*International Journal of Engineering Applied Sciences and Technology, 2016*
**Vol. 1, Issue 6, ISSN No. 2455-2143, Pages 188-191**
Published Online April - May 2016 in IJEAST (http://www.ijeast.com)

# FREQUENT SUBGRAPH EXTRACTION BASED ON MAP REDUCE

Cheetirala Sudhakar, Chandini Sasanapuri, Swapna Priya, Pavan Kumar N
Asst. Professor, Department of CSE
Vignan IIT, Vizag, Andhra Pradesh, India

***ABSTRACT:*** **Frequent sub graph extraction from a large number of small graphs is a primitive operation for many data mining applications. To extract frequent subgraphs, existing techniques need to enumerate a large number of subgraphs which is super linear with the cardinality of the dataset. Given the rapid growing volume of graph data, it is difficult to perform the frequent subgraph extraction on a centralized machine efficiently. So, there is a need to investigate how to efficiently perform this extraction over very large datasets using MapReduce. Parallelizing existing techniques directly using MapReduce does not yield good performance as it is difficult to balance the workload among the compute nodes. This framework adopts the MRFSE strategy to iteratively extract frequent subgraphs, i.e., all frequent size-(i+1) subgraphs are generated based on frequent size-i subgraphs at the $i^{th}$ iteration using a single MapReduce job. To efficiently extract frequent subgraphs, preparation and mining phase are used which includes isomorphism testing to eliminate duplicate patterns. Frequent subgraphs extraction can be done efficiently and efficiently by using a distributed environment named Hadoop MapReduce framework.**

***Keywords: Map Reduce, Frequent sub graph, Feature extraction.***

## I. INTRODUCTION

Graph mining is a process of finding new, unknown, interesting, useful and understandable patterns from a large volume of data. One important aspect in graph mining is concerned with the discovery of frequent subgraphs extraction from a large volume of data. Frequent subgraphs extraction helps in wide range of applications like identifying the relationship between chemical compounds and building graph indexes. Frequent subgraph extraction is used in various other disciplines like social networks, bioinformatics and semantic web etc.

MapReduce is a programming model introduced by Google in 2004 which performs distributed computing on large volumes of data in parallel mode on large number of clusters of commodity hardware. MapReduce solves the issues like data distribution and load balancing, fault tolerance, parallelization and makes the user concentrate on problem solving rather than worrying about these internal issues. MapReduce provides data distribution and load balancing by splitting entire volume of data into equal sized blocks. Replication of data is also provided in MapReduce to avoid loss of data. MapReduce distributes these equal sized blocks to all the nodes and makes the users free from focusing on load balancing and data distribution. MapReduce re-performs crashed tasks and provides fault tolerance. In MapReduce framework, throughput is increased by assigning uncompleted tasks of slower nodes to the idle nodes.

Hadoop is an open source implementation of Google MapReduce architecture provided by Apache Software Foundation. This architecture uses Hadoop Distributed File System (HDFS) for efficiently processing huge volumes of data parallel on large clusters of commodity hardware in a reliable manner. So, all the sequential algorithms need to be redesigned into parallel computing algorithms to execute efficiently on Hadoop MapReduce framework.

## II. LITERATURE SURVEY

Frequent subgraph mining task is to discover all subset of graphs which occur repeatedly. gSpan, Mofa, FFSM, Gston are some of the most extensively used algorithms for frequent subgraph extraction. The algorithms use a multiple-pass generation-and-test method to generate the candidate (k+l) subgraph from the frequent k-subgraphs. As we mentioned earlier, the sequential algorithms performance is inefficient, especially when the data sets volume grows towards a terabyte or petabytes of data. Therefore, parallel algorithms were proposed. However, parallel mining algorithms reveals new problems that did not exist in sequential computing, such as workload balancing, data partitioning and distribution, jobs assignment, and parameters passing between nodes. Thus, significant time and effort are required to solve these problems. MapReduce is a framework that takes care of these

*International Journal of Engineering Applied Sciences and Technology, 2016*
**Vol. 1, Issue 6, ISSN No. 2455-2143, Pages 188-191**
Published Online April - May 2016 in IJEAST (http://www.ijeast.com)

internal issues. Because of these benefits of the MapReduce model, MRFSE algorithm using Hadoop MapReduce model is implemented. Most of the real-life application data sets can be denoted by direct or un-direct graphs. Numerous new classes of applications such as Social networks, computer networks, Cyber-security, mobile call networks, Protein-Protein regulation networks, the World Wide Web can be represented by a graph and it can be handled in parallel. For example, in Cyber-security, a network traffic dataset can be modelled as a graph where vertices represent IP addresses and edges are typed by classes of network traffic.

