compared to similar algorithms, and provides results, while Section V concludes with... [complete as appropriate].

II. RELATED WORK

Past research can be categorized into two main areas: A) Association Rule Mining, and B) Distributed Algorithms for Frequent Pattern Discovery, with a focus on efficient algorithms like DistEclat and BigFIM for distributed computing environments.

A. Association Rule Mining

Association rule mining has been extensively studied in recent years (Agrawal et al., 1993, 2001; Agrawal and Srikant, 1995; Bayardo, 1997; Ester et al., 1996). Proposed by Agrawal et al. (1 993), the problem involves discovering association rules, crucial for various data mining applications. Given a database (DB) with transactions and a set of items, association rules are represented as $X > Y$, where X and Y are itemsets, and $X \cap Y = \emptyset$. The data of itemset is the number of transactions in it, while confidence measures the likelihood of Y appearing in transactions containing X. Frequent patterns (FPs) are itemsets with support exceeding a threshold, and their discovery is a prerequisite for ARM.

Two main types of algorithms have employed for mining frequent patterns: Apriori-like and FP growth-like. Apriori-like algorithms generate candidates and repeatedly scan the database, suffering from inefficiency due to memory-intensive candidate generation and multiple database scans. Han et al. (2000a) introduced FP growth, utilizing a tree-based data structure called FP tree, and requiring only two database scans. In the first scan, item support is calculated, and a header table is created. The second scan filters items below the support threshold and constructs the FP tree. FP growth recursively generates conditional FP trees until only single-path trees remain, from which FPs are easily derived.

Han et al. (2000b) proposed the Database Projection algorithm to address memory limitations when mining large databases. This algorithm utilizes disk-based storage for mining information and condenses the database when memory is insufficient, ensuring efficient FP tree construction.

Algorithm 1 Database Projection

Input: Database D and Memory M.

Output: The complete set of frequent patterns.

```
Procedure Database Projection (D. M)
{
T = bulid_FP-tree(D);
IF T≤ M then
{
```

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue III Mar 2024- Available at www.ijraset.com

return FP-growth (T);
}
Else
{
get frequent items i1.i2.....in of D;
decompose D into Di1.Di2.Di3.....Din;
return Database Projection (D_{11}. M) U
Database Projection (Di2. M) U
U
….....
Database Projection (Din. M)
}
}

ARM algorithms serve as the foundation for advanced techniques. For instance, MTARM (multitask association rule miner) explores rule correlations among tasks and integrates Apriori, FP growth, and Eclat algorithms for efficient mining.
Algorithm 1 illustrates the Database Projection procedure, which builds an FP tree from the database and applies FP growth if memory allows; otherwise, it recursively projects and condenses the database until memory constraints are met.
This review provides notice into existing association rule mining techniques and their significance in various data mining applications.

B. Distributed Algorithms For Common Pattern Discovery

Numerous algorithms leveraging distributed or parallel computing have been proposed to enhance execution performance in frequent pattern discovery. For instance, PFP tree utilizes a multiprocessor system, but suffers from increased execution time due to the tree structure and high data transmission costs, especially with larger databases or lower support thresholds. QFP (QFP-growth) also employs a multiprocessor system but faces limitations in FP tree construction efficiency per core processor, leading to redundant execution time.
TPFP-tree (TIDset-based parallel FP-tree) is a parallel-distributed mining algorithm utilizing FP tree, but its reliance on transaction identification sets (TIDsets) may lead to significant memory consumption, particularly with big data.
To address issues in heterogeneous computing environments like grid systems, BTP-tree (balanced TIDset parallel FP-tree) was introduced to reduce communication and tree insertion costs, thus decreasing execution time. However, communication costs may still rise as nodes request data from each other. FD-Mine utilizes cloud computing nodes for FP discovery, employing a compressed data structure based on FP growth to reduce transmission time. Yet, this strategy can still lead to increased data transmission costs and reduced efficiency due to unadaptable nodes.
Apache Spark, introduced in 2010, succeeded Hadoop MapReduce in 2014, offering in-memory parallel computing. S-FPG was one of the first methods to utilize Spark, enhancing processing efficiency for large datasets. Subsequent approaches like caching-based parallel FP growth further improved efficiency, but communication costs remained problematic. Despite Spark's potential speed advantage over Hadoop, its computing costs limit its efficacy for large datasets.
The PFP algorithm, based on Hadoop, employs a MapReduce approach for parallel FP growth. While advantageous, communication costs between mappers can inflate execution time. Variants like PIFP growth and R-PFP address incremental mining and parallelization, yet communication bottlenecks persist. PBFP growth combines Apriori and FP-growth to reduce scanning frequency, but improvements are limited, especially with large databases.
The Hadoop-based mining algorithm BigFIM introduces DistEclat and BigFIM mechanisms for efficient mining. DistEclat focuses on workload balancing, while BigFIM addresses big databases by initially using Apriori for mining, switching to DistEclat when memory limitations arise. Though effective, setting parameters like p can be challenging for novice users.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue III Mar 2024- Available at www.ijraset.com

