Journal of Computers, Mechanical and Management
Journal homepage: www.jcmm.co.in
e-ISSN: 3009-075X
AN PUBLISHING

Volume 4 Issue 6
Article Number: 25220

A Multidimensional Evaluation Framework for Parallel Frequent Pattern Mining Algorithms on Big Data Platforms

Baokui Liao[1,2], Mohd Nurul Hafiz Ibrahim[3], Mustafa Muwafak Alobaedy[3], and S. B. Goyal* [3]

[1]City Graduate School, City University, Petaling Jaya, Malaysia 46100
[2]Guizhou Light Industry Technical College, Huaxi University Town, Guiyang, Guizhou, China 550025
[3]Faculty of Information Technology, City University, Petaling Jaya, Malaysia 46100

Abstract

Despite ongoing theoretical refinements in parallel frequent pattern mining algorithms, practical implementations still experience issues such as inefficient resource scheduling, low node interaction efficiency, and limited system robustness. Addressing the lack of comprehensive and systematic testing methodologies for existing algorithms, this paper proposes a data mining algorithm testing framework tailored for big data platforms. The methodology centers on three key dimensions: resource adaptability, communication efficiency, and system robustness, establishing a quantifiable and reproducible experimental evaluation framework. To validate its effectiveness, KVBFP is used as the experimental subject, and three sets of experiments are designed and implemented within a Hadoop cluster environment: algorithm resource adaptability testing, communication frequency testing, and algorithm stability testing. Experimental results show that the first set of experiments accurately measures algorithm resource consumption across different clusters. The second set of experiments shows that KVBFP reduces communication frequency by 51.5% compared to the PFP algorithm. The third set of experiments demonstrates that the algorithm's recovery time remains within 30 seconds under fault conditions. Through comprehensive evaluation of the algorithm via these three experiments, this paper provides a quantitative reference for applying data mining algorithms in real-world scenarios.

Keywords: Hadoop YARN; Parallel FP-Growth; Resource Adaptation; Communication Overhead; Fault Tolerance

1. Introduction

With the rapid advancement of big data technology, the exponential growth of massive datasets has created an increasingly urgent need for scalable and efficient data mining methods [1]. As one of the core tasks in data mining, frequent pattern mining finds extensive applications in recommendation systems, financial analysis, and medical data processing [2]. Traditional algorithms, such as Apriori and FP-Growth, have achieved significant success on small to medium-sized datasets; however, their scalability is constrained in distributed environments, often resulting in performance degradation due to memory constraints, redundant communication, and uneven task distribution [1, 3]. These challenges have driven research and application of parallelized algorithms, with researchers attempting to deploy them on big data platforms like Hadoop and Spark to leverage distributed computing resources for enhanced mining efficiency and scalability.

*Corresponding Author: S. B. Goyal (drsbgoyal@gmail.com)
Received: 28 Sep 2025; **Revised:** 03 Dec 2025; **Accepted:** 11 Dec 2025; **Published:** 31 Dec 2025

DOI: 10.57159/jcmm.4.6.25220.

Despite theoretical and practical advances in parallel frequent pattern mining algorithms, significant shortcomings persist in real-world applications. First, uneven resource utilization prevents clusters from fully leveraging computational power, resulting in inefficient task scheduling and load distribution [4]. Second, frequent node interaction events incur substantial overhead, causing severe network strain in large-scale deployments and limiting system throughput [5]. Third, algorithms lack effective fault-tolerance mechanisms, exhibiting fragility when confronting common industrial-grade issues such as node failures, data corruption, or network interruptions [6]. While existing research has achieved breakthroughs in algorithmic structural optimization and execution efficiency, most evaluations remain confined to single dimensions, such as runtime and accuracy, neglecting critical engineering metrics, including dynamic resource adaptation, communication efficiency, and system stability. This imbalance between theoretical optimization and practical reliability constitutes a critical gap in current research.

To address this issue, this paper proposes a systematic and reproducible evaluation framework for comprehensively examining parallel frequent pattern mining algorithms across three key performance dimensions: resource adaptability, communication efficiency, and robustness. Using the deployment of KVBFP (Key–Value Based FP-Growth) on Hadoop clusters as the primary case study, this framework comprehensively reveals algorithmic behavior in distributed environments while addressing performance dimensions often overlooked in traditional evaluations [7–9]. Unlike prior work, this paper integrates methodological design with empirical validation, delivering quantitative test results through experiments across varying cluster scales, communication loads, and fault conditions. Experimental results demonstrate that this framework not only accurately reflects KVBFP's real-world performance but also establishes a reference paradigm for evaluating the systematic performance of parallel mining algorithms.

1.1. Problem Statement

Although parallel frequent pattern mining algorithms have been extensively studied and applied on distributed platforms such as Hadoop and Spark, existing research remains primarily focused on comparing metrics like runtime and result accuracy, with insufficient attention paid to engineering-level performance factors. Large-scale data platforms often encounter challenges such as uneven resource allocation, excessive node interaction event overhead, and inadequate fault-handling mechanisms [10, 11]. These shortcomings not only limit algorithm scalability but also undermine stability in dynamic heterogeneous clusters. The current lack of a unified, systematic evaluation framework results in incomparable and irreproducible test results across different studies, further hindering the translation of theoretical findings into practical applications. As big data applications grow in scale and complexity, systematically evaluating the performance of algorithms across core dimensions has become a critical barrier to the engineering implementation of parallel frequent pattern mining algorithms. Therefore, constructing a multidimensional, reproducible evaluation system that accurately reflects algorithmic behavior in distributed environments is an urgent priority.

1.2. Contribution

Based on a systematic analysis of the KVBFP algorithm's performance on large-scale data platforms, this paper proposes a performance testing framework for parallel high-frequency pattern mining algorithms, focusing on three critical performance dimensions in big data environments. Through this framework, this paper not only validates the engineering feasibility of KVBFP but also provides unified methodological support for practical verification of parallel algorithms.

1. **Resource Adaptability Testing:** By comparing KVBFP's resource allocation and task scheduling performance across varying cluster scales, results demonstrate its strong dynamic adaptability. It exhibits high compatibility with YARN resource management mechanisms and effectively achieves load balancing.
2. **Node Interaction Efficiency Testing:** Analysis of sampled DataNode log files reveals that KVBFP significantly reduces intra-cluster data exchange frequency while maintaining computational task separation. This provides a distinct communication optimization advantage over traditional PFP algorithms.
3. **Robustness Testing:** Multiple failure scenarios, including node shutdowns, network interruptions, and data corruption, were constructed. Results demonstrate KVBFP's stable performance in task recovery, error handling, and result consistency, showcasing strong fault tolerance and practical value.