## III. PROBLEM DEFINITION

Let, G ={$G_1$,$G_2$,...,$G_n$} be a graph database, where each $G_i$ ϵ G, where i ={1….n} represents a labelled, undirected and connected graph. The size of graph is the number of edges it has. Now, t(g)= { $G_i$ : g is a subset of $G_i$ belongs to G; where i={1…..n} is the support-set of the graph g. Thus, t(g) contains all the graphs in G that has a subgraph isomorphic to g. The cardinality of the support-set is called the support of g. g is called frequent if support ≥ minimum support, where minimum support is minimum support threshold given as input. The set of frequent subgraphs are represented by F. Based on the size of a frequent subgraph, we can partition F into a several disjoint sets, $F_i$ such that each of the $F_i$ contains frequent subgraphs of size i only.

The generation of frequent subgraphs is not possible using a single MapReduce job. This can be made possible by using iterative MapReduce. Here mapper and reducer functions are executed in an iterative manner. Iterative MapReduce is defined as a multi staged execution of map and reduce phases in a cyclic manner, i.e. the output of the stage i reducer is given as an input to the stage i+1 mapper. An external condition is used to decide when to terminate the job.

## IV. ISSUES IN THE EXISTING SYSTEM

The existing algorithms execute sequentially to find all frequent subgraphs. Therefore, the waiting and the scheduling are pure overheads to the mining task. So, some distributed environment like Hadoop MapReduce can be used to solve this problem. The algorithms that previously existed requires one iterative MapReduce phase to find all frequent subgraphs.

All the edges which are having support less than minimum support are used in the iterative MapReduce, but could not generate subgraphs as the graph does not meets the threshold value. Also the number of isomorphism checking's increases and the amount of data generated in the map phase grows exponentially with the length of the transactions in the dataset. Therefore, its performance is inefficient. So, if infrequent edges are eliminated in the first stage, then the amount of data generated in the map and reduces phase's decreases. So, MRFSE framework with two maps and reduces phases is proposed. This can be done by adding one more map and reduce phase to generate frequent edges. These edges are used in iterative map and reduce phases to generate candidate subgraphs. Dynamic partition of data generated after execution is one of the issue in Hadoop MapReduce framework.

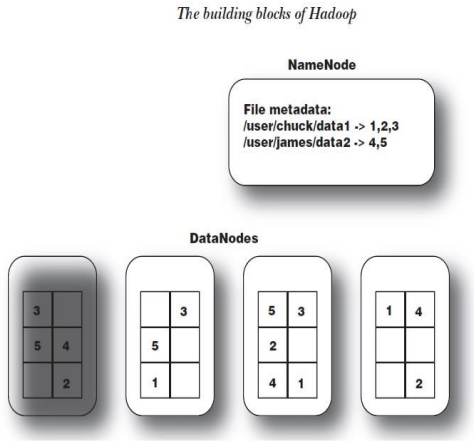



Figure 2.1: Replication of data in NameNode and DataNode

**Job Tracker**: Job Tracker executes the Map Reduce job by assigning tasks to the Task Tracker.

**Task Tracker**: It executes the tasks assigned by Job Tracker. During execution, the Task Tracker frequently communicates with the Job Tracker. Otherwise, Job Tracker thinks that Task Tracker node has crashed and assigns the task to another Task Tracker node.

