International Journal of Multimedia and Ubiquitous Engineering
Vol.15, No.1 (2020), pp.27-34
http://dx.doi.org/10.21742/ijmue.2020.15.1.03

Map-Reduce based Frequent Sub-Graph Extraction

Ch. Sudhakar[1], A.Siva Pavan[2], N. Thirupathi Rao[3], Debnath Bhattacharyya[4]

[1]Assistant Professor, CSE,
[2]III B.Tech CSE, [3]Associate Professor, [4]Professor
Vignan's Institute of Information Technology, Visakhapatnam. India
[1]sudhakar.cheetirala@gmail.com, [3]nakkathiru@gmail.com

Abstract

Frequent subgraph extraction from a substantial number of small graphs is a crude activity for some, information mining applications. To extricate frequent subgraphs, existing systems need to identify countless which is super straight with the cardinality of the dataset. Given the huge developing volume of graph information, it is hard to play out the regular subgraph extraction on a unified machine proficiently. Along these lines, there is a need to explore how to effectively play out this extraction over expansive datasets utilizing MapReduce. Parallelizing existing strategies straightforwardly utilizing MapReduce does not yield great execution as it is hard to adjust the remaining task at hand among the figure hubs. This structure receives the MRFSE procedure to iteratively remove Frequent subgraphs, i.e., all incessant size-(i+1) subgraphs are created dependent on continuous size-I subgraphs at the ith emphasis utilizing a solitary MapReduce work. To productively separate successive subgraphs, arrangement and mining stage are utilized which incorporates isomorphism testing to wipe out copy designs. Frequent subgraphs extraction should be possible productively and effectively by utilizing a disseminated domain named Hadoop MapReduce structure.

Keywords: *MRFSE, Map reduce, Hadoop, Sub graph*

1. Introduction

With the advent in the technology and the exponential growth in volume of the data made more difficult to extract them on a stand-alone computer. So, frequent subgraph mining algorithms need to be parallelized by using some distributed environment in order to speed up the mining task irrespective of increase in volume of data [1][2].

Even if the performance increases due to parallelization of mining algorithms, several internal issues were identified regarding data partition and distribution, load balancing, job assignment and monitoring, parameter passing between nodes etc. To overcome these internal issues, MapReduce framework has been introduced.

The entire input dataset is given as input to preparation phase. In preparation phase, all the infrequent edges are eliminated and only frequent edges, ie, the output of reducer is passed as input to the verification phase. Preparation map and reduce functions executes only once and verification map and reduce functions executes in an iterative manner. In the first phase, frequent edges are generated and in the second phase, candidate subgraphs of size-1, size-2.

Article history:
Received (January 25, 2020), Review Result (March 2, 2020), Accepted (April 11, 2020)

Copyright ⓒ 2020 Global Vision Press (GV Press)

Size-n are generated in n iterations. Min-dfs method is used to identify redundant edges and redundant subgraphs. Some data structures are used to store occurrences list, edge extensions, support count of subgraphs etc. This algorithm on Hadoop MapReduce gives efficient results [3].

2. Graph Computations using Hadoop MapReduce

The Graph computations are used in numerous scientific applications and massive graph computations are often used in many data-intensive scientific applications. The large graph computations involve building of sparse matrix based on adjacency graph which contain edge weights as per applications. The existing algorithms have restrictions to load the complete graph data and process in the memory of the system [4]. Typical graph computational algorithms assume that the graph fits in the memory of a system, and in the case of big data, the graph data is spanned across the multiple Giga to terabytes of data. The graph partitioning methods are employed to distribute the sub-graphs on multiple clusters which show orders of magnitude performance greater than traditional sequential algorithms.

The performance achieved for typical graph analytics depends upon the graph partitioning, communication across sub-graph boundary data and necessity of boundary information of sub-graphs for neighbor sub-graphs. Graph analytics is the process of going from raw data, to a graph, i.e., building, list of vertices, weighed edges, edges, boundary vertices and edges at the interface of each sub-graph, adjacency, subgraph and perform analysis. This involve, applying graph algorithms such as coloring a graph, graph partitioning, analyzing the result, and then potentially repeating the process with each sub-graph. Several algorithms for sub-graph detection have been implemented. After graph partitioning is performed, the computations can be performed in parallel. The real-world graphs such as web graphs, social networks and protein interaction graphs have complex relationships among vertices and edges and they cannot be decomposed to sub-graphs and process them in parallel [5].

Several authors developed algorithms for parallelization of extracting subgraphs from a large number of graphs in a data set. These algorithms involve unstructured communications and load balancing is the significant issue in implementation. The first step is to use decomposition techniques which recognize the concurrency. Achieving load balancing is another main issue in generating subgraphs. The algorithms such as coloring a sparse graph, parallel graph partitioning algorithms, and parallel search algorithms require dynamic load balancing which makes the algorithm more complicated.

