Iterative Approach for Frequent Set Mining Using Hadoop Over Cloud Environment

S. Prasanna, Subashini Narayan, M. K. NallaKaruppan, Chunduru Anilkumar and Somula Ramasubbareddy

Abstract Cloud computing initially gained popularity as it offered an alternative for handling the ever-growing size of data. One of the main advantages of Cloud computing is parallel processing of data, which causes the effect of pooling the resources of various systems. The proposed project aims to implement the feature for the purpose of data mining and will use the Apriori Algorithm to demonstrate the results. Hadoop platform will be utilized for this project. The system will receive a dataset and redistribute it to the nodes of the cloud. Here, Apriori algorithm will be applied upon the sections of the dataset and the results will then be combined to obtain the frequent itemsets in the global data. Using the frequent item sets, rule mining will be achieved.

1 Introduction

Mobile—The main driving force behind Cloud computing are the IT giants such as IBM, Amazon, Microsoft, etc. With the growing popularity of digitization given the benefits it offers in maintaining records, big data is common in more and more organizations. Often the size of data makes it impossible for a human to remove the noise in the data and recognize complex patterns and relations. Data mining is a combination of statistical sampling, estimation, hypothesis testing, and pattern recognition. In abroad sense, data mining is often referred to as knowledge discovery in database or KDD. The importance of data mining algorithms and improvement of the algorithms lies in the fact that with the ever-growing size of data, traditional methods of finding patterns or making predictions continue to grow obsolete. This is mainly because the algorithm has to process a huge amount of data that is also being constantly updated. Data mining algorithms focus on reducing the time and computing power required to process huge data sets. The problem of increasing requirement of computing power is solved using parallel processing. Naturally, data

S. Prasanna · S. Narayan · M. K. NallaKaruppan · C. Anilkumar · S. Ramasubbareddy (✉)
VIT University, Vellore, Tamil Nadu, India
e-mail: svramasubbareddy1219@gmail.com

S. C. Satapathy et al. (eds.), *Smart Intelligent Computing and Applications*,
Smart Innovation, Systems and Technologies 105,
https://doi.org/10.1007/978-981-13-1927-3_43

mining algorithms have to be tweaked so that they can effectively distribute work load amongst various nodes and work efficiently [1, 2]. Most computers in a network never utilize their processing capability to the maximum extent. Cloud computing can be used to efficiently distribute big data and process it at the nodes. This means that the entire processing power need not be centralized and that instead of investing in a new high end system, the systems that are already in use can be utilized. Algorithms must be periodically revised and improved upon to better utilize the power of parallel computing [3, 4]. Cloud computing has given rise to a new business model that allows distributed storage and processing of data thus reducing the cost of IT infrastructure [5]. The usage of "clouds" for utility-based billing means that an organization can reduce costs depending upon the efficiency of algorithms deployed by it. Map/Reduce architecture forms the core operation for parallel computing using Hadoop. The map operation allows big data to be mapped to keys and then distributed amongst the nodes in cloud while the reduce operation gathers results from the nodes and combines them together. MapReduce has been used for the implementation of machine learning based methods such as classification, logic regression, linear support vector, etc.

2 Background

Big Data is an expression used to denote an enormous volume of data that is so massive that it is tough to process utilizing conventional computing strategies. In most business situations the size of data is too huge or it surpasses current handling limits. Big Data enables organizations to enhance operations and make quicker, more insightful decisions. The data is gathered from various sources including messages, cell phones, applications-mails, databases, etc. This information can enable an organization to increase valuable understanding to enhance incomes and growth levels, strengthen operations, improve Customer Feedback, and retain Customers. Big Data usually includes utilizing NoSQL and Distributed Computing techniques to scrutinize the data. Cloud Computing provides IT assets like Infrastructure, Platform, Software, and Storage as Services. Some of the trademark characteristics of Cloud Computing include: Adaptable Scaling, Availability of scalable provisioning, High Availability, Asset Pooling, On-demand and pay as per requirement and demand.