2. Literature Review

FP-Growth, one of the most widely used frequent pattern mining algorithms, has garnered significant research attention due to its ability to avoid candidate set generation and enhance mining efficiency. However, in large-scale data environments, FP-Growth exhibits notable bottlenecks in memory consumption, multiple database scans, and task scheduling. Consequently, numerous scholars have explored the integration of FP-Growth with distributed computing platforms to enhance its practicality in real-world big data settings. Ragaventhiran and Kavithadevi (2020) [12] introduced a CAN-tree-assisted Map-Optimize-Reduce framework in Hadoop, leveraging load balancing to enhance scalability and execution speed. However, its stability under node failures and communication anomalies remains insufficiently validated. Similarly, Al-Badani et al. (2025) [13] optimized the FP-Tree structure by constructing an OFIM (Ordered Frequent Itemset Matrix), thereby reducing memory consumption and mining time. While improving structural compactness, their experiments were limited in scale and lacked empirical testing in complex distributed environments.

With the advancement of in-memory computing frameworks, Spark has emerged as a vital platform for parallel frequent pattern mining. Gupta and Sawant (2021) [3] proposed parallel Apriori and FP-Growth algorithms in the Spark environment, significantly reducing runtime in multi-dataset scenarios by leveraging the RDD mechanism. Wang and Jiao (2020) [14] further proposed a partition-based parallel FP-Growth algorithm, which effectively mitigates load imbalance issues in course grade data mining. However, most of these studies focus on optimizing runtime efficiency, lacking quantitative assessments of communication overhead and system robustness.

At the application level, the applicability of FP-Growth has been validated across multiple domains. Yulani et al. (2024) [15] applied the algorithm to retail transaction data, revealing consumer purchasing patterns that inform inventory and marketing strategies. Wahyuningsih et al. (2023) [16] compared Apriori, FP-Growth, and ECLAT on supermarket transaction data, showing that FP-Growth outperforms Apriori in rule mining efficiency but remains slower than ECLAT in execution time. In healthcare scenarios, Ma et al. (2022) [2] deployed parallel FP-Growth for analyzing traditional Chinese medicine constitutions, demonstrating its superior effectiveness over Apriori in processing large-scale clinical data. Zhang (2021) [17] introduced a hash table mechanism to optimize FP-Growth in cloud computing environments, thereby enhancing both mining efficiency and accuracy. While such studies integrate FP-Growth into e-commerce recommendation, traditional Chinese medicine diagnosis, and transaction behavior prediction, they generally lack systematic comparisons across performance dimensions. Table 1 summarizes the limitations of existing research. It reveals three major shortcomings in the evaluation of algorithms within existing data mining research:

1. A lack of in-depth examination of resource adaptability, failing to demonstrate algorithmic performance across diverse clusters systematically.
2. Insufficient analysis of communication overhead, with no quantitative study of node interaction frequency.
3. Robustness validation remains inadequate, failing to cover complex scenarios such as node failures, data corruption, or network disruptions.

Table 1: Comparative summary of testing deficiencies in related works

Author (Year)	**Algorithm**	**Dataset**	**Evaluation Metrics**	**Limitations**	**Relevance to Current Study**
Mahrousa et al. (2021) [18]	FP-Growth	Transactional dataset	Execution efficiency, memory	No fault tolerance	Adds communication & robustness
Senthilkumar & Hari Prasad (2020) [19]	FP-Growth	Large-scale transaction data	Execution time, communication overhead	No robustness validation	Motivates fault tolerance testing
Ragaventhiran & Kavithadevi (2020) [12]	CAN Tree	Retail store, Chess	Load balancing, scalability	Ignored node failures	Highlights node failure handling
Gupta & Sawant (2021) [3]	Apriori, FP-Growth	Multiple datasets	Efficiency, scalability	No communication/fault tolerance	Needs communication overhead analysis
Wang & Jiao (2020) [14]	FP-Growth	Education data	Load balancing, runtime	Robustness not validated	Emphasizes robustness gap
Li et al. (2022) [2]	FP-Growth	TCM clinical records	Precision, recall	No communication/stability	Adds system stability
Zhang (2021) [17]	FP-Growth	Cloud big data	Mining efficiency, accuracy	No load balancing, fault tolerance	Shows resource adaptability need
Wahyuningsih et al. (2023) [16]	Apriori, FP-Growth, ECLAT	Retail data	Execution time, support	Only efficiency tested	Supports multidimensional evaluation
Al-Badani et al. (2025) [13]	FP-Growth, Apriori	Large-scale inventory	Memory, execution speed	Limited to OFIM, no distributed adaptability	Stresses distributed adaptability
Yulani et al. (2024) [15]	FP-Growth	Retail data	Rule efficiency, inventory optimization	No performance/robustness	Calls for robustness testing

3. Methodology

To comprehensively evaluate the actual performance of the KVBFP algorithm on a large-scale data platform, this paper designs and implements three sets of experiments based on the Hadoop platform, which are conducted in three dimensions: adaptive resource allocation, node communication efficiency, and system robustness. The research methodology framework adopted in this paper is illustrated in Figure 1.

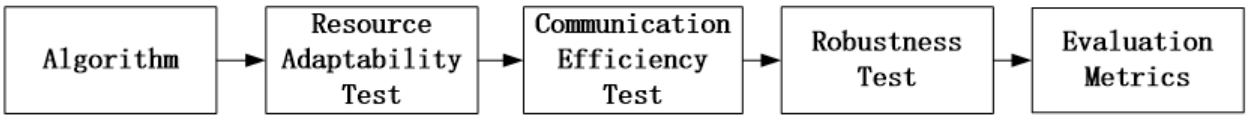

Figure 1: Multi-dimensional evaluation framework

3.1. Adaptive Resource Allocation

Within the Hadoop platform, adaptive resource allocation is implemented by YARN (Yet Another Resource Negotiator). As the core resource management framework for the cluster, YARN dynamically allocates and adjusts computational resources in response to changes in task load. Its internal schedulers ensure fairness among multiple tasks and optimize resource utilization through container allocation and reclamation mechanisms. YARN's adaptive scaling capability enables the cluster to automatically add nodes during periods of high load and reduce nodes during periods of low load, thereby enhancing overall throughput and reducing computational costs.