**Algorithms in MRFSE**
**Generation Phase**
**Mapper_Generation**

Input : (key, value) pair where key is offset and value is list of all graphs in that partition

Output : (key, value) pair where key is min-dfs code and value is a graph object

```
Mapper_generation(Long key, Text value)
Begin
Generate  level_one_occurence list(value)
Generate level_one_map(value)
P=get size_one_edges( )
forall P_i in P
Begin
 key_to_reducer = min-dfs(P_i)
 value_to_reduce r= object(p_i)
emit(key_to_reducer,  value_to_reducer)
End
```

*International Journal of Engineering Applied Sciences and Technology, 2016*
**Vol. 1, Issue 6, ISSN No. 2455-2143, Pages 188-191**
Published Online April - May 2016 in IJEAST (http://www.ijeast.com)

End

**Reducer_Generation**

Input : (key, value) pair where key is min-dfs code and value is a graph object.
Output : (key, value) pair where key is min-dfs code and value is a graph object

```
Reducer_generation(Long key, Text value)
Begin
forall Pi in P
Begin
 if length(Pi.OL) > minimum support
Begin
key=min-dfs (Pi)
value=object(Pi)
emit(key, value)
End
End
End
```

After this generation phase, all frequent edges are generated. These edges are given as input to mapper of Verification phase.

**Verification Phase**
**Mapper_verification**
Input : (key, value) pair where key is min-dfs code and value is a graph object.
Output : (key, value) pair where key is min-dfs code and value is a graph object.

```
Mapper_verification(Long key, Text value)
Begin
P = reconstruct_graphs(value)
reconstruct_all_datastructures(value)
P=candidate_generation(p)
forall Pi in P
Begin
        if pass_isomorphism(Pi) = true
        Begin
                if length(Pi.OL) > 0
                Begin
                        key_to_reducer = min-dfs(Pi)
                        value_to_reducer=object(Pi)
                        emit((key_to_reducer,
value_to_reducer)
                End
        End
End
End
```

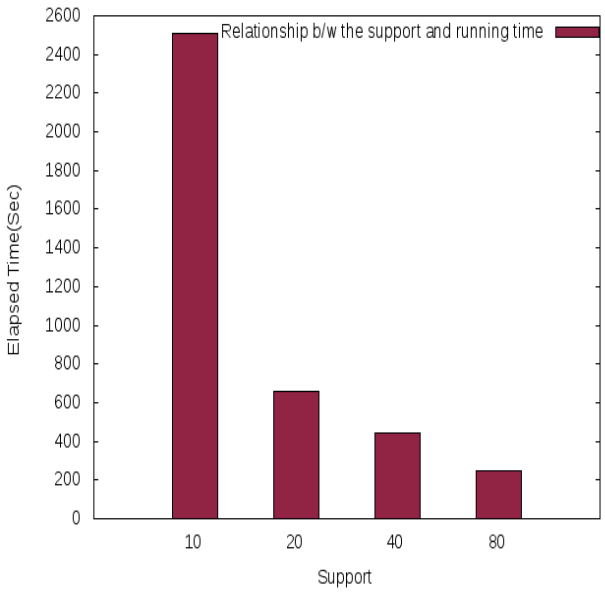
**Reducer_verification**

Input : (key, value) pair where key is min-dfs code and value is a graph object.
Output : (key, value) pair where key is min-dfs code and value is a support count.

```
Reducer_verification(Text key, Intwritable value)
Begin
        support= 0
        forall Pi in P
        Begin
                support= support + length(Pi.OL)
        End
        if support >= minimum support
        forall Pi in P
        Begin
                write(min-dfs(Pi), support) to HDFS
        End
End
```

## V. RESULTS

A real dataset containing a large number of graphs is taken as a data source from a dataset repository and is processed taking threshold (minimum support count) values as input.
The frequent subgraph extraction algorithm has been tested in Map Reduce environment (Pseudo Distributed Mode) taking another input as threshold value. For each threshold value, the number of subgraphs obtained and running time to generate frequent subgraphs in the MapReduce environment are noted. It is observed that in Map Reduce (Pseudo Distributed Mode) environment, with the increase in threshold value, the running decreases. Also the running time varies with the change in number of reducers. The experimental results are shown in the following figure.