The existing algorithms have restrictions to load the complete graph data and process in the memory of the system. The graph data such as vertices, adjacency, edges, and weights associated with vertices and edges need parallel processing capabilities. The multi-threading model in shared memory programming environment to process sub-graphs in parallel have been used. Several restrictions exist to process large graph data on shared address space platforms for performance and scalability analysis for graph analytics. The graph partitioning methods are employed to distribute the graph on to multiple clusters which show orders of magnitude performance greater than traditional sequential algorithms. To distribute each graph on numerous processors of a distributed computing system, Hadoop MapReduce programming environment can be used to process the data in parallel.

3. Hadoop Framework

In recent years, if users want to store and analyze data, they would store the data in a database and analyze data using SQL (Structured Query Language). With the rapid increase in technology, data obtained is of large size and is in semi-structured and unstructured format. So it is not possible to store unstructured data into a schema and process it. To handle this problem, Hadoop framework is introduced [6].

Hadoop is a MapReduce based distributed processing framework for processing extremely distributable problems across huge data sets. Several researchers widely worked on Hadoop MapReduce framework to solve large-scale sequence analysis applications using low-end multi-core processor nodes [7]. Most of the applications written using Hadoop MapReduce need large number of Parallel I/O operations in a distributed computing environment. Several open source software based on Hadoop MapReduce such as Neo4j, Giraph, Pegasus, Pregel, and GraphLab are used in large scale graph analytics. The objective is to analyse the performance for unstructured type of graph computations based on Hadoop Map Reduce programming environment in a BIG Data framework.

The key features present in Hadoop that differentiate with other frameworks are

Simple: Hadoop framework helps in writing and executing parallel code efficiently.

Accessibility: Hadoop executes on large clusters of commodity systems.

Scalable: Hadoop processes large volumes of data by accumulating additional nodes to the cluster.

Robust: Frequent hardware malfunctions may occur as Hadoop is running on commodity hardware. Hadoop is architected in such a way to handle these problems.

Hadoop Installation can be done in two types. They are

Fully Distributed Mode: This is the actual setup that runs of multiple machines. Hadoop configuration consists of five types of nodes. They are Name Node, Secondary Name Node, Data Node, Job Tracker and Task Tracker. Name node and data nodes are nodes of HDFS and they access Hadoop Distributed File System (HDFS). Job Tracker manages all the Map Reduce jobs. Task Tracker executes the tasks which are a part of Map Reduce jobs [8]. Hadoop users submit Map Reduce jobs to the Job Tracker. Job Tracker assigns Map and Reduce jobs to the Task Tracker and collects back the results.

Name Node: Hadoop framework employs master/slave architecture for facilitating distributed storage and processing. It contains metadata of how the files are stored in different data nodes. It directs Data Node daemons in executing the I/O tasks.

Secondary Name Node: It is also similar to Name Node which contains metadata and directs the data nodes.

Data Node: Data Node acts as a slave node in HDFS. It works as per the directions of Name Node. All the files are stored in data blocks. All the blocks that need to be processed are present in data nodes. The default replication factor is 3. So, data blocks are replicated over 3 data nodes. Data Node regularly reports to Name Node about the data blocks it stored.

4. Frequent subgraph Selection

4.1. Isomorphism Checking

An applicant subgraph can be produced from different created subgraphs, yet just a single such subgraph is investigated amid the hopeful age step and the rest of the subgraphs are distinguished and along these lines disregarded. To distinguish invalid applicant age subgraphs, a graph mining algorithm needs to tackle the diagram isomorphism issue, as the copy duplicates of a competitor subgraphs are isomorphic to one another. There are different types of isomorphism checking schemes. Min-dfs-code is the isomorphism checking technique. According to this technique, min-dfs code is applied on all the generated subgraphs. If two or more subgraphs are identical then they have the same min-dfs code. Thus, only one generated subgraph is considered and remaining are considered as duplicates and hence ignored.

4.2 Minimum Support

Minimum support is a threshold value which differentiates frequent and infrequent sets. If the count of unique subgraph is less than minimum support, then those graphs are said to be infrequent graphs. If the count is greater than minimum support, then those are said to be frequent graphs. Minimum support is given as input to the algorithm for identifying the frequent subgraphs.

Map Reduce based Frequent Subgraph Extraction is implemented in Pseudo Distributed Mode in Hadoop MapReduce framework.

4.3. Phases in MRFSE

MRFSE generates frequent subgraphs in two phases. They are generation phase and verification phase. Here generation phase contains mapper and reducer and verification phase contains mapper and reducer that run iteratively.