During the parallel execution of the KVBFP algorithm, as transaction data scales up, the computational capacity of a single node often becomes insufficient. Dynamic cluster node expansion is required to maintain performance. When scaling the cluster from p_1 to p_2 nodes, the system exhibits differences across metrics, including startup latency, task execution time, job completion time, CPU and memory utilization, disk I/O rate, and container allocation count. To quantitatively assess these differences, this study defines the resource utilization function in Eq. (1):

$$U_{\text{res}} = \frac{\sum_{i=1}^{P} \left(\frac{1}{n} \sum_{k=1}^{n} CPU_{i,k} \Big/ \frac{1}{n} \sum_{k=1}^{n} MEM_{i,k} \right)}{P} \tag{1}$$

where $CPU_{i,k}$ denotes the CPU utilization of node i at the kth sampling point; n denotes the total number of samples taken over the entire task execution time; $MEM_{i,k}$ denotes the memory usage of node i during the kth sampling instance; and P denotes the number of nodes in the cluster. To ensure reproducibility of experiments, the pseudocode for the adaptive resource allocation experimental workflow designed in this paper is as follows:

Adaptive Resource Allocation Pseudocode

Algorithm 1 Adaptive Resource Allocation Procedure

Require: Dataset D, initial cluster size p_1, extended cluster size p_2
Ensure: Resource utilization metrics U_{res}
1: Initialize Hadoop cluster with p_1 nodes
2: Deploy YARN with FairScheduler enabled
3: Submit the KVBFP job on dataset D
4: Record baseline metrics: CPU, MEM
5: Dynamically scale cluster to p_2 nodes
6: YARN reallocates containers across p_2 nodes
7: Re-run the KVBFP job on D
8: Collect metrics: CPU, MEM
9: Compute U_{res} for both cluster sizes
10: Compute $\Delta U_{\text{res}} = U_{\text{res}}(p_2) - U_{\text{res}}(p_1)$

3.2. Cluster Interaction Frequency

During parallel mining, communication efficiency between nodes is a crucial factor that affects system performance. Excessive interaction frequency increases network bandwidth consumption, thereby reducing overall throughput. This study systematically evaluates the differences in interaction frequency and communication overhead between nodes. It does this through comparative experiments using the KVBFP and PFP algorithms on a Hadoop cluster. The findings highlight the communication efficiency of both algorithms in distributed environments. The experimental methodology comprises four steps: log collection, log parsing, data storage and statistics, and result visualization. Cluster operation logs were gathered from the NameNode, DataNode, and YARN components to capture interaction events, including task scheduling, HDFS read/write operations, and MapReduce execution. Log files were parsed to extract node interaction event details, including source node i and destination node j. This information was stored in HBase for subsequent querying and analysis. Finally, MapReduce jobs aggregated node interaction events to calculate the number of interactions and data volume per unit time between nodes. Communication interaction frequency can be defined as Eq. (2):

$$F_{\text{comm}} = \sum_{i=1}^{p} \sum_{j=1}^{p} N_{ij} \tag{2}$$

Here, N_{ij} denotes the number of interactions between node i and node j, and p represents the number of nodes in the cluster. The detailed code steps for the cluster interaction frequency test are as follows:

Cluster Interaction Frequency Pseudocode

Algorithm 2 Cluster Interaction Frequency Evaluation Procedure

Require: Hadoop cluster logs (NameNode, DataNode, YARN, MapReduce)
Ensure: Communication metrics $\{F_{\text{comm}}\}$
Collect log files from all cluster nodes
Parse logs to extract records $\langle source, target, timestamp, type, data\_size \rangle$
Store extracted records in HBase
Run MapReduce job:
for each node pair (i, j) **do**
    Count N_{ij} (node interaction event count)
    Sum D_{ij} (data volume)
end for
Compute $F_{\text{comm}} = \sum_i \sum_j N_{ij}$
Output metrics and visualize results

3.3. Robustness and Error Handling

This study tested the robustness of the KVBFP algorithm against three common types of failures: node failures, data corruption, and network interruptions. By actively injecting errors and monitoring system performance, we observed the algorithm's behavior in terms of recovery speed, task completion rates, and consistency of mining results. This process not only validates KVBFP's stability in dynamic environments but also provides insights for the engineering application of parallel mining algorithms. The robustness testing workflow is illustrated in Figure 2. The overall process comprises four stages: fault injection, task execution, status monitoring, and result analysis.

During the fault injection phase, the system simulates various abnormal scenarios: randomly shutting down some DataNodes, inserting erroneous records into transaction data, and artificially reducing network bandwidth. In the task execution phase, KVBFP jobs continue running to observe the algorithm's fault tolerance. Subsequently, the monitoring module records task status, node logs, and recovery latency. Finally, the result analysis phase compares performance differences under abnormal versus normal conditions. The pseudocode for the entire robustness testing process is as follows:

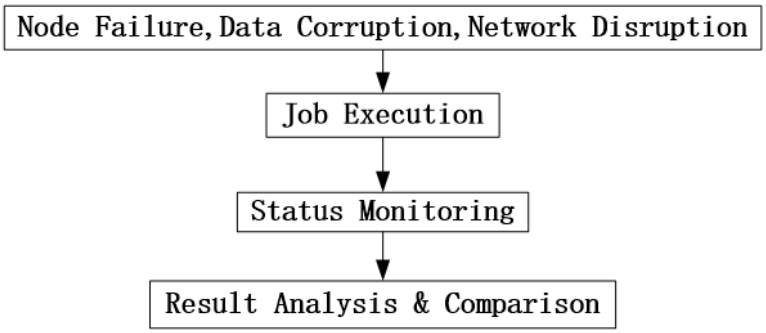

Figure 2: Robustness testing workflow

Algorithm 3 Robustness Evaluation Procedure

Require: Dataset D, cluster size p
Ensure: observed robustness behaviors under different failures

1: Initialize Hadoop cluster with p nodes
2: Select error type:
   - Node failure → randomly shut down nodes
   - Data corruption → inject invalid or missing records
   - Network disruption → reduce bandwidth / add latency
3: Submit the KVBFP job on dataset D
4: Monitor execution:
   - Task completion status
   - Node/container logs
   - Recovery time
5: Compare observed performance against baseline run
6: Report behaviors: stable/degraded / failed

Robustness Evaluation Pseudocode

3.4. Statistical Methods for Experimental Results

To enhance the reliability and reproducibility of experimental results, this study conducted at least five independent replicate runs for each experimental group. All experiments were conducted under identical hardware and software conditions. To ensure the accuracy of experimental data, the mean value of five tests was calculated for each dataset, along with a reasonable range of fluctuation.

