Journal of Intelligent & Fuzzy Systems 45 (2023) 6161–6177
DOI:10.3233/JIFS-232048
IOS Press

Improving Data Processing Speed on Large Datasets in a Hadoop Multi-node Cluster using Enhanced Apriori Algorithm

M.R. Sundarakumar[a,*], Ravi Sharma[a], S.K. Fathima[b], V. Gokul Rajan[a], J. Dhayanithi[b], M. Marimuthu[b], G. Mohanraj[c], Aditi Sharma[d] and A. Johny Renoald[e]
[a]*School of Computing Science and Engineering, Galgotias University, Greater Noida, Uttar Pradesh, India*
[b]*Department of CSE, Sona College of Technology, Salem, Tamilnadu, India*
[c]*Department of Smart Computing, Vellore Institute of Technology, Vellore, Tamilnadu, India*
[d]*School of Computer Science and Engineering, Parul Institute of Technology, Parul University, Gujarat, India*
[e]*Department of EEE, Erode Sengunthar Engineering College, Perundurai, Tamilnadu, India*

Abstract. For large data, data mining methods were used on a Hadoop-based distributed infrastructure, using map reduction paradigm approaches for rapid data processing. Though data mining approaches are established methodologies, the Apriori algorithm provides a specific strategy for increasing data processing performance in big data analytics by applying map reduction. Apriori property is used to increase the efficiency of level-wise creation of frequent itemsets by minimizing the search area. A frequent itemset's subsets must also be frequent (Apriori property). If an itemset is rarely, then all of its supersets are infrequent as well. We refined the apriori approach by varying the degree of order in locating frequent item sets in large clusters using map reduction programming. Fixed Pass Combined Counting (FPC) and Dynamic Pass Combined Counting (DPC) is a classical algorithm which are used for data processing from the huge datasets but their accuracy is not up to the mark. In this article, updated Apriori algorithms such as multiplied-fixed-pass combined counting (MFPC) and average time-based dynamic combined counting (ATDFC) are used to successfully achieve data processing speed. The proposed approaches are based on traditional Apriori core notions in data mining and will be used in the map-reduce multi-pass phase by ignoring pruning in some passes. The optimized-MFPC and optimized-ATDFC map-reduce framework model algorithms were also presented. The results of the experiments reveal that MFPC and ATDFC are more efficient in terms of execution time than previously outmoded approaches such as Fixed Pass Combined Counting (FPC) and Dynamic Pass Combined Counting (DPC). In a Hadoop multi-node cluster, this paradigm accelerates data processing on big data sets. Previous techniques were stated in terms of reducing execution time by 60–80% through the use of several passes. Because of the omitted trimming operation in data pre-processing, our proposed new approaches will save up to 84–90% of that time.

Keywords: Algorithms, pruning, data mining, hadoop cluster, map reduce

1. Introduction

Data processing from huge repositories of big data analytics is a typical approach to extract accurate information in today's digital environment. Multi-node clusters are used to manage large amounts of data and improve performance on large datasets. Traditional approaches are used to extract information from bigger datasets; however, the accuracy and consumption time have beyond the norm. The apriori approach is well known in data mining techniques and is used to construct common item collections

*Corresponding author. M.R. Sundarakumar, School of Computing Science and Engineering, Galgotias University, Greater Noida, Uttar Pradesh, India. E-mail: sundarakumarmr@gmail.com.

ISSN 1064-1246/$35.00

from large datasets [1] via candidate generation. This approach is employed as part of the Association Rule Mining (ARM) methodology. This is the most successful technique for pre-processing large datasets that adheres to data mining research concepts [2]. Data was collected in a variety of ways before being stored in a data warehouse for processing. Unwanted data must be removed or routinely utilized data sets extracted during various phases, depending on their quality. In big data analytics, data is represented not only by volume but also by format diversity and velocity via speed [3]. Algorithms in datasets must manage or collect data using a variety of ideas and approaches. Because of the administration of huge data sets in data mining from data warehousing, the Apriori approach deals with frequent item set extraction by utilizing single or multiple passes in handling big data. When dealing with massive amounts of big data, however, scalability is weak. Hadoop is a framework that enables parallel data processing on enormous data volumes. Despite this, data mining techniques must be redefined, modified, or optimized when mining datasets using the Map-Reduce architecture [5]. Apriori is used to increase the efficiency of level-wise creation of frequent itemsets by minimizing the search area. A frequent itemset's subsets must also be frequent (Apriori property). If an itemset is rarely, then all of its supersets are infrequent as well. Following the generation of a candidate $(k+1)$-itemset, it is degenerated to its k-itemset subsets. If any of these subsets is not big, i.e., is not a member of Lk, the candidate is dismissed and considered small or uncommon; otherwise, the applicant must pass the second test. In the second pruning process, the support of the candidate itemset supplied by the first pruning is tallied. If this support is more than the minimal support, minsup, the candidate will be considered a frequent itemset.

Hadoop uses the Map Reduce programming paradigm to process enormous amounts of data stored in a distributed file system (DFS). Hadoop Distributed File Systems (HDFS) may now access numerous datasets as user input for a number of purposes. To construct this work, two unique functions, mapper and reducer, were executed on separate computers as equal job execution [6]. When all jobs are finished, they are saved as reducer tasks in HDFS. When the recursive approach is employed, several tasks may be broadcast and processed in parallel across multiple nodes, with the final result stored in HDFS. The functions map and reduce were used to divide the input datasets into a number of divisions in order to develop the a priori method in Map Reduce. The apriori technique is used to produce candidate item sets for frequent item set generation and iterative operations on map-reduce. The mapper and reduction functions were in control of these tasks, with the mapper dealing with local candidates and the reducer merging mapper output counts to produce the outcome of frequent item sets.

The Apriori algorithm relies on the Map Reduce Framework, and its foundation is scheduling and waiting time in a queue. Because the new job must wait for the previous job to finish, numerous passes can be entered as input into the map-reduce model, i.e., the same tasks are conducted in several iterations. As a consequence, a default queue has been constructed, and jobs are handled one at a time with different iterations. As a result, completing all of the chores takes a long time. Single Pass Counting (SPC), Fixed Pass Combined Counting, and Dynamic Pass Combined Counting are three enhanced methodologies for data mining algorithms [7].

These data mining algorithms rely heavily on iterative Map Reduce with fixed values. SPC is utilized for single-pass operations, whereas FPC and DPC are used for initiatives with several passes that incorporate integrated candidate creation and counting. In comparison to SPC, the FPC approach reduces the number of requests by combining a set number of subsequent phases. To balance workloads across phases, DPC dynamically incorporates candidates from multiple successive stages. When we combine candidate generations in a multi-pass technique, the candidate item sets are generated from the prior sets in three successive passes. If the candidate item sets are created independently, the results of the frequent item set will be incorrect. As a result, it is generated using prior candidate sets. Because FPC has a limited number of passes, false-positive candidates always yield inaccurate results. DPC overcomes this problem by employing connected, dynamic, consecutive passes. The challenge with DPC in the map-reduce phase is recognizing the number of passes to combine in a dynamic way. As a result, execution time varies among cluster nodes, as does capacity for large datasets. When working with large data sets, these data mining methodologies raise challenges with the map-reduce methodology in terms of execution time and scheduling modifications. These unique approaches to the problem are

described and evaluated on various clusters by implementing the time necessary to finish many jobs in several runs.