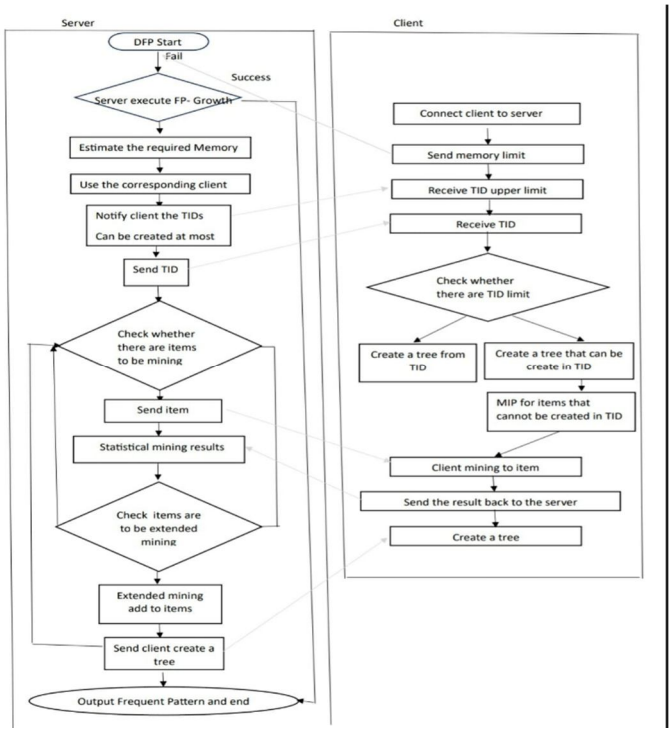


Fig 1. The flow chart of DFP algorithm.

.

Efficient frequent pattern discovery in distributed environments remains a critical research area, with various algorithms addressing different aspects of scalability, efficiency, and usability.

III. PROPOSED METHOD

In this section, we first define the problem in Section III-A and then elaborate on our proposed method, the DFP algorithm, along with its key features in Section III-B.

A. Problem Definition

The FP-growth algorithm encounters inefficiencies when processing large databases. Consequently, various distributed or parallel computing algorithms have been developed. However, these methods often suffer from high communication costs, inadequate memory utilization, low computing efficiency, and redundant computation. To address these challenges, we propose the Distributed Frequent Pattern (DFP) algorithm, building upon the foundation of FP growth. If the initial mining stage fails, DFP estimates the required memory and nodes for subsequent distributed computing steps.

The exceptional performance of DFP is demonstrated in the Experiment Results section. If node memory is insufficient, our extended database projection method, termed mixing projection (MP), is activated. MP efficiently compresses crucial information and facilitates FP tree-based mining, outperforming traditional methods like DP, especially in scenarios where further mining is impossible. MP repeatedly compresses a large database into a condensed data structure until it fits into main memory, albeit at the cost of increased input/output (I/O) operations.

With our proposed MP algorithm and main memory space estimation method, the need for repeated memory space estimation procedures is eliminated. Moreover, if database condensation is required, only specific items are condensed instead of the entire dataset, further reducing I/O costs.

By leveraging the extended MP method, memory capacity evaluation for the database becomes more efficient, leading to improved transmission costs between the server and nodes.

B. DFP Mining Algorithm

In this section, we introduce an efficient DFP algorithm based on the FP tree data structure. The algorithm is divided into two parts: the procedure for the nodes of the server and the client.