- In the mapper of generation phase, all the individual edges are taken into consideration and count the number of occurrences of each individual edges. Now, eliminate all the edges that are less than minimum threshold (support) and remaining are considered as frequent edges and are used for generating candidate subgraphs.
- The mapper generates all the frequent size-1 subgraphs and these are used to generate size-2 subgraphs. This output of mapper is given as input to reducer of generation phase.
- The reducer writes the key and value pairs to HDFS.
- The verification phase generates candidate subgraphs and verifies whether redundant subgraphs are generated.
- This verification phase is processed iteratively to generated subgraphs of size-1, size-2, size-3…in successive iterations respectively.
- The mapper generates candidate subgraphs and performs isomorphism checking to check whether redundant subgraphs are obtained from same graph. All the frequent subgraphs are sending to the reducer phase of verification phase.
- The reducer counts such subgraphs and if the count is greater than minimum support, then those subgraphs are said to be frequent. These subgraphs with there support count is written into the HDFS.
- In the process of generating frequent subgraphs using MRFSE, some data structures are used. It stores all the possible extensions from a vertex considering all the edges that exists in the graphs. For example, if a graph has edges such as B-D, B-C, B-A and B-E. While generating candidate subgraphs, if B is an extension stub, then the vertices A, C, D and E are the vertices adjacent to B. This data is stored in the data structure named in all the graphs. Another type of data structure is Edge- Occurrence list which stores the graph ids of each and every edge. If B-E is an edge and if it is present in graphs G1, G3 and G5, then the Edge-Occurrence list contains G1, G3 and G5.

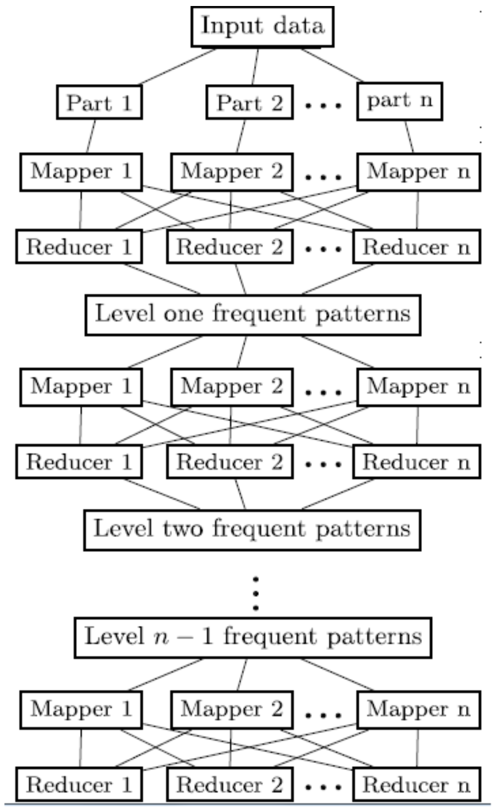
Input data
Part 1
Part 2
part n
Mapper 1
Mapper 2
Mapper n
Reducer 1
Reducer 2
Reducer n
Level one frequent patterns
Level two frequent patterns
Level $n-1$ frequent patterns

Figure 1. MRFSE Framework

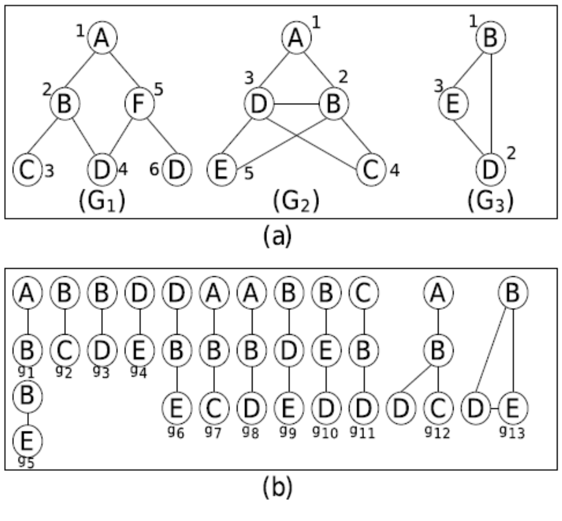

Figure 2. (a) Graph database with three graphs with labelled vertices (b) Frequent subgraphs of (a) with minimum support=2

In the above [Figure (a)], there are three graphs G1, G2 and G3 having some vertices and edges. [Figure (b)] contains all the subgraphs that are present at least in two graphs, i.e, minimum support=2. From these 3 graphs, 13 frequent subgraphs are generated. Mapper and Reducer works based on key and value pairs. They are Occurrence list, vertex set and support value. The output of mapper is given to the reducer. Here the key is min-dfs code and value contains the attributes. The reducer stores the results in HDFS. It contains the edges or min-dfs as key and the static data structures as value.