Apriori is a relational database mining and association rule learning algorithm. It then goes on to find the most often occurring individual items in the database and expands them to bigger and larger item sets as long as those item sets appear frequently enough in the database. The concepts shown here are utilized in a map-reduce architecture to lower the time it takes to complete a single operation. In addition to the traditional FPC and DPC algorithms, they include MFPC, ATDFC, optimized-SFPC, and optimized-ATDFC. To address the issue of FPC, DPC created additional algorithms such as variable-size FPC (VFPC) and elapsed-time DPC (ETDPC). It was attempting to reduce the execution and waiting times for individual jobs. However, at various stages of execution, the passes begin with 1, 2, 4, 8, 16,... When it comes to a multi-node cluster, the time required has not dropped significantly. Our unique MFPC algorithm employs the approach of executing numerous passes after combining, where $n = 1, 2, 3,...$ This approach accelerates the development of the number of passes, easily mixes the phases, and provides a low execution and waiting time. ATDPC also leverages the approach of feeding a pre-processed dataset item into Map Reduce and maintaining it constant throughout the operation. because it minimizes the average execution time when several iterations are performed. The optimized versions of MFPC and ATDPC are similar, with the difference being the number of passes as an input for combined phases. The unique approaches are relevant to Hadoop multinode cluster real-time operations on big data collections on Hadoop multinode clusters. In comparison to earlier strategies, this will provide a novel method for extracting data from datasets with varying processing capacities. It will also be suitable for the real-time generation of big data sets with access in different places and a minimum of waiting time. The apriori feature is used to increase the efficiency of level-wise creation of frequent item sets by minimizing the search area. All frequent item set non-empty subsets must be frequent. The Apriori algorithm's core premise is its anti-monotonicity of support measures. Finding frequent item sets from incoming data sets and appropriately arranging them using algorithms on a Hadoop multinode cluster provides accuracy and high latency. The authors' primary work is provided here, in which they compare the proposed algorithms to standard algorithms and differentiate their advances.

- In terms of execution time, the efficient algorithms FPC and DPC improve and generalize MFPC and ATDPC.
- These novel strategies increase apriori algorithm features including speed, size, and throughput.
- FPC and DPC are neither interoperable or flexible, while MFPC and ATDPC are.
- Optimized-MFPC and Optimized-ATDPC are upgraded versions of MFPC and ATDPC that outperform MFPC and ATDPC, respectively.
- The improved versions are particularly effective at mining large-size frequent item sets.
- All of the solutions mentioned have improved scalability and demonstrated good performance among cluster nodes.
- There is no need to wait for a lengthy amount of time due to the number of times candidates with multiplier (2^n) values are formed. Limited values have been evaluated for enhanced performance.
- Data pre-processing is unnecessary for numerical datasets due to the frequent item sets in multiple passes.

Section 2 discusses the Apriori method and the Hadoop map-reduce concepts. Section 3 discusses the literature study and associated work on data mining concepts. Section 4 discusses the proposed architecture and algorithm for our novel approach. Section 5 discusses the results and comments. Finally, Section 6 discusses our new approach's findings and proposed improvements.

2. Literature survey

[1] used a data structure to count the triangular matrix item sets. They employed candidate generation instead of a combiner in the mapper step. It presented an Apriori algorithm-based Hadoop cluster solution for distributing a large volume of datasets across multiple nodes with homogeneous and heterogeneous nodes. With Hadoop clusters, both nodes are handled as a streaming process via the network. Every bit of digital data must be assessed as a 0 or 1 bit, and the node specifics are combined with an IP address to be identified. FPC algorithms have permanent passes that influence the incorrect positive candidate item sets in their initial phases and then yield just a smaller number of candidates. DPC has solved a number of the issues associated with this problem by dynamically mixing passes for load balancing of effort allotted by each phase. Yet, because

of the dynamic passes, threshold values cannot be surpassed. The execution time is determined by the algorithm devised by the researchers. As a result, for large datasets, it returns different results on cluster nodes from various locations.

[2–5] discussed the homogeneous and heterogeneous systems environments, as well as the many parallel and distributed algorithms used to improve the performance of data mining concepts, particularly association rule mining concepts. Yet, it is unsatisfied with aspects of the distributed database environment such as synchronisation, computation, and data partitioning, task scheduling, workload allocation, and monitoring node failure in the cluster. Scalability difficulties are addressed by grid computing, but the execution and waiting times are ineffective when working with data mining principles. But Google's Map Reduce can fix these issues. It is a programming paradigm created by Google for accessing global file systems (GFS). It includes metadata for every network-connected item in the system. Furthermore, because it never shares anything, this file system employs a shared-nothing design. Many data mining techniques have been created by researchers to operate with Map Reduce, but the most common is based on Apriori.

[6] presented FP-Growth and Éclat, which are simple Map-Reduce approaches. Earlier methods were based on the Apriori algorithm's fundamentals and implemented on the Map Reduce environment by combining multiple successive iterations into a single map-reduce operation. Several tasks are allocated for map-reduce operations since the input will be provided in successive iterations. Because the map and reduce functions are distinct, iterations were used to successfully perform the mapping and reducing tasks.

[7] presented Map Reduce methods for mining fuzzy association rules. In a large data collection, it is difficult to retrieve fuzzy sets. This method identifies the association rule of data mining ideas and generates the fuzzy data set using various criteria. Fuzzy sets were generated from massive data sets and will be sent to all network nodes for further processing. [8–10] suggested an algorithm with the characteristics of map-reduce-based algorithms and that employs association rule mining techniques on map-reduce. This type of comparison shed light on the processing of large data sets in big data using data mining methods. It is processed a priori using data pre-processing within cluster nodes. The data from the node is shared across all of the tasks that are executing concurrently.

[11, 12, 29, 30] give parallel FP-Growth algorithm implementations for machine learning fundamentals and data mining libraries. Machine learning techniques have several packages and libraries for accessing large data sets in the map-reduce process; however, data size is a big challenge. When Java is used as a programming language for map reductions, the data processing time is quite slow due to the code built for tiny data sizes. Python may have been used to retrieve large amounts of data using its libraries and packages. [13, 14] presented an independent data model for Apriori to improve node scalability and efficiency by minimising some processes. The Map-Reduce-based apriori algorithms are implemented using a hash table tree, which has been shown to perform better than a hash tree. Utilizing these approaches, hash values may be produced and stored in a database as a hash table. When the metadata extraction process occurs in large data sets, these data models help boost the scalability and speed of the data over the network by utilising the apriori method. It is used in the data warehouse to produce data set items depending on the preceding data set item. This procedure iterates indefinitely and creates data items on a regular basis.

[15, 16] proposed the FiDoop technique on map-reduce for parallel frequent data collection, and it incorporates the FIU-tree (frequent items ultrametric tree) replaced by the Frequent Pattern (FP) tree. FiDoop-HD was introduced and considered for access to multidimensional datasets. Multidimensional data sets are used in high-volume handling applications and can be found in big data. Using the FiDoop method and working for parallel data operations, metrics like speed, latency, and correctness may be measured. For simplicity of access, all tasks in the map-reduce process are controlled by multi-dimensional structure arrays or tree representations. [17, 26–28] proposed the Parallel Randomized Algorithm (PARMA) method for parallel optimization to locate frequent item sets on Map Reduce. This method generates a tiny sample at random from large databases. The data set stored in the Hadoop cluster may access different-sized jobs in parallel as data items. Because map-reduce is used to obtain data from a large data source in parallel, Data mining algorithms are used to extract these using multiple sub-algorithms from the data warehouse.

[18] suggested a method for running a job 100 times faster than Map Reduce. As a result, it necessitates a large amount of memory for calculation; SPARK is a tool used to carry out the aforementioned

operation. SPARK is used to access the data engine and search engine that are both located in the same block.

Because of the SPARK tool, more tasks in map-reduce jobs share the same place. It outlines the enhancement of the CPU using memory analytics. SPARK makes use of apriori. It is used in real-time applications for stream computation in parallel. R is an analytics tool that is used to do data analysis and cleansing using the SPARK tool. It is powered by machine learning methods and methodologies. To efficiently conduct the work on large data sets, a data mining technique is employed for pre-processing principles.

[21, 23] suggested an approach that demands highly rapid access to data. Their primary goal is to provide platforms that improve processing efficiency and access to real-world data in order to promote data streaming. Streaming data in a large data collection is a common activity in real-world applications. It employs a streaming protocol to pull information from data warehouses. Moreover, several data mining methods are used in the big data analytics process to manage large data volumes. This streaming protocol idea employs continuous streaming as data is transferred from one location to another.

[24, 25] the VFPC and ETDPC methods were used to access Sequential Pattern Mining Framework (SPMF) datasets and the Frequent Item Set Mining Dataset Repository Several consecutive tasks have been formed by combining those methods, and the elapsed time and average time of data retrieval have been computed for various minimum suppression levels. The data set obtained from such sources must be accessible, and variances with different database sizes are seen. [31–37] suggested a method for identifying incorrect correlations between dataset item sets. Apriori algorithms, for example, are used in data mining to discover relationships between all of the data available in the network and to preserve that link for database connectivity. Incorrect relationships or duplicate data must be deleted using data mining techniques before it can be accessed by an a priori algorithm for further categorization.

