Mining Frequent Item Sets Using Map Reduce Paradigm

R.C.SARITHA1, M.USHA RANI2,
1Research Scholar, 2Associate Professor
1,2Department of Computer Applications, Sri Padmavathi Mahila Viswavidyalayam, Tirupati
1saritharc@yahoo.com, 2musha_rohan@yahoo.co.in

ABSTRACT

In Text categorization techniques like Text classification or clustering, finding frequent item sets is an acquainted method in the current research trends. Even though finding frequent item sets using Apriori algorithm is a widespread method, later DHP, partitioning, sampling, DIC, Eclat, FP-growth, H-mine algorithms were shown better performance than Apriori in standalone systems. In real scenario, as the data over the internet is expanding regularly, the unstructured data documents are scaling up and the existing computing resources in a single machine may not be sufficient to support the big data in text mining process. In order to handle big data, we need to parallelize the execution of the text mining process. Recently, Hadoop implements a computational paradigm named MapReduce, where the application is divided into many small fragments of work, each of which may be executed or re-executed on any node in the cluster. In addition, it provides a distributed file system that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster and enables the applications to work with thousands of computation-independent computers and petabytes of data. In this paper, we present about the characteristics of map reduce paradigm and shows the experimental results of finding frequent items using map reduce paradigm.

Keywords: Text Mining, Big data, frequent itemsets, Mapreduce paradigm, Hadoop, HDFS

1. INTRODUCTION

Text mining[1], also known as text data mining or knowledge discovery from textual databases[2], refers generally to the process of extracting interesting and non-trivial patterns or knowledge from unstructured text documents. Text mining is an interdisciplinary field which draws information on information retrieval, data mining, machine learning, statistics and computational linguistics. Information retrieval [3] is the activity of obtaining information relevant to an information need from a collection of information resources. The categorization techniques used in text mining are Classification and Clustering methods and are supervised and unsupervised respectively. In both the techniques, finding the frequent item sets became a basic part of the mining process.

In real scenario, as the data over the internet is expanding regularly, the unstructured data documents are scaling up and the existing computing resources in single machine may not be sufficient to support the text mining process. Hence, we need parallelism to execute the text mining process efficiently. Message-passing and shared memory threads are traditional parallel programming techniques used by the developers and are very cumbersome. The programmer should manage the concurrency explicitly by creating threads and synchronizing them through messages or locks and also the location of data. So, it is very cumbersome for the programmer to write scalable and correct parallel code for non-trivial algorithms.

With datasets growing larger and larger, Hadoop has become the solitary solution to address the above issues and enables the development of reliable, scalable, efficient, economical and distributed computing on large data sets for little cost. Hadoop with MapReduce programming model is familiar in distributed and parallel programming applications and in this paper it is used to find frequent item sets from text documents.

2. FREQUENT ITEMSETS GENERATION METHODS

The documents under the same topic share the common itemsets than those under different topics. Itemsets are found based on word presence not on TF or IDF. An itemset is frequent if its support is greater than or equal to user specified minimum support value. An association mining algorithm generates frequent patterns and then makes association rules satisfying a minimum support. Traditionally, the frequent itemsets can be found using Apriori algorithm. Apriori approaches [4,5] were usually used based on the downward closure property, i.e., if any itemset of length k is not frequent then its superset itemsets are also not frequent. Using this characteristic, Apriori based algorithms prune candidate itemsets. However, Apriori based algorithms need to generate and test all candidates. Moreover, they must repeatedly scan a large amount of the original database in order to check if a candidate is frequent or not. This is inefficient and ineffective. To overcome this problem, pattern growth based approaches FP-Growth [6,7, 8, 9] were developed, where FP-tree based methods mine the complete set of frequent patterns using a divide and conquer method to reduce the search space without generating all the candidates. A data structure Binary String is used to describe the database in an algorithm BitApriori[10] and then performed bitwise AND operation on the binary strings in order to implement the support count.
2.1 Maximal and closed frequent itemsets:
Maximal and closed frequent itemsets are compact representation of frequent itemsets which is useful to identify a small representative set of itemsets from which all other frequent itemsets can be derived. An itemset is a closed itemset if none of its immediate supersets has exactly the same support count and its support is greater than or equal to minsup. CLOSET [10], is an extension of the FP-growth algorithm [11], which constructs a frequent pattern tree FP-tree and recursively builds conditional FP-trees in a bottom-up tree search manner.

Another method of finding frequent itemsets is the vertical representation of the data sets. Carpenter[12] closed frequent itemset algorithm, in which each item set can be represented as the intersection of some subset of datasets. Both MAFIA [13] and CHAR[14] use a vertical representation of the datasets. CHAR enumerates closed itemsets using a dual itemset-tidset search tree and adopts the Diffset technique to reduce the size of the intermediate tidsets. The most costly operation for the algorithms using vertical format is the intersection on tidsets. CHAR shows better performance than A-close, Pascal, MAFIA, and CLOSET in many dense datasets. Touch[15] uses vertical representation and deals with computation of both CHAR frequent closed itemsets and Talky-G a frequent generator itemsets.
Pincer Search, MAX miner, Depth-Project, Mafia, Smart Miner, FPmax algorithms are well known for finding maximal frequent itemsets. MAFIA is mainly designed for mining maximal itemsets, but it has an option to mine closed itemsets. One of its main features of MAFIA is the compressed vertical bitmap structure.