©IJRASET: All Rights are Reserved

1) Server Side

The DFP algorithm on the server comprises four stages: memory space evaluation, client node activation, TID distribution, and waiting for mining items distribution.

a) Memory Space Evaluation

As in Fig. 1, Initially, the server executes the FP-growth algorithm to avoid unnecessary transmission time by utilizing the distribution mode and records essential information for memory space estimation. Full garbage collection (GC) is used for determination if the mining has failed. If a full GC occurs, indicating a failed mining attempt, the remaining TIDs are recorded as essential information. This information includes counts, total counts, number of TIDs, and an estimate of the necessary memory space in the db.

b) Client Node Activation

The algorithm estimates the necessary nodes for distributed mining based on the recorded memory space evaluation. Increasing the number of nodes does not necessarily enhance efficiency, so the algorithm selects a limited number of nodes to achieve optimal performance.

c) TID Distribution

The server evaluates activated nodes with respect to processing limitations for TIDs and distributes TIDs accordingly. Each client receives a specific number of TIDs with its memory space allocation.

$$\text{Avg T} = \frac{\text{TIDs}}{\text{number of nodes}}$$

$$\text{TID} = \frac{\text{memory space in nodes}}{\text{memory space in server}} \times \text{avgT}$$

d) Waiting Mining Items Distribution

Once activated, the server performs the FP-growth algorithm and obtains the frequent itemsets in 1-FIs as waiting mining items. It then distributes these items to clients for further mining and waits for the results. The server subsequently mines the common itemsets from the received results, identifies potential mining items, and prompts clients to construct conditional FP trees. The processes continue until all waiting items are processed.

2) Client Side

The DFP algorithm on the client comprises five stages: memory space reporting to the server, receiving the limitation for TIDs, tree construction according to TIDs, mining frequent items, and constructing the conditional FP tree.

a) Memory Space Reporting To Server

Each client reports its memory space to the server upon connection.

b) Receiving The Limitation For TIDS

The server estimates the TID limitation for each client based on reported memory space. If the number of TIDs exceeds the limitation, the MP mechanism is activated to handle the overflow.

c) Tree Construction According To TIDS

Clients construct FP trees based on received TIDs, condensing the database to avoid out-of-memory issues. The MP mechanism reduces I/O costs by incrementing counts according to nodes.

d) Mining Frequent Items

Clients mine frequent items based on expectations from the server and received TIDs. Mining is performed according to FP growth principles, with MP files used to complete the process efficiently.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

e) Construction Of The Condition FP Tree

If potential extended mining items are identified, the server connects to the client to construct a conditional FP tree. Clients extract frequent itemsets and ensure that only relevant items are added to the conditional FP tree, based on information provided by the server.

By following these steps, the DFP algorithm efficiently mines frequent patterns in distributed environments, optimizing resource utilization and reducing computational costs.

IV. IMPLEMENTATION

To accomplish this project, we have designed the following sets:

A. Admin

This module facilitates logging into the admin page, providing access to administrative functionalities and controls.

B. Users

The Users module is responsible for retrieving features from the dataset and displaying the number of users accessing the system.

C. Upload Dataset

This module allows users to upload files or datasets into the system, enabling the incorporation of new data for analysis.

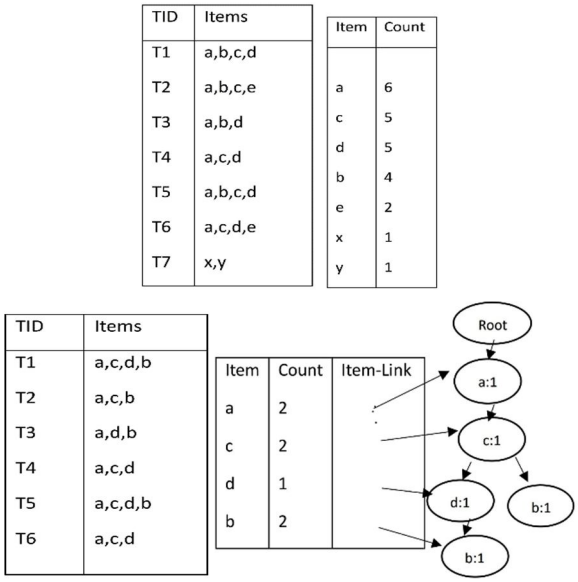


Fig 2. An Vie of memory space estimation

D. View Datase

With this module, users can view and analyze the uploaded dataset, exploring its contents and characteristics

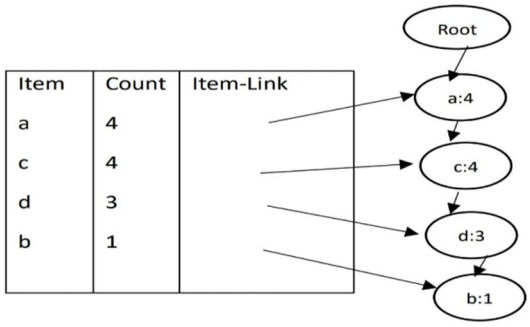


Fig 3. An example of tree construction.