In simple terms, Big data can be described as the input and Cloud Computing can be described as the framework in which operations and jobs can performed on the Big Data for its analysis. Big Data can be organized and processed in a Cloud Environment. MapReduce was developed by a few employees of Google. MapReduce is a programming paradigm utilized for parallel computations. Google's version of MapReduce is likewise called MapReduce and it not accessible to the general population, but the key ideas and concepts were published. Apache Hadoop is a free and open source implementation of the MapReduce model. It is reliant upon its intrinsic Hadoop Distributed File System (HDFS). It is based on Java. The principal thought behind MapReduce is to divide the problem into two subproblems: a guide and a decrease stage. First, the input information gets cut into littler pieces

and each lump is given to a machine that executes a mapper. The mapper forms that lump and gives key esteem matches as the yield. Than mapreduce structure gathers these sets, sorts them in view of the key and passes them to reducers. A reducer gets a key and a rundown of qualities that has a place that key and gives the outcomes. The principle thought behind MapReduce is to divide the proposition into two subproblems: Map phase and Reduce Phase. First, the input information gets cut into little pieces and each lump is given to a machine that executes a mapper. The mapper forms that lump and gives key-value pairs as the output. Than MapReduce structure gathers these sets, sorts them in view of the key and passes them to the Reducers. A reducer then performs the requisite computations and consequently outputs the result. The tasks isolated by the principal application are right off the bat processed initially by the map jobs in a totally parallel fashion [6]. The MapReduce system sorts the yields of the maps, which are then used as input to the reduce jobs. The Hadoop Distributed File System then stores both the input and the output of the job. Hadoop Distributed File System (HDFS) is a framework that holds an expansive amount of data in terabytes or petabytes and gives quick and versatile access to this data [7]. It stores records in a redundant manner over many nodes to give adaptation to resistance failure and high accessibility amid execution of parallel applications. HDFS breaks a document into blocks of constant size (default square size is 64 MB) to store over many nodes. Hadoop utilizes a NameNode as master node and various DataNodes as slave nodes. The NameNode allots ids to the data blocks and stores additional data related to it such as permission, name, etc., in the form of metadata thus enabling quick access to this data. Data Nodes are the individual nodes which store and recover the redundant blocks of different records.

The following diagram illustrates the basic architecture and relationship between various nodes in a Hadoop framework (Fig. 1).

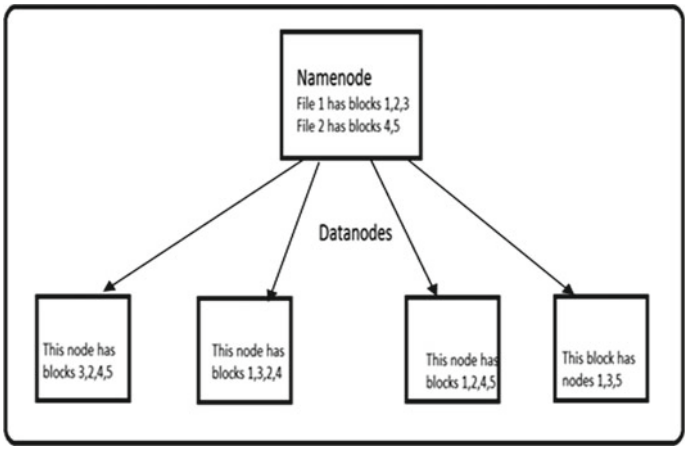

Fig. 1 Data nodes and slave nodes in HDFS with block replication

Association Rule Mining endeavors to discover frequent itemsets among huge datasets and depicts the association relationship among various attributes. It was initially described in the paper "Fast Algorithms for Mining Association Rules" By R. Agarwal and R. Srikant. Apriori Algorithm is arguably the most popular algorithm used for Association Rule Mining. Apriori Algorithm utilizes an iterative approach for finding the frequent itemsets of a given dataset. It performs the search in an iterative manner layer-by-layer. It basically generates $k+1$ itemsets by utilizing the k-itemsets. The basic algorithm can be described as

1. Let C_k and L_k be defined as the Candidate Itemset and Frequent itemset of "k" size respectively.
2. Initialize L_1
3. Now generate C_{k+1} from L_k.
4. For every transaction in the dataset, increase the count of all candidates in C_{k+1} that are included in that transaction.
5. Now generate L_{k+1} using the item candidates that have support greater than the minimum support.
6. Loop till L_k is not empty.
7. From this obtain all the k-sized frequent itemsets L_k.