3. COMPUTING ON THE CLOUD
3.1 Cloud computing:
The definition of Cloud Computing comes from the National Institute of Standards and Technology (NIST) [16]. The NIST definition says that Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. Cloud computing is a construct that allows you to access applications that actually reside at a location other than your computer or other internet enabled device most often that will be a data center. A cloud computing solution is made up of several elements clients, data centers and distributed servers. The characteristics of cloud computing are:
1) On-demand self-service which is an ability for an end user to sign up and receive services without the long delays.
2) Broad network access which is to access the service across standard platforms desktop, laptop, mobile etc.
3) Resource pooling where resources are pooled across multiple customers
4) Rapid elasticity which have capability to scale to cope with demand peaks
5) Measured Service is a utility service where billing is metered and delivered

3.2 Map reduce Paradigm:
Map reduce paradigm is referred to as both a programming model for Bulk synchronous Parallel Processing[17] as well as computational infrastructure for implementing this model. Typically map reduce job has three phases:
Map: In this phase, a user defined function (UDF) also called Map, is executed on each record in a given file. The file typically spread across many computers, and many processors (called mappers) work on the file in parallel. The output of each call to map is a list of <key, Value> pairs.
Shuffle: This phase is hidden from the programmer and the <key, Value> pairs are sent to another set of computers, such that all <key, Value> pairs with same key will go to the same computer, and independent of other keys. At each destination computer <key, Value> pairs with the same key are aggregate together. For example: if <x, y1>,<x, y2>,<x, y3>......<x, yk> are all the key-value pairs produced by Mappers with the same key, at the destination computer for key x, these get aggregated into a large <key, value> pair, <x, {y1,y2,y3.....yk}>. There is no order or sequence guaranteed. The aggregated <key, value> pair is typically called a Reduce record, and its key is referred to as the Reduce Key.
Reduce: In this phase, A UDF also called Reduce, is applied to each Reduce Record often by many parallel processors. Each process is called a Reducer. For each invocation of Reduce, one or more records may written into a local output file.
Map reduce is a processing paradigm that build upon the principles as follows:
a. Key underlying concept “divide and conquer”, where a single program is broken down into multiple individual subtasks.
b. Provides a set of series of transformations from a source to a result dataset. The developer only defines the data transformations.
c. Hadoop MapReduce job manages the process of how to apply these transformations to the data across the cluster in parallel.

3.3 Hadoop: Hadoop is an open-source Cloud computing environment that implements the Googlem MapReduce framework in Java. Hadoop is created and maintained by the Apache project, licensed under the
Apache v2 license. The core parts of the Hadoop are Hadoop Distributed File System (HDFS) and MapReduce. HDFS is a virtual file system which is similar to file system except that when you move a file on HDFS, the file is split into many small files, each of those files is replicated and stored on (usually, may be customized) 3 servers for fault tolerance constraints. Large Datasets on the cloud can be easily processed by writing massive parallel code by Hadoop MapReduce. Hadoop MapReduce divide the work into smaller chunks, processed the chunks concurrently and then combine the results of all chunks to obtain the final result. Unlike traditional relational databases that require structured data with well-defined schemas, MapReduce and Hadoop work best on semi structured or unstructured data.

3.4 Architecture

In a distributed environment Hadoop maintains cluster of nodes to run massive parallel code. In Hadoop cluster as shown in Fig:1 includes a single master and multiple slave nodes. A master node controls a group of slave nodes on which the Map and Reduce functions run in parallel. The master node consists of Namenode, Jobtracker,DataNode and TaskTracker and the slave node consists of DataNode and Tasktracker. The Namenode manages meta data and datanodes and coordinates the storage and retrieval of the individual data blocks managed by a data node. The secondary Namenode can generate snapshot’s of Namenodes’s memory structures and thus preventing file-system corruptions and reducing loss of data. The Job tracker and Task tracker are responsible for processing data. The JobTracker keep track of TaskTracker, the Job execution status and availability of TaskTrackers. The master node assigns a task to a slave node that has any empty task slot. Typically, computing nodes and storage nodes in a Hadoop cluster are identical from the hardware's perspective. Such a homogeneous configuration of Hadoop allows the MapReduce framework to effectively schedule computing tasks on an array of storage nodes where data file are residing, leading to a high aggregate bandwidth across the entire Hadoop cluster. An input file passed to Map functions resides on the Hadoop distributed file system on a cluster. Hadoop's HDFS splits the input file into even-sized fragments, which are distributed to a pool of slaves for further MapReduce processing. But some cases the distribution depends on hardware configuration.