E. Frequent Patterns

The Frequent Patterns module enables the identification and management of frequent items within the dataset, allowing users to analyze patterns and trends in the data.

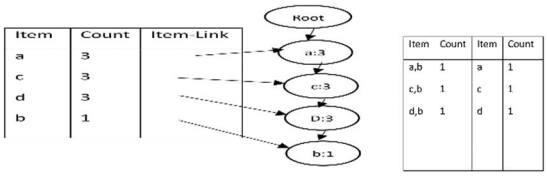


Fig 4. View for mining the frequent items.

1) Input / Output Design

a) Input Design

The input design for this system has been carefully crafted to ensure efficient collection of necessary data while prioritizing user interaction and simplicity. Measures have been implemented to control the amount of input required, prevent unauthorized access, eliminate unnecessary steps, and maintain simplicity throughout the process. Input forms and screens have been designed to streamline user interaction and facilitate ease of use.

b) Output Design

The output design focuses on providing users with screens that enable easy operations in a simple and efficient manner, minimizing the need for excessive keystrokes. Screens emphasize important instructions and information, and error messages are clear and concise.

V. EXPERIMENTAL EVALUATION AND PERFORMANCE STUDY

System testing verifies that all requirements are met by the integrated software system as a whole. It puts a setup to the test in order to guarantee dependable outcomes. The configuration-oriented system integration test is an illustration of a system test. System testing emphasizes pre-driven process connections and integration points and is based on process flows and descriptions.

Evaluation strategy and approach

Functional tests will be meticulously prepared, and field testing will be done by hand.

Objectives of the test: • Every field entry must function correctly.

- You have to click the designated link to activate the pages.
- There shouldn't be any delays in the entry screen, messages, or answers.

Features to be evaluated

- ➢ Verify that the entries are of the correct format
- ➢ No duplicate entries should be allowed
- ➢ All links should take the user to the correct page.

The process of incrementally integrating two or more integrated software components on a single platform to identify interface flaws that lead to failures is known as software integration testing.

The configuration-oriented system integration test is an illustration of a system test. System testing emphasizes pre-driven process connections and integration points and is based on process flows and descriptiom

T……	Estimated memory	DFP clients	FP-growth {execution time
25	4383M	5	177
30	7360M	8	Insufficient Memory
35	9945M	10	Insufficient Memory

T……	Estimated memory	DFP clients	FP-growth {execution time
7	2388M	3	113
13	1397M	2	72
16	1123M	2	61

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

N_K.....	Estimated memory	DFP clients	FP-growth {execution time
5	6712M	7	Insufficient time
10	4727M	5	183
15	2990M	3	88

Fig 5. Execution with various settings of T(35-25), I(16-7), N(15-5K), for DFP and FP-growth on IBM dataset

The purpose of an integration test is to verify that software applications or system components, or even higher up, company-level software applications, function together flawlessly.
Test Results: Every test case that was previously specified was successful. No flaws were found Acceptance Testing

VI. EXPERIMENTAL RESULTS

The aforementioned experiments have demonstrated that DFP exhibits significantly greater scalability compared to DiscEclat and BigFIM. Regarding execution performance, DFP, on average, necessitated only 33% of the execution