3.5. Definition of Normalized Performance Metrics

To ensure consistency across all experimental metrics, this study normalizes all performance metrics using unified quantitative formulas. The definitions of each metric are shown in Table 2, wherein **S:** Total size of the dataset actually processed by the algorithm. **R:** Total runtime required to complete one full execution of the algorithm. **T:** Effective execution time window used for throughput or event-rate calculation. **p:** Number of active DataNodes participating in the parallel computation.

Table 2: Performance metrics definitions

Metric Name	Symbol	Unit	Formula	Description
Inter-node interaction frequency	F_{comm}	events	Eq. (2)	Number of node interaction events per second
Communication throughput	$C_{\text{throughput}}$	MB/s	S/T	Effective data-processing bandwidth
Runtime per GB	R_{perGB}	s/GB	$R/(S/1024)$	Runtime cost normalized by dataset size
Cluster-level throughput	C_{cluster}	GB/s	$(S/1024)/(T \cdot p)$	Average processing efficiency per DataNode
Normalized CPU utilization	U_{CPU}	%	Eq. (1)	Cluster-level CPU utilization
Normalized memory utilization	U_{MEM}	%	Eq. (1)	Cluster-level memory utilization

4. Experiment

4.1. Dataset

The experimental dataset used in this study comes from the public benchmark data resources provided by the Kaggle platform, and the name of the selected dataset is "Characteristics that Favor Freq-Itemset Algorithms" [20]. The dataset has a file size of approximately 815 MB, is stored in plain text format, and its data structure simulates the typical characteristics of large-scale transactional data found in real-world application scenarios, making it both representative and generalizable. This dataset is employed to evaluate the performance of frequent itemset mining algorithms and has been extensively utilized in algorithmic performance analysis. The data mining results demonstrate both correctness and reproducibility; thus, selecting this dataset provides a reliable experimental foundation for this research. The characteristics of the experimental data are presented in Table 3.

Table 3: Dataset characteristics

Characteristics	Value
Transaction count	5,000,000
Number of items	50,000
Max transaction length	5–100
Frequent set density	0.1–0.8
Data file size	854.71 MB

4.2. Experiment Platform

The hardware devices used in this study include six PCs: one PC serving as the NameNode and five PCs as DataNodes, all of which are connected through a switch in a star topology, as shown in Figure 3. The hardware configuration and software information used in the experiment are shown in Table 4. Hadoop is deployed in fully distributed mode, and all software components use stable versions. The experimental data mainly come from YARN, and the YARN ResourceManager Web UI is used to record and display cluster indicators.

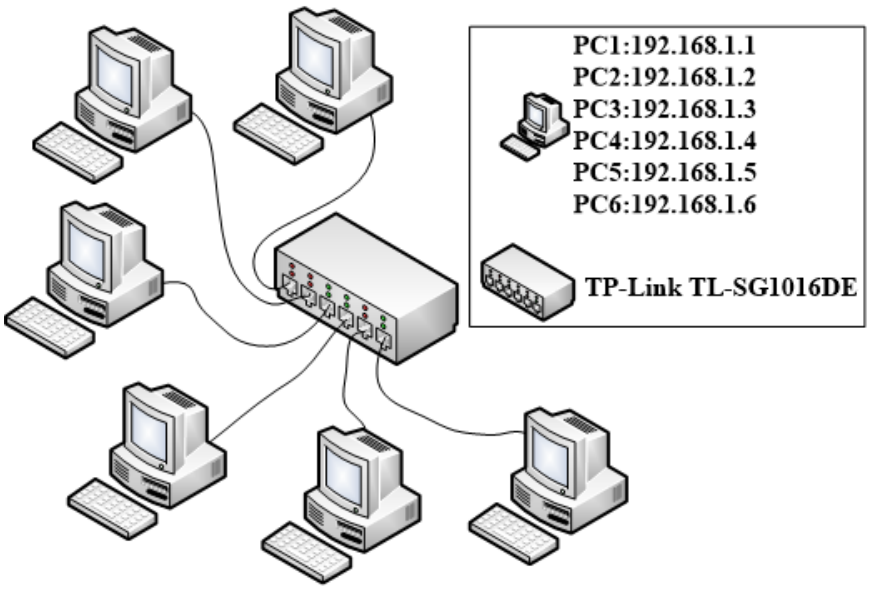

Figure 3: Cluster topology

Table 4: Experimental platform configuration and justification

Category	**Specification**
Hardware	CPU: Intel Core i7-12700F (12 cores)
	RAM: 32 GB DDR4
	Disk: 1 TB SSD
	Network: Gigabit NIC
	Switch: TP-Link TL-SG1016DE
Software	CentOS 7.9
	OpenJDK 1.8
	Apache Hadoop 3.3.1
	YARN
	HDFS

4.3. Experiment 1: Resource Adaptation Test

After configuring the cluster according to the above configuration, the KVBFP algorithm is run on the cluster by starting 1–5 DataNodes respectively for continuous mining of the experimental data, and the experimental data are selected from the first 5×10^4 records in the dataset. The experiments are divided into five groups, and the DataNodes involved in each group are in increasing order. The support threshold is defined as 55. At the end of each group of experiments, the indicators of cluster operation are recorded. The experimental results are shown in Table 5.

Table 5: Resource allocation under different node numbers

DataNode Number	**1**	**2**	**3**	**4**	**5**
Starting (s)	15	12	11	10	10
Execution (min)	7.03 ± 0.21	5.08 ± 0.18	4.13 ± 0.16	3.18 ± 0.13	2.51 ± 0.12
Finish (min)	7.28 ± 0.25	5.24 ± 0.19	4.22 ± 0.14	3.19 ± 0.11	2.68 ± 0.10
CPU (%)	99.1 ± 0.6	82.4 ± 1.8	74.8 ± 1.6	65.2 ± 1.4	56.4 ± 1.2
Memory (%)	62.4 ± 1.3	57.1 ± 1.1	52.3 ± 1.0	44.8 ± 1.2	41.0 ± 1.1
Disk (%)	51.7 ± 2.0	46.5 ± 1.8	40.8 ± 1.7	35.1 ± 1.5	27.9 ± 1.3
Failure & retry	2	1	1	1	0
Container	2 big	1–2 medium	2 small	2–3 small	2–3 small
Node interaction count	0	32 ± 3	87 ± 4	158 ± 6	271 ± 7

The three most critical metrics in Table 5 are execution time, CPU utilization, and memory utilization. Figure 4 illustrates the trends of these three metrics as the number of DataNodes varies; error bars in the figures represent 95% confidence intervals.