Literature survey summarizes the overall performances of the apriori algorithm features on big data analytics with map reduce framework. Data mining algorithms rely heavily on iterative Map Reduce with fixed values. SPC is utilized for single-pass operations, whereas FPC and DPC are used for initiatives with several passes that incorporate integrated candidate creation and counting. In comparison to SPC, the FPC approach reduces the amount of requests by combining a set number of subsequent phases. To balance workloads across phases, DPC dynamically incorporates candidates from multiple successive stages. When we combine candidate generations in a multi-pass technique, the candidate item sets are generated from the prior sets in three successive passes. If the candidate item sets are created independently, the results of the frequent item set will be incorrect. As a result, it is generated making use of prior candidate sets. Because FPC has a limited number of passes, false-positive candidates always yield inaccurate results. DPC overcomes this problem by employing connected, dynamic consecutive passes. The challenge with DPC in the map-reduce phase is recognizing the amount of passes to combine in a dynamic way. As a result, execution time varies amongst cluster nodes, as does capacity for large datasets. When working with large data sets, these data mining methodologies raise challenges with the map-reduce methodology in terms of execution time and scheduling modifications. These unique approaches to the problem are described and evaluated on various clusters by implementing the time necessary to finish many jobs in several runs.

3. MFPC and ATDPC algorithms

3.1. Overview

In 1994, Agarwal introduced and acknowledged Apriori with Srikant. This a priori method is intended to operate with various datasets. The apriori approach is well known in data mining techniques and is used to construct common item collections from large datasets via candidate generation. This approach is employed as part of the Association Rule Mining (ARM) methodology. This is the most successful technique for pre-processing large datasets that adheres to data mining research concepts. Data was collected in a variety of ways before being stored in a data warehouse for processing. Unwanted data must be removed or routinely utilized data sets extracted during various phases, depending on their quality. In big data analytics, data is represented not only by volume but also by format diversity and velocity via speed. Algorithms in datasets must manage or collect data using a variety of ideas and approaches. Because of the administration of huge data sets in data mining from data warehousing, the Apriori approach deals with frequent item set extraction by utilizing sin-

gle or multiple passes in handling big data. Apriori property is used to increase the efficiency of level-wise creation of frequent itemsets by minimizing the search area. A frequent itemset's subsets must also be frequent (Apriori property). If an itemset is rarely, then all of its supersets are infrequent as well. Following the generation of a candidate $(k+1)$-itemset, it is degenerated to its k-itemset subsets. If any of these subsets is not big, i.e., is not a member of Lk, the candidate is dismissed and considered small or uncommon; otherwise, the applicant must pass the second test. In the second pruning process, the support of the candidate itemset supplied by the first pruning is tallied. If this support is more than the minimal support, minsup, the candidate will be considered a frequent itemset. If no extensions are provided, this will end immediately. Because the name of the algorithm suggests Apriori, the frequent item set attribute of prior knowledge is employed by itself. k-frequent item sets may be used to locate $k+1$ item sets using level-wise search. The apriori method discovers all item sets that have supports that are greater than or equal to the minimal suppression value. Several passes are supported by the Apriori algorithm. Individual items are tallied, and frequent things are determined based on the minimal suppression value. A new frequent item set called the candidate item set is generated from the preceding item set before the real count begins. At the end, all potential frequent item sets and next-pass item sets were gathered based on the initial minimum suppression value. An iterative a priori method is used to switch jobs, such as by generating frequent item sets as candidates from earlier data sets and then evaluating the dataset to support the number of candidates over each iteration. The candidate item sets are formed from prior sets in the fifth iteration by combining candidate generations in a multi-pass as a single phase.

3.2. Handling big dataset

In big data, all large files are kept in the Distributed File System (DFS). A distributed file system (DFS) allows enterprises to handle huge data access over numerous clusters or nodes, allowing them to read big data efficiently and do multiple concurrent reads and writes. To manage vast amounts of data, a new inventive way is developed, which necessitates the adoption of an architecture that supports Shared Nothing Architecture (SDA). A framework is a term used to describe such an approach. As a result, data may be stored horizontally on multiple nodes of a cluster and accessible within predefined time frames. Hadoop Distributed File System is based on Google File System (GFS) features. Being an open-source project, it is deployed as commodity hardware at a minimal cost. It breaks the files into a number of parts using various blocks.

The default block size of Hadoop's new file system is 128 MB. The replication factor is set to 3 by default and may be modified depending on the number of systems in the cluster [8]. Map The Reduce programming approach is used to provide parallel and distributed database processing for large-scale databases in big data. MapReduce is a programming approach for developing programs that can handle large amounts of data in parallel across several nodes. MapReduce allows for the analysis of massive amounts of complicated data. It distributes the whole work into a number of independent blocks based on the input file size among a larger number of nodes. The Map-Reduce Framework, which consists of the mapper, combiner, and reducer phases, will be used for large volume dataset extraction optimization. This will run in parallel on all nodes in a cluster from various places. The input and output of these functions are denoted as key-value pairs (k, v). In our implementation work, the input file can be split into a number of parts based on the size of the input and given to the mapper phase. This phase contains a lot of divides across the mapper and makes a lot of splits using Input Split. Every map split employs the Record Readers function, which turns the input text data into key-value pairs (key, value). Following the split, these key-value pairs will be sent to the mapper, which will generate intermediate data as key-value pairs. The reducer phase took those keys as input. The input from that and (k2, list (v2)) from the partitioned file is passed to the combiner class by the reducer. Lastly, the optimised value (k3, v3) must be uploaded to HDFS on each node in the cluster. Because all of the nodes in that cluster are in different locations, the replication factor will make a copy of the written data and distribute it to three distinct nodes.

Therefore, if a data node fails, the second node receives the duplicated value and assigns it to the user-assigned task. For this, fault tolerance and availability must be maintained, and execution and waiting times are relatively considerable. Our algorithm is now working on determining the most common item sets in the input file and then mapper classes, after which it will create the generations. The data was sent from the mapper to the reducer as key-value pairs dur-

ing the intermediate value. The scheduling procedure aids in efficiently sending output from the mapper to the reducer. This method works on scheduling and selecting frequent item sets from a big data set to produce the output as written in the HDFS within the specified execution time and average waiting time.

The concepts are utilized in a map-reduce architecture to lower the time it takes to complete a single operation. In addition to the traditional FPC and DPC algorithms, they include MFPC, ATDFC, optimised-SFPC, and optimised-ATDFC. To address the issue of FPC, DPC created additional algorithms such as variable-size FPC (VFPC) and elapsed-time DPC (ETDPC). It was attempting to reduce the execution and waiting times for individual jobs. However, at various stages of execution, the passes begin with 1, 2, 4, 8, 16,... When it comes to a multinode cluster, the time required has not dropped significantly. Our unique MFPC algorithm employs the approach of executing numerous passes after combining, where $n = 1, 2, 3,...$This approach accelerates the development of the number of passes, easily mixes the phases, and provides a low execution and waiting time. ATDPC also leverages the approach of feeding a pre-processed dataset item into map-reduce and maintaining it constant throughout the operation. because it minimizes the average execution time when several iterations are performed. The optimized versions of MFPC and ATDPC are similar, with the difference being the number of passes as an input for combined phases.

The unique approaches are relevant to Hadoop multinode cluster real-time operations on big data collections on Hadoop multinode clusters. In comparison to earlier strategies, this will give a novel method for extracting data from datasets with varying processing capacity. It will also be suitable for real-time generation of big data sets with access in different places with a minimum of waiting time.

4. Proposed algorithms

The primary node serves as the master node, while the remaining nodes serve as data nodes. The master node will assign the job to the data node while the incoming input data stream is initially transformed to hash values. The SHA function is used for this conversion, and the Mapper function divides the input data into little chunks based on key-value pairings. A number of Mappers will be formed to break up the original data before being reduced to a tiny size in comparison to the input size. The Mapper output is gathered as a partition from the Mapper function and reduced to a small amount of output depending on the Metadata of the input file size. The number of Mapper generated during the map () function is not identical to the number of Mapper created during the reduce () function. Because the reduce () function is used to merge all of the Mapper outputs to provide consolidated output.