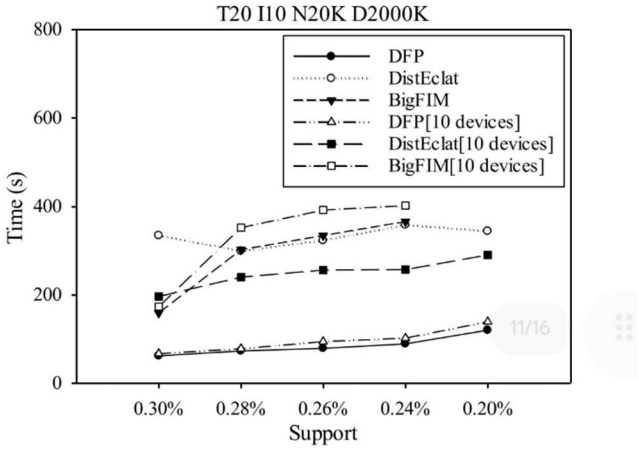


Fig 6. Using an IBM dataset with approximated and ten clients, the execution performance of DFP, DistEclat, and BigFIM with a support level varied from 0.3% to 0.2%. time and 45% of the transmission cost of DistEclat. Additionally, apart from considering execution time and transmission cost, we investigated the size of temporary data generated.

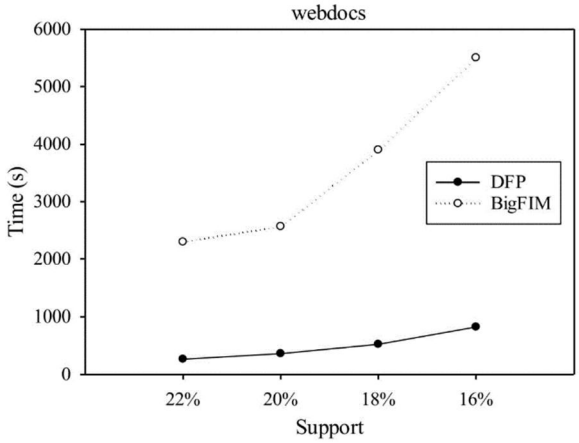


Fig 7. Execution performance of DFP, DistEclat, and BigFIM with support threshold varied from 0.3% to 0.2% on IBM dataset.

Our proposed method, on average, managed to reduce the size of temporary data by 15%. However, this is worth noting that despite this reduction, the ratio of temporarily generated data remains considerable. Addressing this issue is crucial for optimizing execution time, particularly for large databases with substantial temporary data generation.

©IJRASET: All Rights are Reserved

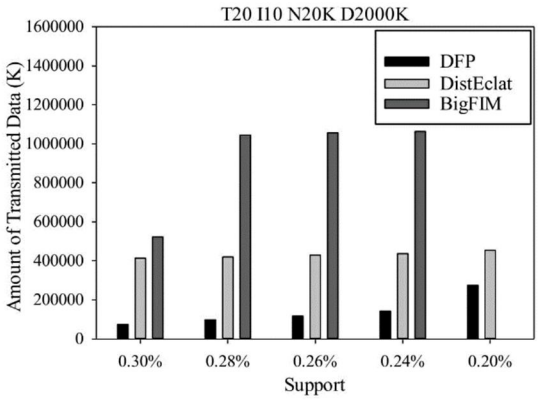


Fig 8. The IBM dataset showed that the amount of transferred data for DFP, DistEclat, and BigFIM with a support threshold ranged from 0.3% to 0.2%..

VII. CONCLUSION

In this study, we came to know that DFP algorithm aimed at enhancing the efficiency of association rule mining through a distributed mechanism. Previous literature has encountered challenges in easily applying distributed mechanisms to database processing, leading to potential increases in transmission costs as communication among clients escalates. Our experiments reveal that proposed DFP algorithm effectively addressed this issue by transferring mining results from the executing client to the server, thereby channeling integrated information through the server rather than dispersing it among clients.

Furthermore, our experiment results demonstrate significant improvements in computing efficiency compared to DistEclat and BigFIM, with the proposed method requiring only 45% and 14.2% of the transmission cost of DistEclat and BigFIM, respectively. The DFP algorithm contributes to enhanced computing efficiency by estimating necessary memory and determining the appropriate number of clients. It addresses challenges such as insufficient memory and repeated scans by estimating the limitations of Transaction IDs (TIDs) for each client and processing remaining TIDs through an extended mechanism called MP, which outperforms DP.

Our experiments underscore the robust performance of the modified mechanism over DP, particularly in the context of big data mining. Moreover, the DFP algorithm offers the capability to automatically determine an optimal number of clients based on various datasets, showcasing its versatility for future applications.