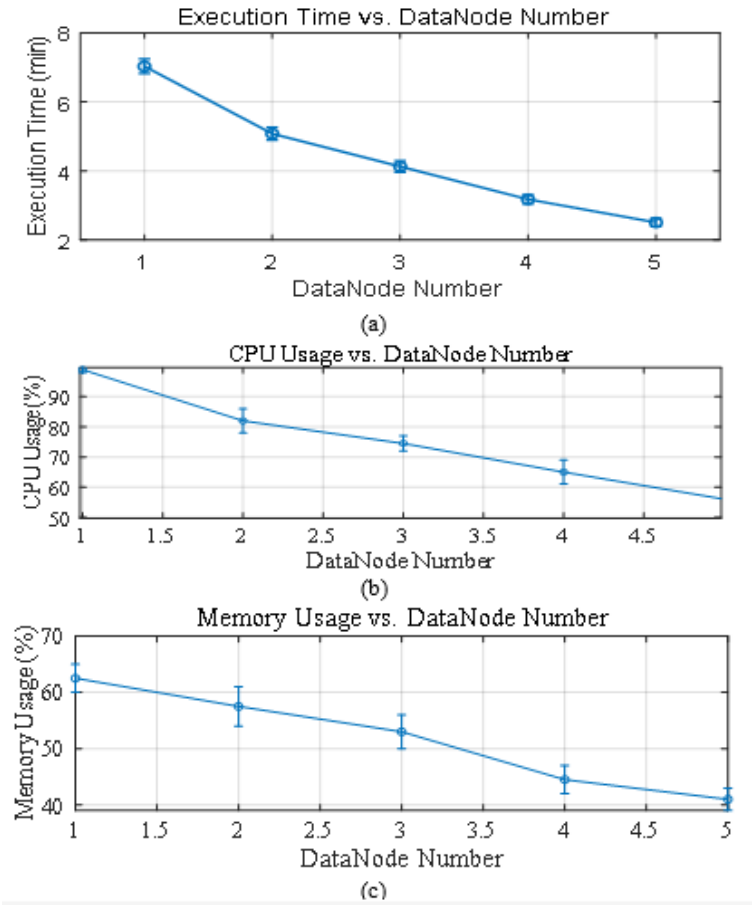

Figure 4: Execution time, CPU usage, and memory usage under varying DataNode numbers

Experimental results demonstrate that YARN dynamically and efficiently allocates computational resources as the cluster scales. With increased participation of DataNodes in computations, execution time, CPU usage, and memory usage all exhibit a marked downward trend. Inter-node usage differences remain consistently controlled within 10%, reflecting balanced resource allocation.

As the cluster expands, the number of Hadoop containers increases while the average container size decreases, further validating YARN's fine-grained adaptive resource scheduling capabilities. However, the increased scale introduces additional communication overhead, causing the frequency of node interaction events to rise with the number of DataNodes. Enhanced multi-node coordination improves overall system stability, resulting in a downward trend in task failures and retry counts. Detailed quantitative analysis results are presented in Table 6.

The experimental results demonstrate that the KVBFP algorithm is well-suited for the Hadoop platform and effectively utilizes its resources. As the number of cluster nodes increases, work tasks can be evenly distributed to the new worker nodes, achieving load balancing of both computing and storage tasks. The entire process does not require excessive human intervention; the cluster automatically completes adaptive allocation of resources. The KVBFP algorithm has good compatibility with Hadoop's resource adaptive framework. By gradually changing the number of nodes, the data mining algorithm is stress-tested, enabling a comprehensive assessment of its adaptability to the platform. The experiment results can then be used as an evaluation index to determine whether the algorithm is compatible with the platform.

Table 6: Analytical summary of resource adaptation experiments

Metric	**Change Trend**	**Key Values**	**Statistical Notes / Analytical Conclusion**
System startup time	Decreases with increasing nodes	15 → 10 s	Avg. variance < 5%; multi-node parallelism reduces initialization delay
Execution time	Significant decrease	7 → 2.5 min	Small fluctuation indicates good scalability and efficient task allocation
Task completion time	Noticeably shortened	7.25 → 2.67 min	Consistent with execution time trends, cluster scaling improves throughput
CPU utilization	Continuously decreases	98–100% → 52–60%	Approx. 40% reduction; workload becomes more balanced
Memory utilization	Steadily decreases	60–65% → 39–43%	Fluctuation within ±3%; improved resource allocation
Disk utilization	Gradual decrease	52% → 28%	Linear decline; storage pressure alleviated with more nodes
Failures and retries	Markedly reduced	2 → 0	Consistent across runs; system robustness significantly improved
Container number and scale	More containers, smaller scale	2 large → 3 small	Matches YARN scheduling; fine-grained allocation improves execution
Node interaction event count	Sharp increase	30 → 274	Linearly correlated with node count; higher communication cost

4.4. Experiment 2: Cluster Interaction Frequency Test

This experiment compares the interaction frequency of the KVBFP algorithm with PFP [14]. Flume was employed to collect logs from each DataNode. From the collected logs, the source and target nodes for each log access were extracted, alongside the program's runtime timestamps. The extracted data were stored in MySQL for querying purposes. The table structure design and partial data are shown in Table 7.

It displays partial access information extracted from the logs. To calculate the number of accesses between DataNode2 and DataNode3, the following SQL query is used:

```
SELECT *
FROM tablename
WHERE Node IN ('Node2', 'Node3')
AND VisitingNode IN ('Node2', 'Node3');
```

Table 7: MySQL node interaction log element extraction

Node	Visiting Node	Start Time	Stop Time
Node2	Node3	T1	T2
Node3	Node2	T1	T2
⋮	⋮	⋮	⋮

This query filters all log entries where either node is the source or target, thereby capturing the mutual interaction events occurring between DataNode2 and DataNode3. The first 1×10^6 records of the dataset were selected, and the start point of time collection was set at 15 seconds after the program reached stable operation. The duration of the collection period was 10 seconds. The support threshold was defined as 55. After the task was completed, logs from each DataNode were obtained, and the interaction frequencies between Node1 and Node5 were queried and counted, then plotted in the interaction frequency statistics shown in Table 8.

Table 8: Comparison of node interaction event frequencies between PFP and KVBFP

PFP / KVBFP	Node Interaction	Summary Stats
Node1–Node1	0 / 0, 17.3 / 12.2, 25.5 / 19, 46.7 / 19.3, 33.1 / 14.3	Total Interaction: 691.9 / 282.2
Node2–Node2	26.8 / 18, 0 / 0, 57.2 / 16.8, 15.2 / 13.6, 41.2 / 9.9	Average Interaction: 27.7 / 11.3
Node3–Node3	16.2 / 15.5, 21.2 / 8.6, 0 / 0, 26.2 / 11.6, 52.7 / 9	Median Interaction: 26.8 / 13.6
Node4–Node4	52.1 / 6, 38 / 14.3, 43.7 / 6.1, 0 / 0, 47 / 13.6	Variance: 328.75 / 49.43
Node5–Node5	25.2 / 20.6, 32.1 / 15.8, 36.5 / 20.9, 38 / 17.1, 0 / 0	Standard Deviation: 18.13 / 7.03

To visualize the difference between the two algorithms in terms of node interaction frequency more intuitively, a heatmap of inter-node interactions can be generated using Matplotlib, as shown in Figure 5.