Our new method defined two types of Map Reduce Jobs, Job 1 and Job 2. The first task creates frequent item set 1, and the second job generates k-item sets ($k > 2^n$). The first job is generated by several classes, specifically Apriori Mapper. Job 2 is used to collect intermediate data from the mapper output and split it using Apriori Combiner and Apriori Reducer.

Map Reduce based Apriori Algorithms were developed in Java utilising the Mapper, Pass1, PassK, and Reducer classes. The mapper class method produces candidates using a dataset split as input and a local support count. Reducer counts the sums of all local counts and creates frequent item sets with a global count. The combiner is used in these functions to collect intermediate data from the mapper and to minimise the cost between the mapper and the reducer. Moreover, Pass1 and PassK classes were created to receive user input and divide it before passing it to the mapper function for further processing. That will be repeated for k iterations. Here, k may be computed using the 2^n formula, yielding n of 1,2,4,8..... The dataset's content will be separated into the specified number of splits and turned into key-value pairs (k, v). The item sets are designated as (item, 1). This item set is known as Apriori Mapper. The other files are named Aprioripass1Mapper and Aprioripassk Mapper. The combiner is used to obtain data from map-reduce for merging numerous consecutive iterations using the Apriori Reducer class. Eventually, the reducer class will obtain the intermediate value from the mapper and partition it before combining it with the reducer class named Apriori Reducer.

The Aprioripass1Mapper class represents novel algorithms, while the Aprioripassk-Mapper class represents the notions of plain forward Single Pass Counting (SPC) used in task 2 for calling a new instance in a single pass or iteration of Apriori. The Trie data structure is utilised during the implementation phase to generate candidates and sort them. The most common k-item sets and potential sets are highlighted. To minimise time in a single phase, the preceding techniques were employed to combine the repeated Map Reduce Apriori phase runs. The preceding method, FPC, features permanent passes

that affect the incorrect positive candidate item sets in their initial phases, resulting in fewer candidates. DPC has addressed this issue by dynamically combining passes for load balancing of effort allotted by each phase. Yet, because of the dynamic passes, threshold values cannot be surpassed. The execution time is determined by the algorithm devised by the researchers.

As a result, for large datasets, it returns different results on cluster nodes from various locations. To address the aforementioned issue, we have developed new algorithms such as MFPC, ATDPC, Optimized-MFPC, and Optimized-ATDPC. Because of the dynamically integrated multiple passes, MFPC and ATDPC are the best and most flexible algorithms for large datasets in clusters with varying processing resources. Optimized-MFPC and Optimized-ATDPC are avoidance-pruning-based optimised versions of MFPC and ATDPC. Due to time constraints and function optimization, pre-processing can be omitted during the mapper phase.

4.1. Working of MFPC and ATDPC

MFPC and ATDPC are strategies that use a low number of passes during the beginning phase and a large number of passes throughout the remaining stages. As compared to all other algorithms, the Apriori method has a very small number of frequent or candidate item sets ($k = 1, 2, 3....$) at first. If this value is used as an input to the mapper, the iteration counter values must be increased and then lowered based on the mapper and reducer classes. Also, in the initial and final phase iterations, the mid-value of iterations is less than the number of candidate/frequent item sets. The iterations and passes are gradually increased and decreased with various values due to the K value, which is item set value. If the k value is varied, such as k is 1, 2, 4, 6,.... multiples of 2n, the iterations and passes provide different results at various times. Identifying the locations where values vary will offer the notion to adjust the passes and combines dynamically. It checks in two places: starting and finishing. Only the burden must then be balanced using combined passes. It is built on two novel strategies by total candidates among waiting time for combining the passes, which are the fundamentals of MFPC and ATDPC. MFPC candidates of 2n pass successively in one phase, such as 1, 2, 4, 8, 16, and so on. Then it works for $k >= 2$ passes. Since these passes determine the amount of iterations throughout execution time for frequent item set creation.

4.1.1. Pseudo Code for our new algorithm (Mapper side)

Step 1: Create item sets of size $k + 1$.

Step 2: Self-join L_k

Step 3: Generate the $(k + 1)$-item set with k-item sets

Step 4: Generate same item set up to (k-1) items

Step 5: If suppose 2 item set and their length 3 means create (list [0...2]) then [1, 2, 3], [1, 2, 4].

Step 6: Checks the item sets are the same from index 0 to index 1

Step 7: Generate 4-newItem set.

Step 8: If suppose 2 item set and their length 3 means create (list [0...2]) then [1, 2, 3], [2, 3, 4].

Step 9: Check index 0 and break.

Step 10: No need to generate the next item set.

Step 11: Consider generating $(k + 1)$-new Item set only for the same thing upto (k-1) consecutively.

Step 12: If all of the subsets are in L_k, then we add the item set to $C_{(k+1)}$.

Step 13: Add the item sets in $L_{(k+1)}$ into the Trie. In n-Map phase: $(n > 1)$

Step 14: Read part of transaction file and for each transaction, call find Item sets ()

Step 15: Emit the above item sets as (item set, 1).

4.1.2. Pseudo Code for Algorithm (Reducer side)

Step 1: Add the numbers up and if it's above minimum suppression.

Step 2: Add the item set into L_(k + 1).

The following Fig. 1 and Fig. 2 represents the working of new algorithms on the example data sets with

* Number of transactions = 6.
* min_sup = 50%.
* min_number of item = 2.

ATDPC determines whether the total number of consecutive passes generated by combined passes is less than a candidate threshold value. The key advantages of both MFPC and ATDPC are that they provide dependability and resilience with large datasets on Hadoop multi clusters at varied degrees of processing capability.

4.2. Optimized-MFPC and optimized-ATDPC

With this Optimized MFPC and ATDPC, the combined consecutive calculations avoid the preprocessing step in some phases. It decreases the number of candidates during Apriori property preprocessing. Apriori-gen is a novel mechanism used by MFPC and ATDPC mappers (). It employs

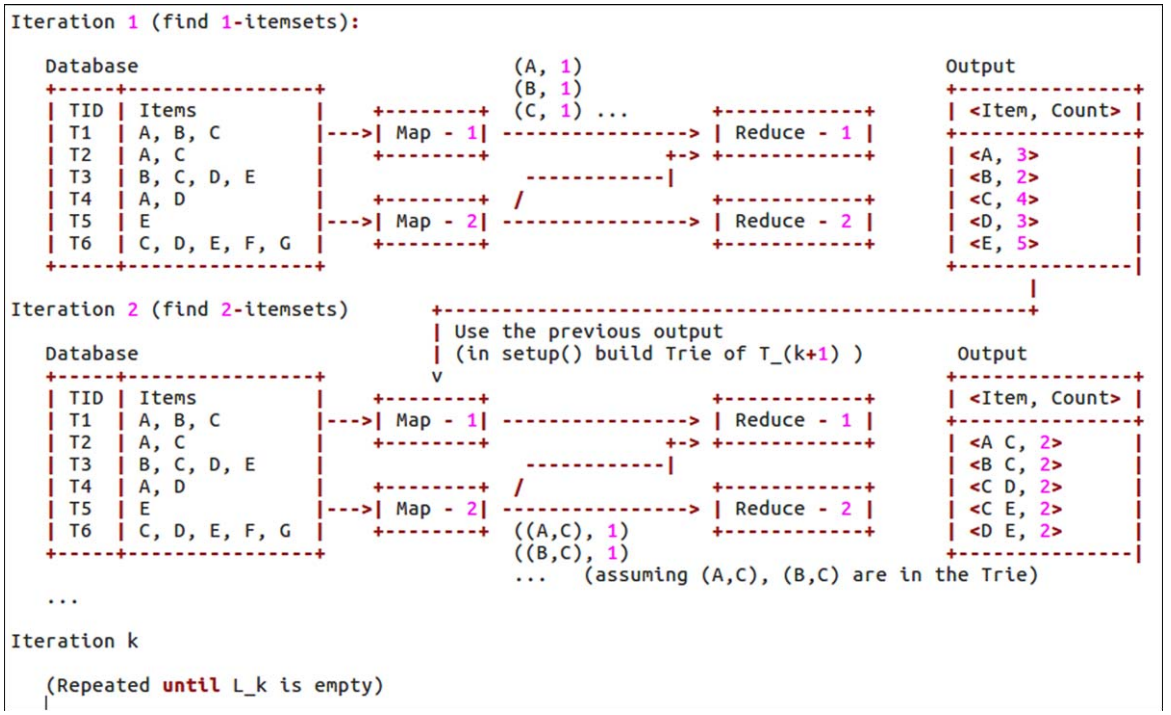

Fig. 1. Apriori Mapper working in MFPC.

two methods: connect and prune. Another technique, no-Apriori-gen (), is used to avoid the pre-processing stage and comprises of join functions. Both are incorporated into the Optimized techniques of our new algorithms. The unpruned candidates were created as a result of this no-apriori-gen (), which saves money owing to the pruning procedure.