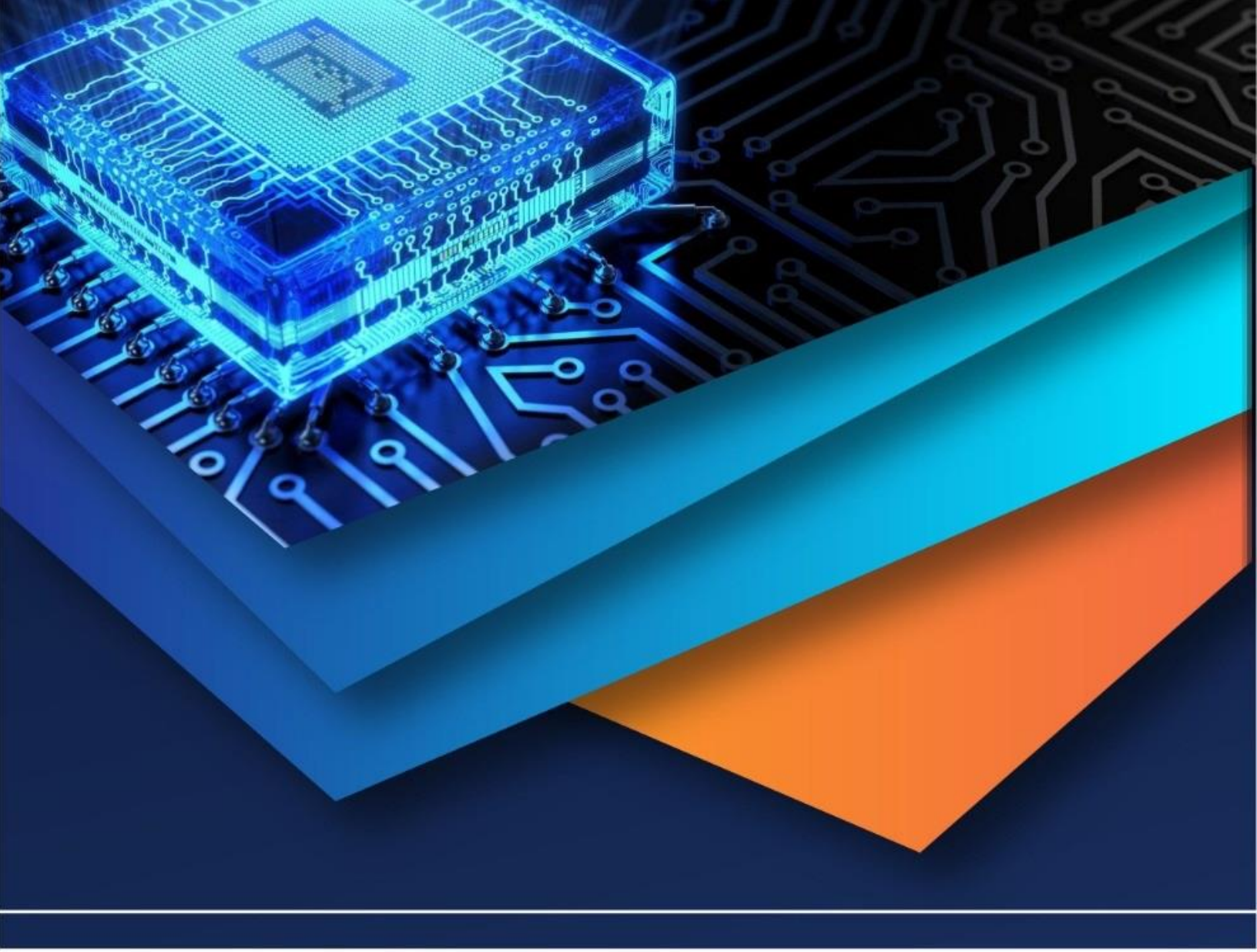
The DFP algorithm holds promise for future advancements. For instance, by distributing mining tasks to clients with highly correlated TIDs, the efficiency of mining can be further improved by reducing the branches of the FP tree. Additionally, efficiently balancing data transmission times and workload across FP trees in each client would enhance mining efficiency, particularly for processing large databases. Given that the size of temporarily generated data inherited from DP remains substantial, future efforts will focus on refining the algorithm to enhance scalability

