ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

A Review on Various Frequent Itemsets Mining Algorithms

Bhavana G[1], Jyothi Jeniffer Dias[2], Prof Shobha M S[3], Dr Jitendranath Mungara[4]

B.Tech Student, Department of Information Science & Engineering, NHCE, Bangalore, India[1, 2]

Senior Assistant Professor, Department of Information Science & Engineering, NHCE, Bangalore, India[3]

Head of Department, Department of Information Science & Engineering, NHCE, Bangalore, India[4]

ABSTRACT: Existing parallel mining algorithms for frequent itemsets lack a mechanism that enables automatic parallelization,load balancing, data distribution, and fault tolerance on large clusters. As a solution to this problem, we design a parallel frequent itemsets mining algorithm called FiDoop using the MapReduce programming model. To achieve compressed storage and avoid building conditional pattern bases, FiDoop incorporates the frequent items ultrametric tree, rather than conventional FP trees. In FiDoop, three MapReduce jobs are implemented to complete the mining task. In the crucial third MapReduce job, the mappers independently decompose itemsets, the reducers perform combination operations by constructing small ultrametric trees, and the actual mining of these trees separately. FiDoop on the cluster is sensitive to data distribution and dimensions, because itemsets with different lengths have different decomposition and construction costs. To improve FiDoop’s performance, we develop a workload balance metric to measure load balance across the cluster’s computing nodes.

KEYWORDS: Frequent itemsets, load balance, Frequent Itemset Mining, MapReduce Programming Model.

I.INTRODUCTION

The first step in the discovery of association rules is to find each set of items (called itemset) that have co-occurrence rate above the minimum support. An itemset with at least the minimum support is called a frequent itemset. The size of an itemset represents the number of items contained in the itemset, and an itemset containing k items will be called a k-itemset. Finding all frequent itemsets is a very resource consuming task and has received a considerable amount of research effort in recent years. The second step of forming the association rules from the frequent itemsets is straightforward as described: For every frequent itemset f, find all non-empty subsets of f. For every such subset a, generate a rule of the form a => (f -a) if the ratio of support(f) to support(a) is at least minconf. The association ruleX=>Y holds in the database D with confidence c if c% of transactions in D that contain X also containY. The rule X=>Y has support s if s% of transactions in D contain X U Y. Mining association rules is to find all association rules that have support and confidence greater than or equal to the user-specified minimum support (called minsup) and minimum confidence (called minconf) , respectively.

Frequent itemsets mining (FIM) is a core problem in association rule mining (ARM), sequence mining, and the like. Speeding up the process of FIM is critical and indispensable, because FIM consumption accounts for a significant portion of mining time due to its high computation and input/output (I/O) intensity. When datasets in modern data mining applications become excessively large, sequential FIM algorithms running on a single machine suffer from performance deterioration. To address this issue, we investigate how to perform FIM using MapReduce—a widely adopted programming model for processing big datasets by exploiting the parallelism among computing nodes of a cluster.

In this survey paper, we will have a look at the algorithms that will be used in this project, in addition to the algorithms that were previously used. Here, we have some base papers that use these algorithms.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

II. LITERATURE SURVEY

A. MR-Apriori: Association Rules Algorithm Based on MapReduce

Apriori algorithm in a nutshell:

First find frequent itemsets (sets of items that have minimum support). A subset of a frequent itemset must also be a frequent itemset (Apriori Property). For example, if {A,B} is a frequent itemset then {A} and {B} should be frequent itemsets. Use these frequent itemsets to generate association rules.

MR-Apriori (MR-MapReduce) is an iterative process implemented with the MapReduce programming model. MR-Apriori algorithm can reduce number of times of scanning the transaction database. Also, reduces amount of data sent to reducers. This algorithm is as follows:

1. Data partitioning and distributed data: Transaction database is divided into subsets.
2. Formatting data subsets: Format the datasubsets in <key,value> pairs, where key is the transaction ID. Also,set k to 1.
3. Execute map task to generate candidate itemsets (frequent itemsets generated based on Apriori property).
4. Execute combiner function: Combines map function output and gives<itemset,support_count> intermediate pairs. Combiner function then uses partition function to divide intermediate pairs generated by combiner function in different partitions.
5. Execute Reduce task to get frequent itemset L_1: At reduce function, key itemsets are first sorted, then add support count of same candidates to get actual support count for the whole transaction database and finally get the frequent itemset L_1, by comparing with the minimum support.
6. Increment k and global frequent itemset L_{k-1} is sent to each node.
7. Repeat steps 3-6.