The multi-pass phase pre-processing skipping technique is described in Algorithm 5 using Optimized new algorithms Mapper and Combiner class. It is comparable to the MFPC and ATDPC algorithms, with the exception of Mappers for job2. During runtime execution, the apriori-gen () function is utilised to discover all values.

4.3. *Skipped-pruning on multi-pass map reduce phase analysis*

The complexity of Apriori algorithms is affected by the number of transactions, item sets, average transaction time, and user threshold value. The pruning, subset, and apriori-gen () modules are sequential Apriori algorithm principles based on their components. Optimized algorithms of our new model work in sequential stages with the assistance of 2^n multipliers, and their dependencies are modified based on the computational cost in various modules. These are Pass k, Pass $k+1$, and Pass $k+2$. Because of the non-pruning strategy, the cost of the multiple pass phases has been lowered, but more non-pruned candidates have been produced. It is determined by the datasets, the min suppression parameter, and the number of transactions. The cost and performance of non-pruned candidate generation have been lowered as a result of the subset and self joining technique adopted. The prefix tree's size was expanded and saved for a separate multiple pass phase. It operates on each transaction and numerous passes using Mapper map () methods, and the result is saved in a tree. The candidates are generated using the apriori-gen () / no-apriori-gen () routines, as well as the subset () function. Since this subset () verifies a transaction's subset in each potential transaction. Because the apriori-gen () technique is used to obtain values from early item sets rather than transactions, it is recommended that it be used frequently in the map () function. During a multi-pass phase, the no-apriori-gen() technique is used to avoid the repeating of apriori-gen() function invocations. As a result, the time required to complete transactions and iterate is lowered, and the average time is determined accordingly. It operates because of the notions of self-joining and pruning.

```
AprioriDriver class : set up excution information about mapreduce job.

AprioriPass1Mapper class :
  It read the dataset line by line and emit <Individual item, 1>
  The dataset line is composed of [line number] [itemset]

  For example,
  sample.txt
  +---------------+
  |  1 1 3 4 2 5  |  -> line number : 0 Items : 1 3 4 2 5
  |  2 2 3 5      |  -> line number : 1 Items : 2 3 5
  |  3 1 2 3 5    |  -> line number : 2 Items : 1 2 3 5
  |  4 2 5        |  -> line number : 3 Items : 2 5
  + --------------+

  In this class, The Key is line number, txnRecord is items.
  And you should emit in this form: [individual item] 1
  i.e [1] 1
      [3] 1
      [4] 1

AprioriPasskMapper class:
  Pre-map phase
  n-Map phase

  you should emit in this form: [itemset] 1
  i.e if itemset length is 2,
      [1, 2] 1
      [2, 3] 1
      [3, 4] 1

AprioriReducer class
  Reduce phase

Transaction class:
  It represent transaction using list(Integer).

AprioriUtils class:
  It contains utility methods for Apriori algorithm.

TrieNode class
Trie class
```

Fig. 2. Working of Aprioripassk Mapper class.

5. Results and discussions

These research findings the enhanced features of apriori algorithm to improve the speed of the data processing in big data by eliminating the duplicate data sets from the collected frequent item sets is called pruning. During a multi-pass phase, the no-apriori-gen() technique is used to avoid the repeating of apriori-gen() function invocations. As a result, the time required to complete transactions and iterate is lowered, and the average time is determined accordingly. It operates because of the notions of self-joining and pruning.

5.1. Experimental overview

Our proposed study lab used Apache Hadoop 2.10.0. The findings have been compared in terms of

Table 1
Configuration of Hadoop Cluster

Node Name	Type of Nodes	No of CPU Cores	RAM Size GB	Processor Speed
Name Node	Physical	4	16	Intel 2.30GHz
Data Node 1	Virtual	8	8	Intel 2.10GHz
Data Node 2	Physical	8	4	Intel 2.15GHz
Data Node 3	Physical	4	8	Intel2.20GHz
Data Node 4	Virtual	8	4	Intel 2.15GHz

Table 2
Experimental results with transactions and average time

Datasets	Total Transactions	Total Items	Width Average
Mushroom	8124	119	23
Connect	67557	952	184
Retail	88162	1309	253

dataset size, processing time, execution time, average time, CPU running time, and latency. The cluster consists of four data nodes that are all running Ubuntu 16.04TLS 64 bit. The name node is physically setup as a server node, whereas the other data nodes are configured as virtual and physical, respectively. Java JDK 1.8.242 and the Map Reduce 2.0 (YARN) library were utilised to operate this configuration. Hadoop Distributed File System is based on Google File System (GFS) features. Being an open-source project, it is deployed as commodity hardware at a minimal cost. It breaks the files into a number of parts using various blocks. The default block size of Hadoop's new file system is 128 MB. The replication factor is set to 3 by default and may be modified depending on the number of systems in the cluster. Table 1 show that different configuration of Hadoop cluster.

The datasets namely sample, retail, connect and mushroom [29, 30] has taken for consideration and their total number of transactions, total number of items and average width of the transactions are listed below in Table 2. Each number are varied according to the size of the data set and it would be changed during the data processing.

Table 2 explain the analysis of average time with different datasets.

5.2. Performance analysis of MFPC and ATDPC

Classical approach and algorithms were tested by finding the frequent item sets with the combinations of K, $K+1 \ldots\ldots\ldots .2\,\mathrm{N}$ times calculations where as $N = 1,2,3,4$. But our new approaches findings the same with $2\,\mathrm{N}$ where N is $= 1,2,4,8,16 \ldots\ldots\ldots\ldots$ This method used in larger data sets to find the frequent item sets quickly and given accurate information from the repositories. Pruning is the concept used in this approach to remove the duplicate items immediately after the pre processing.

The comparison of various running times of tasks supplied as item sets may be analysed for all methods, and the results will be displayed on a graph. It demonstrates that all of the algorithms are running on time, but the time to complete the operation varies. Every execution is based on the minimum suppression values of all transactions at all levels. Because the many passes begin with 1, 2, 4, 8, and 16, there is no need to examine the performance at each pass. It will provide results with varying timeframes after completing all passes. The following findings describe the varied minimum suppression value transaction execution times of all methods, including the conventional technique. The figures below depict the variations in various

Figures 3, 4, and 5 show the relevance of using a traditional algorithm approach to data mining for obtaining information from large data sets. Since the execution time of the SPC algorithm is quite short when compared to all other new and old algorithms. Our new method runs with 2^{N} passes, therefore the result is displayed promptly at each level of the single multi pass phases. SPC has greater execution time value in Fig. 3a, 4a, and 5a when remaining algorithm is getting extremely near to executing the transactions. At the convergence point, all of the new algorithms are concentrated at one single site for interception, and the number of passes is gradually increased.