![Fig:1 Map reduce and HDFS components in Hadoop](image)

4. FINDING FREQUENT ITEMSETS USING MAPREDUCE

In the above sections, the importance of map reduce paradigm for parallel and distributed environment is explained. Finding frequent itemsets using map reduce paradigm is an efficient approach for large datasets in document categorization process. To improve the performance of the Apriori like frequent itemset based algorithms, many parallelized techniques have been proposed. Three algorithms SPC, FPC, DPC [18] have been proposed to implement Apriori algorithm in map reduce framework, where DPC features in dynamically combining candidates of various lengths and outperforms both the straight forward algorithm SPC and fixed passes combined counting algorithm FPC. In order to filter the redundant and compress the itemsets, a parallelized AFOPT- close algorithm[19], in order to address the bottle necks of closed frequent itemsets mining in large datasets based on cloud computing.

5. PROPOSED SOLUTION

This paper proposed a new algorithm to represent the documents data in vertical format and find frequent itemsets using AND operations in map reduce framework. Scan the documents once and generate frequent 1-itemset and then transform the horizontal format of frequent 1 itemset into vertical format. Apply "And
operation among the each element in the itemsets and record the result. If the result is more than min_sup then candidate set is generated else we do the next "AND operation". The "AND operations" will stop, if the result of "AND operation" is less than min_sup or frequent itemset left and not able to "AND operation". From this algorithm, we scan the documents only once which is efficient when compared to Apriori and FPgrowth. The pseudo code of the new algorithm is as follows:

**Method Map(key,value,context)**

- Get the Document name and path from the context
- For each word belongs to document
  - If the word is digit or stopword or contains "_": continue
  - Find the root stem of the word
- Emit(word, Document name)

**Method Reduce(key,values)**

- For each val in the values
  - Find unique Document list without repetition of document names
- Emit(key, Document list)

**Fig: 2 Frequent 1-item list generation map reduce algorithm**

6. **EXPERIMENTAL STUDY**

A set of 100 Text Documents is taken as input data set for the above said algorithm Fig: 2. The algorithm generated frequent 1-itemsets from the document collection in the vertical format. Some of the frequent 1-itemsets generated are shown in below Table 1:

<table>
<thead>
<tr>
<th>Frequent Itemsets</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>abov</td>
<td>crude7.txt,bop14.txt,crude4.txt,money-fx3.txt,money-fx8.txt</td>
</tr>
<tr>
<td>abroad</td>
<td>trade3.txt,cpi14.txt</td>
</tr>
<tr>
<td>absenc</td>
<td>money-fx2.txt</td>
</tr>
<tr>
<td>absorb</td>
<td>cpi6.txt</td>
</tr>
<tr>
<td>absorpt</td>
<td>crude7.txt</td>
</tr>
<tr>
<td>acceler</td>
<td>cpi6.txt,bop14.txt</td>
</tr>
<tr>
<td>access</td>
<td>trade9.txt,trade3.txt,trade6.txt</td>
</tr>
<tr>
<td>accomod</td>
<td>coffee14.txt</td>
</tr>
<tr>
<td>accomod</td>
<td>crude11.txt</td>
</tr>
<tr>
<td>accomplish</td>
<td>trade6.txt,trade3.txt</td>
</tr>
<tr>
<td>accumul</td>
<td>cpi2.txt</td>
</tr>
<tr>
<td>accur</td>
<td>cocoa3.txt</td>
</tr>
<tr>
<td>accus</td>
<td>cocoa1.txt</td>
</tr>
<tr>
<td>achiev</td>
<td>cpi7.txt,cpi14.txt,coffee13.txt</td>
</tr>
<tr>
<td>acid</td>
<td>trade5.txt</td>
</tr>
<tr>
<td>acquir</td>
<td>acq2.txt,acq10.txt,crude7.txt,acq4.txt,earn10.txt</td>
</tr>
</tbody>
</table>

**Table 1 : Frequent 1-itemsets generated using map reduce proposed algorithm**

Similarly, 2-Frequent itemsets, 3-itemsets and so on can be generated by the proposed solution without scanning the entire document collection.

**CONCLUSIONS**

The cutting edge technology in parallel and distributed environment is the map reduce paradigm to run parallel massive datasets. The existing methods in the generation of frequent items need to scan the database more than once in Apriori algorithm using map reduce. The Proposed algorithm scan the document collection only once and generate frequent items and is efficient than other existing algorithms using map reduce.

**REFERENCES**


[6] Jiawei Han, Jian Pei, Yiwen Yin, Mining frequent patterns without candidate generation, ACM SIGMOD, May 2000.


[8] Jian Pei, Jiawei Han, CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets, DMKD, May 2000.

[9] Jianyong Wang, Jiawei Han, Jian Pei, CLOSET+: searching for the best strategies for mining frequent closed itemsets, ACM SIGKDD, Aug 2003.