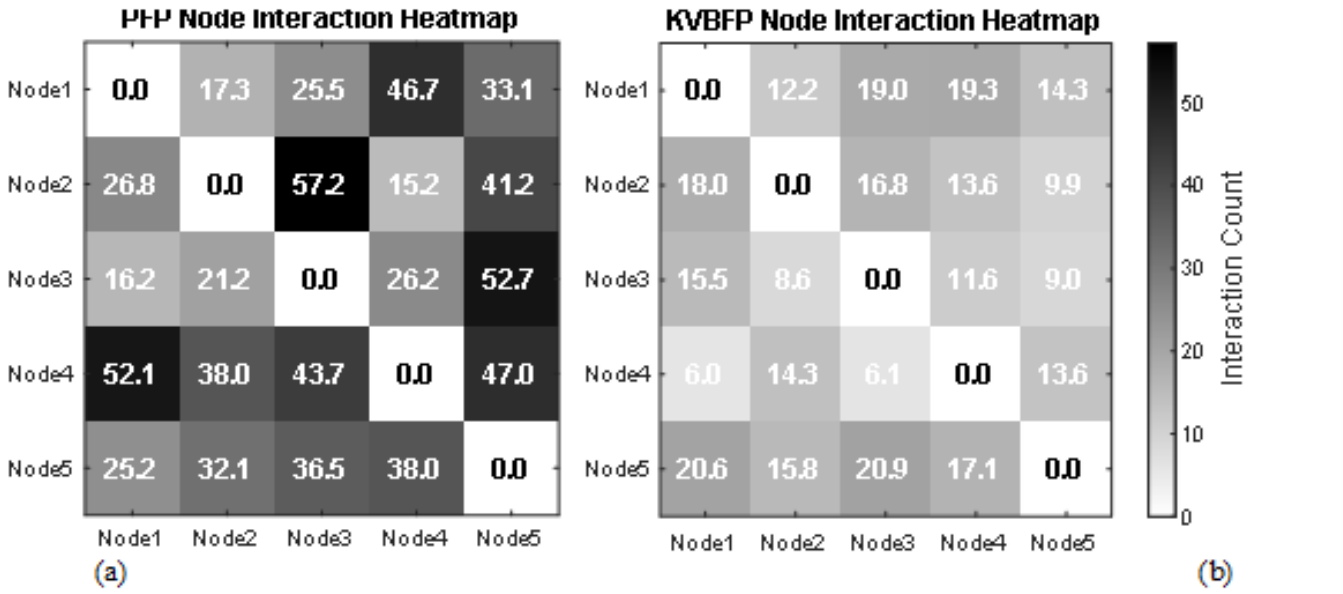

Figure 5: Node interaction heatmap of PFP and KVBFP

As illustrated in Figure 5, PFP exhibits a higher number of interactions compared to KVBFP, with KVBFP demonstrating a distinct advantage in communication efficiency. The mean and median values for both algorithms show minimal disparity, indicating neither suffers from severe single-node overload issues. The variance and standard deviation of PFP are markedly higher than those of KVBFP, indicating greater volatility and poorer communication stability. KVBFP exhibits a more concentrated and stable interaction distribution, reflecting its superiority in communication load balancing and adaptability to distributed computing platforms.

4.5. Experiment 3: Algorithm Fault Tolerance and Robustness Test

To verify the stability and fault tolerance of the KVBFP algorithm during operation, Experiment 3 is designed to test the robustness and error-handling ability of the system when processing large-scale data. The experiment is divided into three groups. The first group artificially fails a node in the cluster to observe the system's operation. The second group artificially destroys the data slices in HDFS to observe recovery from redundant backups. The third group artificially creates a network failure to observe system behavior after disconnection and reconnection. Finally, the number of frequent itemsets extracted for these three fault types is compared against the complete set of frequent itemsets. Four metrics were calculated—Jaccard similarity, overlap ratio, precision, and recall—to assess the extent to which faults impacted the final results. The experimental data selected the first 5,000 transactions from the data file for testing, with the support threshold set to 55.

Node Failure

In this test, cluster nodes are artificially made to stop running unexpectedly during task execution. The failure occurs halfway through the task (at 50% progress). In theory, the system should quickly detect node failure, reassign tasks to the node, and continue executing the remaining tasks. The system detected the anomaly within 2 seconds, completed task redistribution within 5 seconds, and ultimately restored to a stable state within 22 seconds. Affected by the sudden resource depletion, CPU utilization temporarily rose to 66%, but decreased to 52% after resource rebalancing. The task was ultimately completed with a 34-second delay, while ensuring task continuity and consistency.

Data Segmentation Corruption

This test intentionally corrupts a data slice during execution and observes various system metrics as the data is restored using HDFS's redundant backup mechanism. The failure occurs when the task reaches 70% of its execution progress. Theoretically, the system should detect the data corruption and restore data through redundancy or backup mechanisms to continue execution. The system detected the decline in data integrity within 5 seconds after failure injection. Leveraging the HDFS redundancy mechanism, it completed data recovery within 17 seconds, requiring only a single task retry to resume execution. The final task took 23 seconds longer than under normal conditions but maintained 100% data accuracy, demonstrating excellent compatibility between KVBFP and the HDFS backup mechanism.

Network Interruption

In this test, the network is artificially interrupted to simulate a communication interruption during task execution. The failure introduction time is set when task progress reaches 60%. Theoretically, the system should maintain data consistency between nodes and continue execution after network recovery. The system detected communication anomalies within 3 seconds and restarted tasks after network recovery within 14 seconds, ensuring data consistency among nodes. The network outage caused latency to rise to 1128 ms, ultimately extending task completion by 43 seconds. However, overall correctness and consistency were maintained, validating KVBFP's stability under network fluctuations.

Testing Result

Key indicators collected during the experiment are shown in Table 9.