REFERENCES

[1] Peng-Yu-Huang and Wan -Shu Cheng, “A Distributed Method for Fast Mining Frequent Patterns From Big Data”, Received September 3, 2021, accepted September 17, 2021, date of publication September 24, 2021, date of current version October 8, 2021.
[2] M. J. Zaki. (Oct. 6, 2016). IBM Generator: IBM Synthetic Data Generator for Itemsets and Sequences. Github. Accessed: Jul. 21, 2021. [Online].https://github.com/zakimjz/IBMGenerator
[3] B. Goethals, ‘‘Frequent itemset mining dataset repository,’’ in Proc. FIMI Workshop Committee, Jan. 2019. Accessed: Jul. 21, 2021. [Online]. Avail- able: http://fimi.uantwerpen.be/data/
[4] Z. Cai, X. Zhu, Y. Zheng, D. Liu, and L. Xu, ‘‘A caching-based parallel FP-growth in Apache Spark,’’ in Proc. Int. Conf. Algorithms Archit. Parallel Process., Nov. 2018, pp. 519–533.
[5] Y. Miao, J. Lin, and N. Xu, ‘‘An improved parallel FP-growth algorithm based on spark and its application,’’ in Proc. Chin. Control Conf. (CCC), Jul. 2019, pp. 3793–3797.
[6] Y. Zhang and L. Wang, ‘‘A optimization algorithm for association rule based on spark platform,’’ in Proc. Int. Conf. Comput. Netw., Electron. Autom. (ICCNEA), Sep. 2020, pp. 82–86.
[7] (Mar. 4, 2014). Apache Spark—Unified Analytics Engine for Big Data. Apache Software Foundation. Accessed: Jul. 21, 2021. [Online]. Available: https://spark.apache.org/
[8] D. Quintero, IBM Platform Computing Solutions. Redwood Shores, CA, USA: IBM Redbooks, 2012.
[9] S. Bagui and P. C. Dhar, ‘‘Positive and negative association rule mining inHadoop’s MapReduce environment,’’ J. Big Data, vol. 6, no. 1, pp. 1–16, Dec. 2019.

[10] G. Grahne and J. Zhu, ‘‘Mining frequent itemsets from secondary mem- ory,’’ in Proc. 4th IEEE Int. Conf. Data Mining (ICDM04), Nov. 2004,pp. 91–98.
[11] P. Y. Taser, K. U. Birant, and D. Birant, ‘‘Multitask-based association rulemining,’’ Turkish J. Elect. Eng. Comput. Sci., vol. 28, no 2, pp 933–955,2020.
[12] A. Javed and A. Khokhar, ‘‘Frequent pattern mining on message pass-ing multiprocessor systems,’’ Distrib. Parallel Databases, vol 16, no 3, pp. 321–334, Nov. 2004.
[13] Y. Qiu, Y.-J. Lan, and Q.-S. Xie, ‘‘An improved algorithm of miningfrom FP-tree,’’ in Proc. Int. Conf. Mach. Learn. Cybern., Aug. 2004,pp. 1665–1670.
[14] J. Zhou and K.-M. Yu, ‘‘Tidset-based parallel FP-tree algorithm for the frequent pattern mining problem on PC clusters,’’ in Proc. Int. Conf. Grid Pervas. Comput., May 2008, pp. 18–28.
[15] J. Zhou and K.-M. Yu, ‘‘Balanced tidset-based parallel FP-tree algorithm for the frequent patterns mining on grid system,’’ in Proc. 4th Int. Conf. Semantics, Knowl. Grid, Dec. 2008, pp. 103–108.
[16] K. W. Lin and Y.-C. Luo, ‘‘A fast parallel algorithm for discovering frequent patterns,’’ in Proc. IEEE Int. Conf. Granular Comput., Aug. 2009,pp. 398–403.
[17] R. Agrawal and R. Srikant, ‘‘Fast algorithms for mining association rules in large databases,’’ presented at the 20th Int. Conf. Very Large Data Bases,Sep. 1994.
[18] J.Han, J.Pei, Y.Yin, and R. Mao, ‘‘Mining frequent pattern without candidate generations: A frequent pattern tree approach,’’ Data Mining Knowl. Discovery, vol 8, no 1, pp 53 87, 2004.
[19] R. Agrawal, T. Imieliński, and A. Swami, ‘‘Mining association rulesbetween sets of items in large databases,’’ in Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD), 1993, pp. 207

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)