The benefits of using this simple approach to Apriori algorithm is that is easy, intuitive, straightforward, no convoluted derivations are involved. Furthermore, the nature of this simple algorithm significantly diminishes the number of candidates to be verified and hence we can observe an enhancement in the efficiency of this algorithm.

Some of the disadvantages and overheads associated with this approach are

1. Colossal overhead is caused due to scanning the database multiple times. For datasets containing N-length frequent itemsets as the largest frequent itemset, we need to scan the database N number of times. This puts additional burden on the processor and hence slows down the entire process
2. It may create countless itemsets. Sometimes this number becomes very large and the algorithm becomes infeasible to implement.

Hence by generating large number of itemsets and by scanning the database multiple times, Apriori algorithm becomes computationally very expensive and will consume a lot of time as well as memory space. To overcome these limitations of the basic Apriori Algorithm, we utilize improved Apriori Algorithm. As a cloud computing environment can support parallel/distributed approach to computation, it can be utilized to optimize the existing Apriori Algorithm. Hence, parallel association rule mining techniques can be used for this purpose.

The improved algorithm can be described as

1. Divide the database into "n" subsets and distributed to "m" nodes for performing the parallel scan of the database.
2. Scanning of individual divided data set is performed at each node and then candidate set C_p is generated on the basis of this.

3. Support count of these candidate sets C_p is set to 1. After this, the candidate itemset is partitioned into and sent to "r" nodes along with their support counts.
4. These nodes then sum up the support counts of the same itemsets and to generate the final support count.
5. Establish the frequent itemset in the partition after tallying with the minimum support count as per the required application.
6. Merge the outputs from these nodes to produce the final global frequent itemset.

The aforementioned improved Apriori algorithm is utilized to extensively lessen the time as in this calculation the database needs to be scanned only once instead of the multiple times in the original Apriori Algorithm.

This modified and improved Apriori Algorithm can hence be implemented on Apache Hadoop Framework with the MapReduce Model as this Framework has the requisite tools and methodologies to successfully implement this modification. The algorithm can then be described as [8, 9, 10]

(1) The database is partitioned into n sets and are then sent to m nodes for processing by executing Map jobs.
(2) The datasets need to be in the form of a key and value pair for processing in MapReduce Model. Hence, this formatting needs to be done.
(3) Candidate sets are generating by the mapping function running in each node.
(4) The results are then combined by the combiner function.

The drawbacks and benefits of using this MapReduce

Advantages	Disadvantages
Reduction of Network Bandwidth Usage	Can only operate on data which is in the format of key, value pairs
Parallelization and efficient workload distribution	Next stage of the computation cannot be started until and unless the reducer has finished executing
Makes the entire problem very scalable	The data distribution version of Apriori is not suitable to be implemented using MapReduce
A platform which enables both distributed storage and high computing power	

3 Proposed Methodology

Using the improvements provided by Lian et al. [4] we intend that the algorithm reads through the entire dataset only once. Additionally, with each subsequent iteration, to find the larger frequent itemset, the data to be considered is reduced. For this purpose, we utilize the core operations of Hadoop that are MapReduce. While mapping, as the

transactional dataset is scanned, we assign each item as a key and the value as one. Once the mapping procedure is complete, the Hadoop framework distributes data to all available nodes. For reduce operation, the reducer counts the occurrence of each item, combines the findings and eliminates the items that do not reach the minimum support. The generation of k-itemsets involves referring to the $k-1$ itemsets where the itemset is key and the number of occurrence is value. The frequent $k-1$ itemsets that hold similar items are the keys and the corresponding values will be sent to the same reducer job. Subsequently, the same reducer procedure is repeated where an itemset of size k is generated by combining the available $k-1$ itemsets and their count is compared to the minimum support count. The values corresponding to $k-1$ itemsets are further used in parallel for rule mining. The key-value combination is saved in corresponding files or other available memory. This means that during association rule mining, the key-count of both k-itemsets and $k-1$ itemsets are already available and need not be computed again. This saves processing time as well as provides the association rules for all frequent itemsets from size two to k, where k is the size of the largest itemset that satisfies the minimum support count.