Table 9: Consolidated results of fault-tolerance experiments

Failure Type	Fault Detection Time (s)	Recovery Time (s)	Task Retry Number	Completion Time (s)	Data Integrity / Consistency
Node Failure	2	22.6 ± 1.7	2	186.4 ± 5.1	–
Data Corruption	5	16.8 ± 1.2	1	148.2 ± 4.3	100%
Network Interruption	3	14.3 ± 1.1	2	226.7 ± 6.4	Held together

The results demonstrate that the KVBFP algorithm maintains task correctness and stability through task redistribution and redundancy recovery mechanisms across three common failure scenarios: node failure, data corruption, and network disruption. These faults represent common issues encountered in practical applications, and KVBFP successfully passed all three tests. This testing methodology can also be applied to assess the robustness of other parallel data mining algorithms, with the test results serving as a benchmark for evaluating algorithmic stability.

Jaccard Similarity, Overlap Ratio, Precision, and Recall Analysis

To quantitatively evaluate the consistency of mining results under fault conditions, four commonly used similarity metrics are employed: Jaccard similarity, overlap ratio, precision, and recall. Let F_b denote the frequent itemset collection obtained under the baseline (fault-free) condition, and F_f denote the corresponding collection obtained under fault-injection conditions. The test results for these four indicators are shown in Table 10.

Jaccard similarity is defined in Eq. (3):

$$J = \frac{|F_b \cap F_f|}{|F_b \cup F_f|} \tag{3}$$

Overlap ratio is defined in Eq. (4):

$$O = \frac{|F_b \cap F_f|}{\min(|F_b|, |F_f|)} \tag{4}$$

Precision is defined in Eq. (5):

$$P = \frac{|F_b \cap F_f|}{|F_f|} \tag{5}$$

Recall is defined in Eq. (6):

$$R = \frac{|F_b \cap F_f|}{|F_b|} \tag{6}$$

Table 10: Quantitative similarity between baseline and fault-injection mining results

Fault Scenario	Jaccard Similarity	Overlap Ratio	Precision	Recall
Node Failure	0.972 ± 0.011	0.984 ± 0.008	0.991 ± 0.006	0.974 ± 0.010
Data Corruption	0.985 ± 0.007	0.997 ± 0.004	1.000 ± 0.000	0.986 ± 0.006
Network Failure	0.968 ± 0.013	0.979 ± 0.009	0.989 ± 0.007	0.971 ± 0.012

As shown in Table 10, the vast majority of frequent itemsets were preserved during scenarios of node failure, data corruption, and network failure, demonstrating that the algorithm maintains stable performance across multiple fault conditions.

4.6. Discussion

The experimental results across the three evaluation dimensions—resource adaptability, communication efficiency, and robustness—demonstrate that the proposed multidimensional evaluation framework captures algorithmic behavior that is not revealed by conventional runtime- or accuracy-centric assessments. More importantly, the results provide insight into why KVBFP exhibits superior engineering performance in distributed environments, rather than merely confirming the existence of performance gains.

From the perspective of resource adaptability, the observed monotonic reduction in execution time, CPU utilization, and memory usage with increasing DataNode count indicates that KVBFP aligns effectively with YARN's container-based scheduling model. As shown in the resource adaptation experiments, workload distribution becomes increasingly balanced as the cluster scales, with inter-node resource utilization differences remaining within a narrow range. This behavior can be attributed to the key–value–oriented decomposition strategy of KVBFP, which enables fine-grained task partitioning and allows YARN to allocate smaller, more numerous containers dynamically. Unlike traditional FP-Growth variants that rely on coarse-grained task assignments, KVBFP avoids persistent resource contention on individual nodes, thereby improving overall cluster-level efficiency. These findings extend prior studies that focused primarily on load balancing or execution speed by demonstrating that dynamic resource adaptability can be quantitatively characterized and validated in a reproducible manner.

The communication efficiency analysis further highlights a critical trade-off inherent in distributed frequent pattern mining. While node interaction frequency inevitably increases as cluster size grows, the results show that KVBFP significantly constrains this growth compared to the PFP algorithm, achieving a reduction of more than 50% in total interaction events. This improvement is not incidental; it reflects structural differences in how intermediate mining results are generated and exchanged. By aggregating local patterns more effectively before shuffle phases, KVBFP reduces redundant inter-node data transfers and mitigates network congestion. In contrast, PFP exhibits higher variance and volatility in interaction frequency, indicating less stable communication behavior and greater sensitivity to cluster topology. These results provide empirical support for the argument that communication overhead should be treated as a first-class evaluation metric, particularly for large-scale deployments where network bandwidth becomes a dominant bottleneck. Robustness testing under node failures, data corruption, and network interruptions demonstrates that KVBFP maintains both operational continuity and result consistency in the presence of realistic fault scenarios. The relatively short detection and recovery times observed across all fault types confirm that the algorithm integrates well with Hadoop's fault-handling mechanisms, including task reallocation and HDFS redundancy. More importantly, the high Jaccard similarity, precision, and recall values between baseline and fault-injection outputs indicate that robustness extends beyond task completion to the semantic correctness of the mined frequent itemsets. This distinguishes the proposed framework from prior evaluations that equate fault tolerance solely with job recovery, without verifying the integrity of analytical results. By explicitly quantifying output consistency, this study demonstrates that robustness evaluation can and should incorporate both system-level and data-level metrics.

Taken together, these findings underscore the necessity of multidimensional evaluation for parallel data mining algorithms deployed on big data platforms. While many existing studies report improvements in runtime or scalability, they often overlook the interconnected effects of resource allocation, communication behavior, and fault tolerance. The proposed framework addresses this gap by providing a systematic methodology that reveals trade-offs among these dimensions, enabling more informed decisions regarding algorithm selection, cluster sizing, and deployment strategies in real-world applications. Despite these strengths, several limitations must be acknowledged. The experimental evaluation primarily focuses on KVBFP, with PFP serving as the main baseline; therefore, it does not exhaustively represent the diversity of parallel frequent pattern mining algorithms. Additionally, the cluster scale and hardware configuration, while sufficient to expose key performance trends, may not fully capture behavior in very large or heterogeneous production environments. Finally, fault injection in this study is performed manually, which, although effective for controlled experimentation, could be complemented by automated anomaly generation and monitoring mechanisms in future work. Addressing these limitations will further strengthen the generalizability and practical relevance of the proposed evaluation framework.