*International Journal of Engineering Applied Sciences and Technology, 2016*
**Vol. 1, Issue 6, ISSN No. 2455-2143, Pages 188-191**
Published Online April - May 2016 in IJEAST (http://www.ijeast.com)

Fig 4.1 Relationship between support and running time

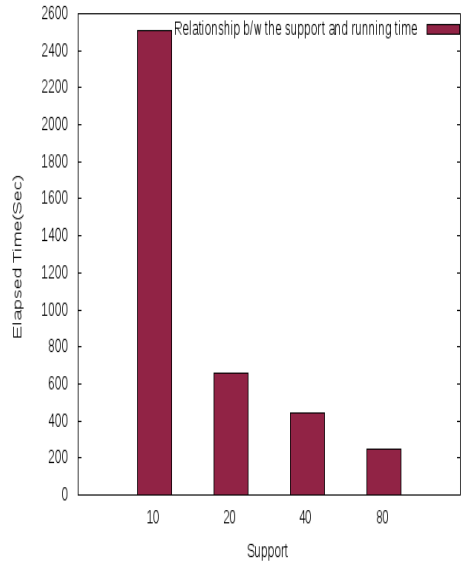



Fig 4.2 A Graph showing Relationship between elapsed time and support.

## VI. CONCLUSION

As a parallel programming model, MapReduce is one of the most important techniques for mining large volumes of data on large number of clusters. MRFSE is an algorithm which uses iterative MapReduce to generate frequent subgraphs. It extracts the frequent subgraphs that are present with in a graph database. A subgraph is said to be frequent if its count is greater than minimum support. MRFSE uses two MapReduce functions. One MapReduce is used to identify frequent edges and another is used to generate candidate subgraphs in an iterative manner. The elapsed time for different support value is recorded. The number of subgraphs for a particular support value is also recorded. The entire algorithm is executed in a single cluster. For good and effective results, we can implement this algorithm on a group of clusters. The execution time decreases with the increase in the number of clusters.

## VII. FUTURE WORK

As Frequent Subgraph Mining has a major amount of research attention, many frequent subgraph mining algorithms have been proposed in the past decades. However, the enlarging data in applications makes frequent subgraph mining of very large volume of data is a challenging task. Here the iterative frequent subgraph extraction algorithm based on MapReduce, takes advantage of MapReduce's parallel computation capability to make the algorithm accelerated. And as the number of nodes involved in the computation can be dynamically changed, it makes the method with high scalability. Frequent Sub graph Extraction based on Map Reduce implemented in Hadoop fully distributed mode will give better efficient results than in pseudo distributed mode for large data sets. Embeddings means list of all edges that are adjacent to a sub graph. Embeddings can be used to perform isomorphism testing to eliminate duplicate subgraphs. It may reduce the time taken for isomorphism testing.

## VIII. REFERENCE

[1] J.Han. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[2] A.Inokuchi, T.Washio, and H.Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In PKDD, pages 13–23, 2000.

[3] M.Kuramochi and G.Karypis. Frequent subgraph discovery. In ICDM, pages 313–320, 2001.

[4] X.Yan and J.Han. gspan: Graph-based substructure pattern mining. In ICDM, pages 721–724, 2002.

[5] J.Cheng, Y.Ke, and W.Ng. Efficient query processing on graph databases. ACM Trans. Database Syst., 34(1), 2009.

[6] J.Dean and S.Ghemawat, "Mapreduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, pp. 107–113, 2008.

[7] F. Afrati, D.Fotakis, and J.Ullman, "Enumerating subgraph instances using map-reduce," in Proc. IEEE 29th Int. Conf. Data Eng., Apr. 2013, pp. 62–73.

[8] Agrawal R., & Shafer, J.C. (1996). Parallel mining of association rules.Knowledge and Data Engineering, IEEE Transactions on, 8(6), 962-969.

[9] R.Agrawal and R.Srikant, "Fast algorithms for mining association rules in large databases," in Proc. 20th Int. Conf. Very Large Data Bases, 1994, pp. 487–499.

[10] J.Huan, W.Wang, J.Prins, and J.Yang. Spin: mining maximal frequent subgraphs from graph databases. In KDD, pages 581–586, 2004.