In Fig. 3b, 4b, 5b all the new algorithms are running simultaneously but when one saturation point comes it will get converged with different levels of minimum suppression values. Similar to that the execution of each data set is varied according to the size and number of transactions in the datasets. When huge data sets are executed it must be processed with our new approach and given the value accurately when saturation point comes. There must be lot of different size of

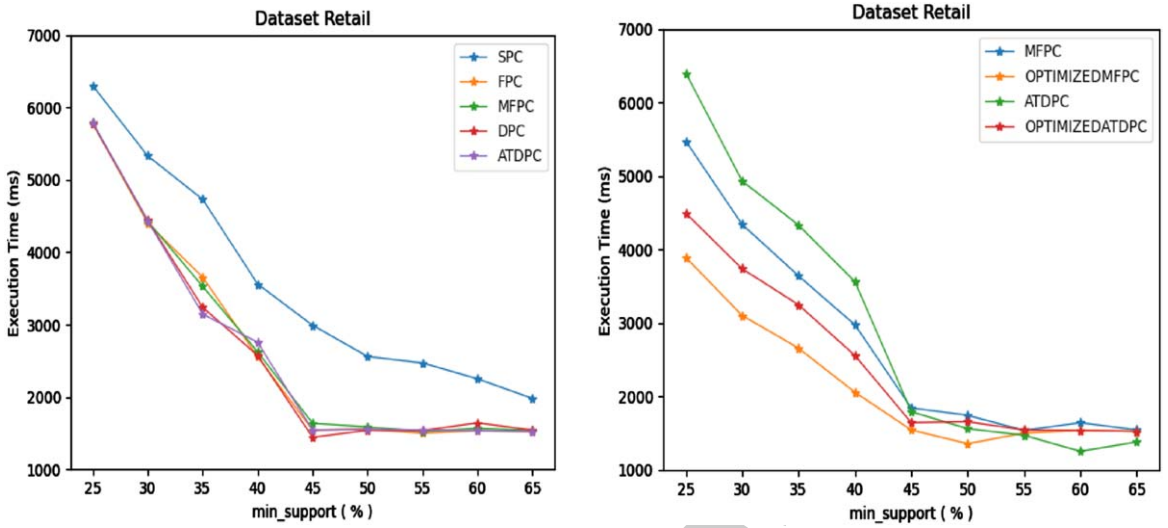

Fig. 3. (a) Execution time based on different min_sup using old algorithms (b) New algorithm execution time variations.

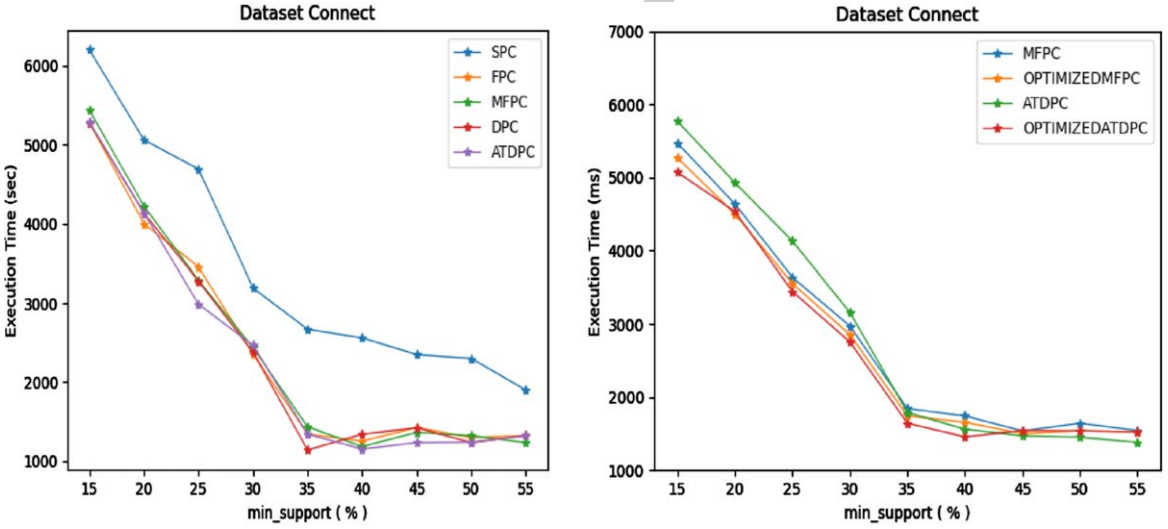

Fig. 4. (a) Execution time based on different min_sup using old algorithms (b) New algorithm execution time variations.

data sets taken for consideration which implies their results at their min_support values.

5.3. Result comparisons

The uniqueness of the suggested method should be justified via quantitative analysis based on several trials. All trials on the test bed were conducted based on the number of transactions and the minimum suppression values. The Tables 3–5 depicted that experimental values and demonstrate the varied outcomes at various min sup levels.

Tables 3–5 show three separate data sets processed at minimal suppression levels of 0.15, 0.55, and 0.65. It demonstrates that when the process begins, all transactions are unchanged, but after two or three runs, they are slightly altered with different values. When the size of the data set is grown, the pass output values may change since more passes are required to create outputs. The many passes begin at 1, 2, 4, 8, 16, 32, and 64. The above Table 3–5 demonstrates that when compared to the real time and time required finishing the operation with different passes, the outcomes for conclusion will vary. Lastly, as the size of the data collection grows, more passes and time are

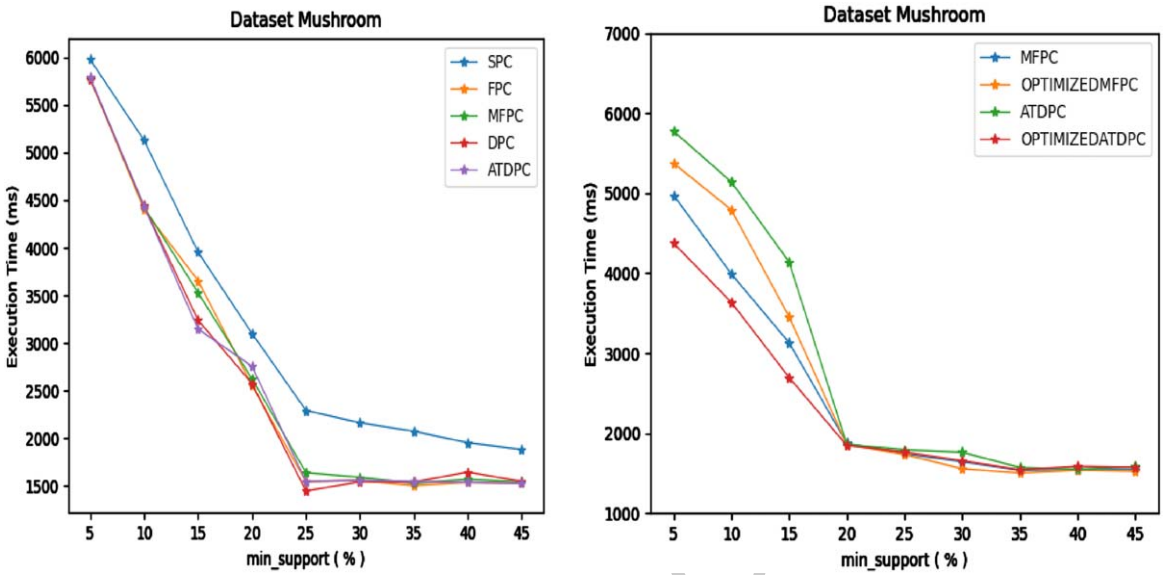

Fig. 5. (a) Execution time based on different min_sup using old algorithms (b) New algorithm execution time variations.

required for processing. There are several amounts of minimum suppression used, however when numerous passes are performed, the results will change. All yield the same result, with the average execution time of all passes dependant on the number of transactions in the overall operation. The time taken for complete the process with its total time taken and actual time has compared for the performance analysis.

Table 6 shows how different data sets generate their frequent item set at varied execution time intervals. Figures 3–6 shows that outcomes of candidate generation for several data sets operating on the same cluster. All of the findings show that at the beginning of each transaction, the levels are the same. Yet, it has taken longer execution time to accomplish the work by the time it reaches the intermediate level, such as the 8th pass iteration. As comparison to existing algorithms, our novel methods are employed to complete the work in less time. Table 7 depicted as generation of candidates at minimum suppression 0.15 on mushroom. Table 8 show that generation of candidates at minimum suppression 0.55 on connect. Table 9 shows that generation of candidates at minimum suppression 0.65 on retail.

Tables 10–12 described the execution time levels for each data set at varying minimum suppression levels. When comparing the actual time taken and overall time taken to finish the procedure, the data set size at each pass may vary. As compared to traditional methods, our technique provides reduced execution and average time for several runs.

5.4. Speedup and scalability

The test was performed to assess the scalability and speed of the test bed at various levels of minimum suppression values. Two timings have been taken into account: average execution time and average time. Both figures must be used to determine the amount of time necessary to generate frequent data item sets and candidate sets. The frequent data item sets, on the other hand, will be formed exclusively from the preceding item set. As a result, the timing levels should be updated in a single run each time a data set is created. When several passes are needed for an experiment, the average timing is calculated and compared to the original transaction time. Different minimum suppression levels are examined, and dataset size varies with each stage. Lastly, because of the 2n passes considered for execution, our unique approach provides findings from enormous data sets in a very short amount of time. Scalability has also been tested with various size input data sets, and all function in less time. Figures 6 (a & b) show the specifics of our new algorithm's speed and scalability numbers.