B. Balanced Parallel FP-Growth with MapReduce

FP-growth algorithm consists of 2 steps:

Step-1:Construction of FP-tree:

Build FP-tree (compact DS) which is built by 2 passes over the dataset. In pass-1 create the F-list and scan data and find support for all items. Ensure to discard infrequent items. Sort frequent items in decreasing order based on support (use this order for building FP-tree ->since common prefixes can be shared). In pass-2 build a FP-tree.In FP-tree, nodes correspond to items and each node has a counter. FP-growth reads one transaction at a time and maps it to a path. Since fixed order is used, paths can overlap when transactions share items (i.e., when they have same prefix ->in this case counters are incremented).

Step-2: Extracts frequent itemsets directly from the FP-tree:

Frequent Itemset generation: Frequent itemsets can then be extracted from the FP-tree.FP-growth traverses nodes in FP-tree from least frequent item in F-list (bottom-up). While visiting each node from node to root of tree it collects some items. These items form conditional pattern base (CPB) for that item.The CPB is like a small database of patterns that co-occur with the item. This is repeated until no CPB can be generated.

In BPFP, we first estimate work load of each mining unit and then fairly divide these units into several groups in order to balance load among all groups.With load balance feature, BPFP partitions the whole mining task into even more subtasks, hence achieving higher parallelization.Given transaction database and minimum support count, BPFP uses 2 rounds of MapReduce to parallelize FP-growth. This algorithm is as follows:

1. Sharding: Dividing database into successive partitions and storing on different machines.
2. Parallel counting: Count all the inputs (i.e., <key, value> pair of a transaction) of database. This done in parallel by a MapReduce job. The result is the F-list, which is in descending order.
3. Balanced grouping: Fairly divides all items in F-list into groups based on the load of that group. It is done by computing load unit and fairly divide all load units into several groups. This is done by balanced partition. Balanced partition means all items in F-list are sorted by load in descending order forming L-list.

Repeat these 2 steps until all items in L-list are grouped:

1. Add next non-grouped item in L-list to group with minimum load.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

2. Increase load of that group by load of new item.

C. *An Efficient k-Means Clustering Algorithm:Analysis and Implementation*

A k-nearest neighbor join (kNN join) is a special type of join that combines each object in a dataset R with k objects in another dataset S that are closest to it (scan S once for each object in R). kNN join is an expensive operation, since it's a combination of k nearest neighbors query and the join operation. To improve scalability we consider parallelism in a distributed computational environment. MapReduce is a programming framework for processing large scale datasets by exploiting the parallelism among a cluster of computing nodes. Mappers cluster objects into groups. Reducers perform kNN join on each group of objects separately. To improve efficiency, we have to reduce shuffling and computation costs. The existence of replicas leads to a high shuffling cost and also increases the computational cost of the join operation within a reducer. Hence, a good mapping function that minimizes the number of replicas is one of the most critical factors that affect the performance of the kNN join in MapReduce. In Voronoi-diagram based partitioning each object is assigned to partition with its closest pivot and the data space is split into voronoi cells.

kNN join using MapReduce begins with preprocessing step. It finds set of pivot objects. These objects are used to create voronoi diagram. Pivots are selected using random selection, farthest selection (The first pivot is selected at random, and the next pivot isthe point whose smallest distance to existing pivots is the maximum) and K-means selection (first clusters then pivots are selected).

The first MapReduce job takes the selected pivots and datasets R and S as inputs. It consists of a single map phase. It finds nearest pivot for each object in RÈS and computes the distance between object and pivot. It also collects some metadata about each partition. The result is the partitioning of R based on pivots in the voronoi diagram. In second MapReduce job the Mappers find subset of S for each subset of R based on metadata collected in the first MapReduce job. Finally, each reducer performs kNN join between a pair of (R, S) mappers.