4 Simulation Results

The running times of the both the original and improved algorithms were plotted against the no. of jobs to execute. The dataset chosen was a data set containing 1000 transactions. The running times for each of these situations were measured independently five times and the result chosen for plotting this graph is the average of these five observations. The results observed could be tabulated in the form of a graph as shown (Fig. 2).

In the above graph, the y-axis represents the seconds required to execute the MapReduce job and the x-axis represents the no. of jobs (the no. of cores/processors). As we can see, increasing the degree of concurrency to six or eight jobs results in no significant performance improvement and reaches almost saturation levels w.r.t reduction in processing time and speed of execution. This can be attributed to the fact that this testing was carried out in an environment where the CPU had four cores and increasing the degree of concurrency would not increase the utilization of CPU

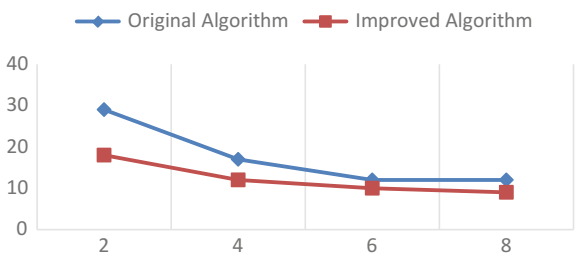

Fig. 2 Comparison of running times of the algorithms plotted against the number of jobs to execute

further. Also, as a general trend, it can be concluded from the graph that execution time for the improved algorithm is much faster than the running time of the original algorithm when CPU utilization is high and the CPU is not overwhelmed with a high no. of concurrent jobs.

5 Conclusion

In this paper, we discussed and analyzed on how to improve the performance and efficiency of frequent set mining using Apriori algorithm. For this purpose, a cloud-based environment utilizing HadoopMapReduce framework was proposed in this paper. Furthermore, an improved Apriori algorithm was proposed in this paper that further optimizes the performance of the MapReduce job by using the intermediate results obtained instead of scanning the original data set repeatedly resulting in massive overheads. The performance of these two algorithms was compared by comparing the running times of the algorithms in different scenarios. As a general rule of thumb, it could be observed from the tabulated results that the improved algorithm performs significantly faster than the original Apriori algorithm. Hence, an improved and robust apriori algorithm technique was put forward in this paper for frequent rule mining in a cloud environment.

References

1. Ekanayake, J., Fox, G.: High performance parallel computing with clouds and cloud technologies. In: International Conference on Cloud Computing. Springer, Berlin, Heidelberg (2009)
2. Sheth, N.R., Shah, J.S.: Implementing parallel data mining algorithm on high performance data cloud. Int. J. Adv. Res. Comput. Sci. Electr. Eng. (IJARCSEE) **1**(3), 45 (2012)
3. Jin, R., Yang, G., Agrawal, G.: Shared memory parallelization of data mining algorithms: techniques, programming interface, and performance. IEEE Trans. Knowl. Data Eng. **17**(1), 71–89 (2005)
4. Lian, W., et al.: Cloud computing environments parallel data mining policy research. Int. J. Grid Distrib. Comput. **8**(4), 135–144 (2015)
5. Chang, X.-Z.: Mapreduce-Apriori algorithm under cloud computing environment. In: 2015 International Conference on Machine Learning and Cybernetics (ICMLC), vol. 2. IEEE (2015)
6. Ezhilvathani, A., Raja, K.: Implementation of parallel apriori algorithm on hadoop cluster. Int. J. Comput. Sci. Mob. Comput. **2**(4), 513–516 (2013)
7. Tiwary, M., Sahoo, A.K., Misra, R.: Efficient implementation of apriori algorithm on HDFS using GPU. In: 2014 International Conference on High Performance Computing and Applications (ICHPCA). IEEE (2014)
8. Vajk, I.A.: Performance evaluation of Apriori Algorithm on a Hadoop cluster. Wseas. Us, pp. 114–121 (2013)
9. Singh, S., Garg, R., Mishra, P.K.: Review of apriori based algorithms on mapreduce framework. arXiv preprint arXiv:1702.06284 (2017)
10. Saabith, A.L.S., Sundararajan, E., Bakar, A.A.: Parallel implementation of apriori algorithms on the hadoop-mapreduce platform-an evaluation of literature. J. Theor. Appl. Inf. Technol. **85**(3), 321 (2016)