6. Conclusion and future enhancement

Data mining algorithms are the traditional ones used for mining information from repositories, however their accuracy and latency are bad in terms

Table 3
Different average timing levels at minimum suppression of 0.65 on retail data set

Multiple Passes	1	2	4	8	16	32	64esults	Total	Actual
SPC(16)	23	24	56	225	143	86	24	1360	1480
FPC(8)	22	24	340	582	362	63	25	1423	1523
DPC(12)	23	23	53	430	246	122	22	1237	1301
MFPC(8)	21	52	161	321	464	242	24	1261	1320
ATDPC(14)	21	25	30	56	152	212	23	1188	1238

Table 4
Different average timing levels at minimum suppression of 0.55 on connect data set

Multiple Passes	1	2	4	8	16	32	64	Total	Actual
SPC(12)	18	21	24	102	138	24	19	807	894
FPC(6)	21	19	120	130	456	163	27	856	887
DPC(10)	18	24	86	234	264	84	18	736	764
MFPC(7)	21	27	36	157	264	86	20	826	859
ATDPC(12)	20	25	37	163	129	328	21	735	771

Table 5
Different average timing levels at minimum suppression of 0.15 on mushroom data set

Multiple Passes	1	2	4	8	16	32	64	Total	Actual
SPC(14)	16	19	32	81	34	18	16	524	693
FPC(8)	17	27	136	262	85	24	24	528	572
DPC(8)	19	35	82	126	42	22	22	500	532
MFPC(9)	19	32	38	170	190	28	23	523	582
ATDPC(13)	20	32	82	165	173	26	21	512	558

Table 6
Generation of frequent item set creation time taken at different min_sup

Data base name	L1	L2	L4	L8	L16	L32	L64
Mushroom	46	520	6751	16837	1001	31	0
Connect	67	617	7992	21023	1384	19	0
Retail	56	609	7545	5100	417	16	0

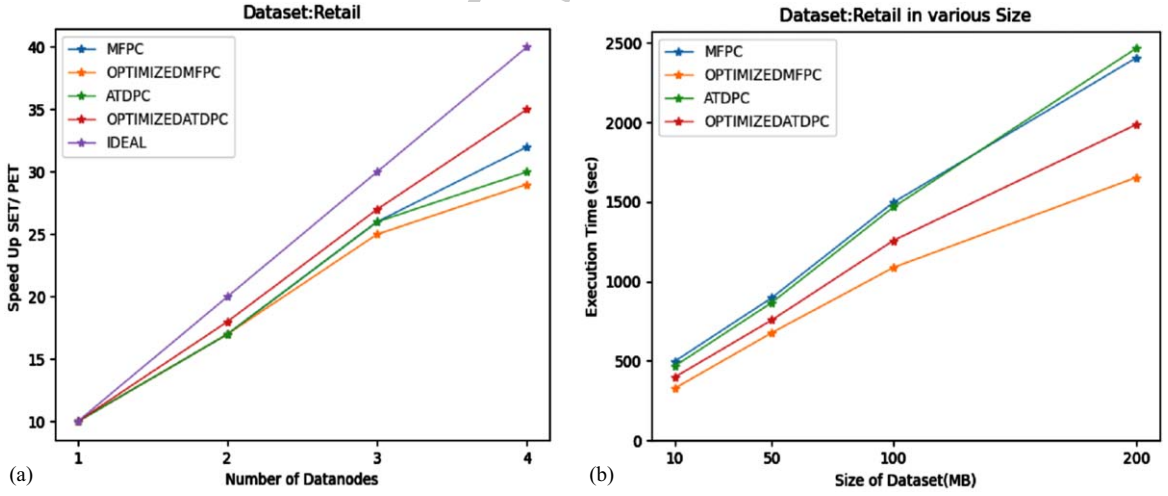

Fig. 6. (a) Speedup. (b) scalability.

of time. Modern ways to data processing are utilized in big data analytics, but the cost management and procedure are fairly typical when dealing with such large amounts of data. Despite this, data min-

Table 7
Generation of candidates at minimum suppression 0.15 on mushroom

Algorithm with no of phases	Pass 2	Pass 4	Pass 8	Pass 16	Pass 32	Pass 64
SPC	1118	7754	11536	1001	224	31
MFPC	16524	20424	32873	24358	11205	6
Optimized- MFPC	16524	21484	34728	24328	11225	6
ATDPC	1118	7754	11536	1001	224	2
Optimized-ATDPC	1118	7754	11536	1001	224	2

Table 8
Generation of candidates at minimum suppression 0.55 on connect

Algorithm with no of phases	Pass 2	Pass 4	Pass 8	Pass 16	Pass 32	Pass 64
SPC	1703	3602	5320	928	117	2
MFPC	9236	10349	7853	1052	1052	0
Optimized- MFPC	9236	12349	7462	1062	1052	0
ATDPC	1703	3602	5320	928	117	2
Optimized-ATDPC	1703	3602	5320	928	117	2

Table 9
Generation of candidates at minimum suppression 0.65 on retail

Algorithm with no of phases	Pass 2	Pass 4	Pass 8	Pass 16	Pass 32	Pass 64
SPC	2460	6577	20537	1287	253	2
MFPC	12345	14356	22631	2474	267	2
Optimized- MFPC	12765	16732	24835	2673	283	2
ATDPC	2460	6577	20537	1287	253	2
Optimized-ATDPC	2460	6577	20537	1287	253	2

Table 10
Different execution timing levels at minimum suppression of 0.65 on retail data set

Multiple Passes	1	2	4	8	16	32	64	Total	Actual
MFPC(8)	20	24	86	234	284	82	21	726	766
Optimized- MFPC(8)	21	23	75	190	86	74	23	567	612
ATDPC(9)	21	25	34	160	86	320	22	725	801
Optimized-ATDPC(9)	20	24	37	160	82	72	23	707	757

Table 11
Different execution timing levels at minimum suppression of 0.55 on connect data set

Multiple Passes	1	2	4	8	16	32	64	Total	Actual
MFPC(8)	19	26	96	214	264	92	21	626	666
Optimized- MFPC(8)	20	25	84	160	53	77	23	467	712
ATDPC(9)	20	27	42	140	45	235	22	625	701
Optimized-ATDPC(9)	21	26	39	140	75	75	23	507	557

ing techniques such as improved MFPC, FTPC, and their optimized supplementary algorithms are utilized to extract information from repositories with high latency and accuracy. Traditional data mining apriori methods were inappropriate for managing huge datasets due to the volume of data given as input to the HDFS system's map-reduce system. Although the apriori approach was utilized to construct frequent

Table 12
Different execution timing levels at minimum suppression of 0.15 on mushroom data set

Multiple Passes	1	2	4	8	16	32	64	Total	Actual
MFPC(8)	17	20	56	134	184	72	19	426	566
Optimized- MFPC(8)	18	22	65	130	76	44	20	467	512
ATDPC(9)	18	21	24	130	76	220	20	525	601
Optimized-ATDPC(9)	17	22	27	120	72	62	19	407	457

item sets and candidate item sets, the latency and execution time are insufficient for huge data analytics.

Proposed unique algorithms are based on apriori principles and reduce execution and average time by increasing pass values while utilizing an appropriate number of passes. Data mining concepts are advantageous for any input value pre-processing system, and it may have produced different frequent item groups for better data management. The enormous input files in HDFS are divided based on the number of files, and tasks are formed as item sets utilizing data mining pre-processing methods. Previous techniques were stated in terms of reducing execution time by 60–70% through the use of several passes. Because the trimming phase in data preprocessing is bypassed, our proposed new approaches will save up to 84–90% of that time. Machine Learning approaches will be developed in the future to boost the speed and scalability of the Hadoop Map Reduce System on multi-node clusters in HDFS. Additionally, the Java and Python programming languages will be used in the same way to optimize the process of big data analytics across several platforms.

References

[1] S. Singh, R. Garg and P.K. Mishra, Performance optimization of MapReduce-based Apriori algorithm on Hadoop cluster, *Computers & Electrical Engineering* **67** (2018), 348–364.