5. Conclusion

This paper presented a multidimensional evaluation framework for assessing the engineering performance of parallel frequent pattern mining algorithms on big data platforms. Unlike conventional evaluations that emphasize runtime or result accuracy in isolation, the proposed framework systematically examines three critical dimensions—resource adaptability, communication efficiency, and system robustness—thereby providing a more comprehensive and reproducible basis for algorithm assessment in distributed environments. Using KVBFP as a representative case study deployed on a Hadoop cluster, extensive experiments demonstrated that the framework can effectively characterize algorithm behavior under varying cluster scales, communication loads, and fault conditions. The resource adaptability experiments showed that KVBFP integrates well with YARN's dynamic scheduling mechanisms, achieving balanced utilization of CPU, memory, and storage resources as the number of DataNodes increases. Communication efficiency analysis revealed that KVBFP significantly reduces inter-node interaction frequency compared with PFP, indicating lower communication overhead and more stable network behavior. Robustness testing under node failures, data corruption, and network interruptions further confirmed that the algorithm maintains task continuity and output consistency, with recovery times remaining within acceptable bounds and mining results exhibiting high similarity to baseline outputs. Beyond validating the performance of KVBFP, the primary contribution of this work lies in the evaluation methodology itself. The proposed framework offers a unified and quantitative approach for analyzing parallel frequent pattern mining algorithms from an engineering perspective, addressing gaps in reproducibility and multidimensional assessment observed in existing studies. By integrating robustness metrics alongside resource utilization, the framework provides practical guidance for algorithm deployment and system configuration in real-world big data applications. Nevertheless, this study has certain limitations. The experimental evaluation focuses primarily on KVBFP, and the cluster scale is constrained by the available hardware environment. In addition, fault injection is conducted manually, which, while effective for controlled testing, may not capture the full diversity of anomalies encountered in production systems. Future work will extend the framework to a broader range of parallel mining algorithms, explore larger and more heterogeneous cluster environments, and incorporate automated fault-injection and monitoring mechanisms to further enhance evaluation realism and scalability.

Declaration of Competing Interests

The authors declare no known competing financial interests or personal relationships.

Funding Declaration

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data Availability Statement

The data supporting the findings of this study are available from the corresponding author upon reasonable request.

AI Declaration

The authors used an AI-based language tool to improve grammar and readability. The scientific content and conclusions were reviewed and validated by the authors.

Author Contributions

Baokui Liao: Conceptualization, Supervision, Data Analysis, Writing – Review and Editing; **Mohd Nurul Hafiz Ibrahim**: Methodology, Validation, Investigation, Writing – Original Draft; **Mustafa Muwafak Alobaedy**: Software, Visualization, Investigation; **S. B. Goyal**: Supervision, Project Administration, Writing – Review and Editing.

References

[1] C. Liu, "Parallel frequent itemset mining algorithm and optimization based on spark," in *2023 5th International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 2023.

[2] H. Ma, J. Ding, M. Liu, and Y. Liu, "Connections between various disorders: Combination pattern mining using apriori algorithm based on diagnosis information from electronic medical records," *BioMed Research International*, 2022.

[3] P. Gupta and V. Sawant, "A parallel apriori algorithm and fp-growth based on spark," in *ITM Web of Conferences*, 2021.

[4] B. N. Arunakumari, J. M. Suhas, and A. Nair, "Optimizing resource management in hadoop yarn for efficient allocation, utilization, and scheduling," in *International Conference on Computing Communication and Networking Technologies*, 2024.

[5] F. Ullah, G. Srivastava, S. Ullah, K. Yoshigoe, and Y. Zhao, "Nids-vsb: Network intrusion detection system for VANET using spark-based big data optimization and transfer learning," *IEEE Transactions on Consumer Electronics*, 2023.

[6] T. J. Akinbolaji, G. Nzeako, D. Akokodaripon, and A. V. Aderoju, "Proactive monitoring and security in cloud infrastructure: Leveraging tools like prometheus, grafana, and hashicorp vault for robust devops practices," *World Journal of Advanced Engineering Technology and Sciences*, 2024.

[7] A. Hakeem, R. Curtmola, X. Ding, and C. Borcea, "DFPS: A distributed mobile system for free parking assignment," *IEEE Transactions on Mobile Computing*, 2021.

[8] H. Guo and N. Guo, "Research and application of a multidimensional association rules mining algorithm based on hadoop," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2021.

[9] S. Chaturvedi, S. K. Saritha, and A. Chaturvedi, "Spark based parallel frequent pattern rules for social media data analytics," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*, 2023.

[10] A. Janowski, M. Hüsrevoğlu, and M. Renigier-Biłozor, "Sustainable parking space management using machine learning and swarm theory—the spark system," *Applied Sciences*, 2024.

[11] M. Gu, H. Tang, and L. Li, "Exploration on facial image recognition and processing algorithms under hadoop," in *2023 International Conference on Integrated Intelligence and Communication Systems (ICIICS)*, 2023.

[12] J. Ragaventhiran and M. K. K. Devi, "Map-optimize-reduce: Can tree assisted fp-growth algorithm for clusters-based fp mining on hadoop," *Future Generation Computer Systems*, 2020.

[13] A. M. Al-Badani, A. A. Shujaaddeen, and M. M. Aljafare, "Efficient mining of fp-growth algorithm structure and apriori algorithm using ofim for big data," *International Journal of Applied Information Systems*, 2025.

[14] X. Wang and G. Jiao, "Research on association rules of course grades based on parallel fp-growth algorithm," *Journal of Computational Methods in Sciences and Engineering*, 2020.

[15] Yulani, R. Kurniawan, and Y. Wijaya, "Implementasi algoritma fp-growth pada data transaksi penjualan seblak jontor," *JIKA (Jurnal Informatika)*, 2024.

[16] R. Wahyuningsih, A. Suharsono, and N. Iriawan, "Comparison of market basket analysis method using apriori algorithm, frequent pattern growth (FP-Growth) and equivalence class transformation (ECLAT): Case study supermarket 'x' transaction data for 2021," *Business and Finance Journal*, 2023.

[17] B. Zhang, "Optimization of fp-growth algorithm based on cloud computing and computer big data," *International Journal of Systems Assurance Engineering and Management*, 2021.

[18] Z. Mahrousa, D. M. Alchawafa, and H. Kazzaz, "Frequent itemset mining based on development of fp-growth algorithm and use mapreduce technique," *Association of Arab Universities Journal of Engineering Sciences*, vol. 28, no. 1, pp. 83–98, 2021.

[19] A. Senthilkumar and D. Hari Prasad, "An efficient fp-growth based association rule mining algorithm using hadoop mapreduce," *Indian J. Sci. Technol*, vol. 13, no. 34, pp. 3561–3571, 2020.

[20] J. Heaton, "Comparing dataset characteristics that favor the apriori, eclat or fp-growth frequent itemset mining algorithms," 2017. arXiv: Databases.