5. Results

It is seen that in Map Reduce platform, with the increased in threshold value, the running reductions. The test results appear in the accompanying [Figure 3].

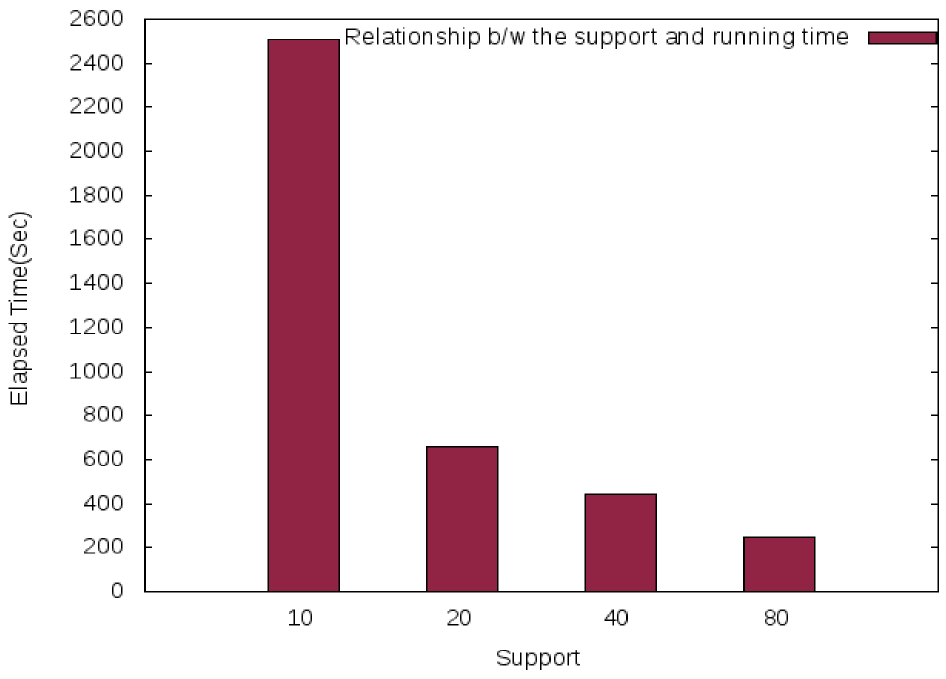

Figure 3. Running time and support relationship

The results obtained in the generation phase are given to the iterative verification phase. Here size i+1 candidate subgraphs are generated from size i subgraphs. Isomorphism checking is done on all candidate subgraphs to check whether similar candidate size i+1 subgraphs are generated from a single size i subgraph. If more than one subgraph is generated, then redundant subgraphs are eliminated taking only one graph into consideration. The information is passed to the reducer of verification phase. The reducer counts the number of occurrences of all the candidate subgraphs and classify them into frequent and infrequent subgraphs based on the given minimum support count. If the count of candidate subgraph is greater than minimum support count, then it is said to be frequent and this subgraph is used to generate next level candidate subgraphs, which occurs in an iterative manner until all the frequent subgraphs are generated. At the end of all the iterations, all the subgraphs and their corresponding support count values are obtained.

6. Conclusions

Map Reduce is one of the most important mostly used standard models which will give us the important models for dealing with the huge amount of information for the clusters.

MRFSE is the major mode of algorithm can be used for generating frequent graphs. It generates and models the filters more graphs which were already available in the database. This model uses two modes. One is used for identifying the frequent edges of the graphs and the second one is used for generating the competitive graphs that can be used for iterative model. The execution time of the model reduces as the number of clusters increases in number.

References

[1] J. Ha., "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, vol.10, pp.13-23, **(2000)**

[3] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," In ICDM, pp.313-320, **(2001)**

[4] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," In ICDM, pp.721-724, **(2002)**

[5] J. Cheng, Y. Ke, and W. Ng, "Efficient query processing on graph databases," ACM Transactions on Database Syst., vol.34, no.1, pp.1-15, **(2009)**

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," Communications in. ACM, vol. 51, pp.107-113, **(2008)**

[7] F. Afrati, D. Fotakis, and J. Ullman, "Enumerating subgraph instances using map-reduce," in Proceedings of. IEEE 29th International Conference on Data Engineering, Apr, pp.62-73, **(2013)**

[8] Agrawal R. and Shafer J.C., "Parallel mining of association rules," IEEE Transactions on KDE, vol.8, no.6, pp.962-969, **(1996)**

This page is empty by intention.