[2] L. Abualigah and B.A. Masri, Advances in MapReduce big data processing: platform, tools, and algorithms, *Artificial Intelligence and IoT* (2021), 105–128.

[3] K.L. Bawankule, R.K. Dewang and A.K. Singh, Load balancing approach for a MapReduce job running on a heterogeneous Hadoop cluster. In *International Conference on Distributed Computing and Internet Technology* (2021, January), (pp. 289–298). Springer, Cham.

[4] X. Wu, V. Kumar, J.R. Quinlan, J. Ghosh, Q. Yang, H. Motoda,... and Z.H. Zhou, Top 10 algorithms in data mining, *Knowledge and Information Systems* **14**(1) (2008), 1–37.

[5] E.R. Lucas Filho, E.C. de Almeida, S. Scherzinger and H. Herodotou, Investigating Automatic Parameter Tuning for SQL-on-Hadoop Systems, *Big Data Research* **25** (2021), 100204.

[6] P. Gupta and V. Sawant, A Parallel Apriori Algorithm and FP-Growth Based on SPARK. In *ITM Web of Conferences* (2021), (Vol. 40, p. 03046). EDP Sciences.

[7] T.H. Sardar and Z. Ansari, MapReduce-based Fuzzy C-means Algorithm for Distributed Document Clustering, *Journal of The Institution of Engineers (India): Series B* (2021), 1–12.

[8] J.C.W. Lin, Y. Djenouri and G. Srivastava, Efficient closed high-utility pattern fusion model in large-scale databases. Information Fusion. (2021).

[9] L.S. Rakhimova and O.U. Khalmuratov, Performance analysis of association rule mining algorithms using hadoop, *Scientific Progress* **2**(2) (2021), 149–153.

[10] M. Sornalakshmi, S. Balamurali, M. Venkatesulu, M.N. Krishnan, L.K. Ramasamy, S. Kadry and S. Lim, An efficient apriori algorithm for frequent pattern mining using mapreduce in healthcare data, *Bulletin of Electrical Engineering and Informatics* **10**(1) (2021), 390–403.

[11] M. Yimin, G. Junhao, D.S. Mwakapesa, Y.A. Nanehkaran, Z. Chi, D. Xiaoheng and C. Zhigang, PFIMD: a parallel MapReduce-based algorithm for frequent itemset mining, *Multimedia Systems* (2021), 1–14.

[12] W. Xiao and J. Hu, Paradigm and performance analysis of distributed frequent itemset mining algorithms based on Mapreduce, *Microprocessors and Microsystems* **82** (2021), 103817.

[13] S. Raj, D. Ramesh and K.K. Sethi, A Spark-based Apriori algorithm with reduced shuffle overhead, *The Journal of Supercomputing* **77**(1) (2021), 133–151.

[14] H. Guo, H. Liu, J. Chen and Y. Zeng, Data Mining and Risk Prediction Based on Apriori Improved Algorithm for Lung Cancer, *Journal of Signal Processing Systems* (2021), 1–15.

[15] H. Yu, Apriori algorithm optimization based on Spark platform under big data, *Microprocessors and Microsystems* **80** (2021), 103528.

[16] P. Chugh and H.K. Verma, Various Techniques to Improve the Efficiency of Apriori Algorithm: A Review. In *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)* (2021, May), (pp. 471–476). IEEE.

[17] S. Dongnan and Z. Zhaopeng, Parallel Design of Apriori Algorithm Based on the Method of "Determine Infrequent Items & Remove Infrequent Itemsets". In *IOP Conference Series: Earth and Environmental Science* (2021, February), (Vol. 634, No. 1, p. 012065). IOP Publishing.

[18] Y. Xun, J. Zhang, H. Yang and X. Qin, HBPFP-DC: A parallel frequent itemset mining using Spark, *Parallel Computing* **101** (2021), 102738.

[19] L. Hamdad and K. Benatchba, Association Rules Mining, *SN Computer Science* **2**(6) (2021), 1–19.

[20] H.T. Rauf, S. Malik, U. Shoaib, M.N. Irfan and M.I. Lali, Adaptive inertia weight Bat algorithm with Sugeno-Function fuzzy search, *Applied Soft Computing* **90** (2020), 106159.

[21] L. Ali, I. Wajahat, N.A. Golilarz, F. Keshtkar and S.A.C. Bukhari, LDA–GA–SVM: improved hepatocellular carcinoma prediction through dimensionality reduction and genetically optimized support vector machine, *Neural Computing and Applications* **33**(7) (2021), 2783–2792.

[22] L. Ali and S.A.C. Bukhari, An approach based on mutually informed neural networks to optimize the generalization capabilities of decision support systems developed for heart failure prediction. Irbm. (2020).

[23] M.J.S. Fard and P.A. Namin, Review of Apriori based Frequent Itemset Mining Solutions on Big Data. In *2020 6th International Conference on Web Research (ICWR)* (2020, April), (pp. 157–164). IEEE.

[24] SPMF Datasets, http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

[25] Frequent Item set Mining Dataset Repository, http://fimi.ua.ac.be/data/

[26] R. Sun and Y. Li, Applying Prefixed-Item set and Compression Matrix to Optimize the Map Reduce-based Apriori Algorithm on Hadoop. In *Proceedings of the 2020 9th International Conference on Software and Computer Applications* (2020, February), (pp. 89–93).

[27] A.W.O. Gama and N.M. Widnyani, Simple Modification for an Apriori Algorithm with Combination Reduction and Iter-

ation Limitation Technique, *Knowledge Engineering and Data Science* **3**(2) (2020), 89–98.

[28] J. Wang, X. Li, R. Ruiz, J. Yang and D. Chu, Energy Utilization Task Scheduling for Map Reduce in Heterogeneous Clusters, in *IEEE Transactions on Services Computing* **15**(2) (2022), pp. 931–944, 1 March-April 2022, doi: 10.1109/TSC.2020.2966697.

[29] Q. Huiqi, Improvement parallelization in Apriori Algorithm. In Proceedings of the 2020 International Conference on Computers, *Information Processing and Advanced Education* (2020, October), (pp. 235–238).

[30] R. Alanazi, F. Alhazmi, H. Chung and Y. Nah, A multi-optimization technique for improvement of Hadoop performance with a dynamic job execution method based on artificial neural network, *SN Computer Science* **1** (2020), 1–11.

[31] H. Wang, H. Jiang, H. Wang and L. Yuan, Research on an improved algorithm of Apriori based on Hadoop. In *2020 International Conference on Information Science, Parallel and Distributed Systems (ISPDS)* (2020, August), (pp. 242–245). IEEE.

[32] H. Wu, Data Association Rules Mining Method Based on Improved Apriori Algorithm. In *2020 the 4th International Conference on Big Data Research (ICBDR'20)* (2020, November), (pp. 12–17).

[33] M.Y. Lin, P.Y. Lee and S.C. Hsueh, Apriori-based frequent itemset mining algorithms on MapReduce. In *Proceedings of the 6th international conference on ubiquitous information management and communication* (2012, February), (pp. 1–8).

[34] M.S. Kumar, S. Sankar, V.K. Nassa, D. Pandey, B.K. Pandey and W. Enbeyle, Innovation and creativity for data mining using computational statistics. In *Methodologies and Applications of Computational Statistics for Machine Intelligence* (2021), (pp. 223–240). IGI Global.

[35] M.R. Sundarakumar, G. Mahadevan, R. Natchadalingam, G. Karthikeyan, J. Ashok, J.S. Manoharan and P. Velmurugadass, A comprehensive study and review of tuning the performance on database scalability in big data analytics, *Journal of Intelligent & Fuzzy Systems (Preprint)*, 1–25.

[36] M.R. Sundarakumar, G. Mahadevan, R. Somula, S. Sennan and B.S. Rawal, An Approach in Big Data Analytics to Improve the Velocity of Unstructured Data Using MapReduce, *International Journal of System Dynamics Applications (IJSDA)* **10**(4) (2021), 1–25.

[37] M.R. Sundarakumar, et al. A Heuristic Approach to Improve the Data Processing in Big Data Using Enhanced Salp Swarm Algorithm (ESSA) and MK-means Algorithm 1 Jan. 2023, 1–16. DOI: 10.3233/JIFS-231389.