D. *Mining Association Rules in Text Databases Using Multipass with Inverted Hashing and Pruning*

In this paper, we propose a new algorithm named Multipass with Inverted Hashing and Pruning (MIHP) for mining association rules between words in text databases. The characteristics of text databases are quite different from those of retail transaction databases, and existing mining algorithms cannot handle text databases efficiently because of the large number of itemsets (i.e., words) that need to be counted.The new Multipass with Inverted Hashing and Pruning (MIHP) Algorithm is a combination of the Multipass-Aprirori (M-Apriori) algorithm and the Inverted Hashing and Pruning (IHP) algorithm. Due to M-Apriori, we can reduce the required memory space by partitioning the frequent l-itemsets and processing each partition separately. At the same time, by using IHP we can prune some of the candidate itemsets generated for each pass efficiently.
Multipass-Apriori (M-Apriori) Algorithm:

The Multipass-Apriori (M-Apriori) algorithm for mining association rules is direct descendent of the Apriori Algorithm. Apriori is not suitable for mining frequent itemsets in text databases because of the high memory space requirement for counting the occurrences of large number of potential frequent itemsets.The Multipass approach directly reduces the required memory space by partitioning the frequent 1-itemsets, and processes each partition separately. Each partition of items contains a fraction of the set of all items in the database, so that the memory space required for counting the occurrences of the sets of items within a partition will be much less than the case of counting the occurrences of the sets of all the items in the database.In practice, if the estimated number of candidate itemsets to be generated is small after processing a certain number of partitions, then we can merge the remaining partitions into a single partition, so that the number of database scannings will be reduced.
Inverted Hashing and Pruning (IHP) Algorithm:

Inverted Hashing and Pruning (IHP) is analogous to the Direct Hashing and Pruning (DHP) in the sense that both use hash tables to prune some of the candidate itemsets. In IHP, for each item, the transaction identifiers (TID) of the transactions that contain the item are hashed to a hash table associated with the item, which is named TID Hash Table (THT) of the item. When an item occurs in a transaction, the TID of this transaction is hashed to an entry of the THT of the item, and the entry stores the number of transactions whose TIDs are hashed to that entry. Thus, the THT of each item can be generated as we count the occurrences of each item during the first pass on the database. After the first pass, we can remove the THTs of the items which are not contained in the set of frequent l-itemsets and the THTs of the frequent l-itemsets can be used to prune some of the candidate2-itemsets. In general, after each pass k>=1, we can remove the THT of each item that is not a member of any frequent k-itemset, and the remaining THTs can prune some of the candidate (k + 1)-itemsets. In general, for a candidate k-itemset, we can estimate its maximum possible support count by adding the minimum count of the k items at each entry of their THTs. If the maximum possible support count is less than the required minimum support, it is pruned from the candidate set.

E. FiDoop-DP: Data Partitioning in Frequent Itemset Mining on Hadoop Clusters

Sequential FIM algorithms running on a single machine suffer from performance deterioration due to limited computational and storage resources which gives rise to a pressing need for the development of parallel FIM techniques. The correlation between the data is often ignored which will lead to poor data locality and the data shuffling costs and the network overhead will increase. FiDoop-DP is a parallel FIM technique in which a large dataset is partitioned across the HaDoop cluster's data nodes in a way to improve data locality. The goal of FiDoop-DP is to enhance the performance of parallel Frequent Itemset Mining on HaDoop Clusters.

Incorporating the similarity metric and the Locality-Sensitive Hashing technique, FiDoop-DP places highly similar transactions into a data partition to improve locality without creating an excessive number of redundant transactions. We design an efficient data partitioning scheme, which facilitates an analysis of correlations among transactions to reduce network and computing load. Our scheme prevents transactions from being repeatedly transmitted across multiple nodes. We implement the above data partitioning scheme by integrating Voronoi-diagram with LSH (Locality-Sensitive Hashing). To validate the effectiveness of our approach, we develop the FiDoop-DP prototype, where the data partitioning scheme is applied to a Hadoop-based FP Growth algorithm.

In most existing parallel FP-Growth algorithms, the MapReduce job is the most performance critical and time consuming. Previous experiment al results suggest that local FP-Growth cost accounts for more than 50% of the overall mining time. The grouping strategy plays the most important role in effecting subsequent data partitioning and local FP-Growth performance. Hence, existing data partitioning algorithms that performed prior to the parallel mining process are inadequate for solving the problem of redundant tasks.

F. FiDoop: Parallel Mining of Frequent Itemsets Using MapReduce

To achieve compressed storage and avoid building conditional pattern bases (CPB), FiDoop incorporates frequent items ultrametric tree (FIUT) rather than conventional FP-trees. In FiDoop, three MapReduce jobs are implemented to complete the mining task. In the crucial third MapReduce job, the mappers independently decompose itemsets, the reducers perform combination operations by constructing small ultrametric trees, and the actual mining of these trees separately, i.e., in parallel. Data partitioning and distribution are critical issues in FiDoop. The main idea of FP-growth is projecting database into a compact DS and then using divide and conquer method to extract frequent itemsets. The main bottleneck of FP-growth is the recursive traverse of FP-tree and that the construction of large number of conditional FP-trees is done residing in main memory.

The advantages of FIUT are: reduces I/O overhead, offers natural way of partitioning a dataset, compressed storage, averting recursive traverse. Frequent Items Ultrametric Tree (FIUT) is constructed by inserting all itemsets as its paths. Each leaf has item_name and support_count (i.e., number of paths ending with that item). Non-leaf node has item_name and link.

III.CONCLUSION

In this paper we have summarized the different techniques used for Frequent Itemset Mining. First, we discussed about the MR- Apriori based approach using MapReduce which is used for finding Frequent Itemsets. It uses the Apriori property to deduce the Frequent Itemsets. Then we went through an explanation on construction of FP-tree and extracting frequent itemsets using balanced FP- growth algorithm with MapReduce. Here we use conditional pattern bases to mine frequent itemsets using FP-trees. Later we looked into K-means clustering algorithm which helps to detect the closest neighbours. It uses the Voronoi diagram based data partitioning technique. Inverted hashing and

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

pruning which is one of the main steps in MapReduce is explained. It is best suitable for text data bases due to counting of large number of potential frequent itemsets. FiDoop-DP gives us an insight on how partitioning the data efficiently helps improve the performance of FIM. It uses the Locality sensitive Hashing technique to place the related transactions into the same data partition thus improving the data locality, reducing shuffling cost, avoiding duplicate transactions and greatly enhanching the performance of the FIM algorithm. Literature survey on all the above papers, has lead us to FiDoop which uses above methods to achieve compressed storage and avoid building conditional pattern bases.

REFERENCES

[1] X. Lin, “Mr-apriori: Association rules algorithm based on mapreduce,” in Software Engineering and Service Science (ICSESS), 2014 5thIEEE International Conference on. IEEE, 2014, pp. 141–144.
[2] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Huang, and S. Feng, “Balanced parallel fp-growth with mapreduce,” in Information Computing andTelecommunications (YC-ICT), 2010 IEEE Youth Conference on. IEEE, 2010, pp. 243–246.
[3] Y. Xun, J. Zhang, and X. Qin, “Fidoop: Parallel mining of frequent itemsets using mapreduce,” IEEE Transactions on Systems, Man, andCybernetics: Systems, doi: 10.1109/TSMC.2015.2437327, 2015.
[4] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, “An efficient k-means clustering algorithm:Analysis and implementation,” Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 24, no. 7, pp. 881–892, 2002.
[5] YalingXun, Jifu Zhang, Xiao Qin, Senior Member, IEEE, and Xujun Zhao, “FiDoop-DP: Data Partitioning in Frequent ItemsetMining on Hadoop Clusters,” IEEE Transactions on parallel and distributed systems, vol. 28, no. 1, January 2017, pp. 101-114.
[6]John D. Holt,and Soon M. Chung, “Mining Association Rules in Text Databases Using Multipass with
Inverted Hashing and Pruning,” 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI’02), 2002.