www.ijiarec.com

ISSN:2348-2079

Volume-7 Issue-1

International Journal of Intellectual Advancements and Research in Engineering Computations

An iterative mapreduce based frequent subgraph mining algorithm with load balancing

Mrs K.E.Eswari MCA., M.Phil., ME[1], Mr A.Angappan[2]

[1]Professor, Department of MCA, Nandha Engineering College (Autonomous), Erode-52.

[2]Final MCA, Department of MCA, Nandha Engineering College (Autonomous),Erode-52.

ABSTRACT

In recent years, the "big data" phenomenon has engulfed a significant number of research and application domains including data mining, computational biology, environmental sciences, e-commerce, web mining, and social network analysis. Frequent subgraph mining (FSM) is an important task for exploratory data analysis on graph data especially when the graph is huge. In the recent years, many algorithms have been proposed to solve this task. These algorithms assume that the mining task's data structure is small enough to fit in the main memory in the systems. However, as the real-world graph data grows, both in quantity and size, such an assumption could not be met. To overcome this, some graph database-centric methods have been proposed in recent years for solving FSM; however, a distributed solution using MapReduce paradigm has not been explored extensively. Since MapReduce is becoming the de-facto paradigm for computation on massive data, an efficient FSM algorithm on this paradigm is of huge demand. This paper proposes a frequent subgraph mining algorithm called FSM-H which uses an iterative MapReduce based framework. FSM-H is complete as it returns all the frequent subgraphs for a given user-defined support, and it is efficient as it applies all the optimizations that the latest FSM algorithms adopt. The experiments with real life and large synthetic datasets validate the effectiveness of FSM-H for mining frequent subgraphs from large graph datasets..

Keywords: Big Data, Subgraph Mining, Mapreduce Framework.

INTRODUCTION

Big Data processing mainly depends on parallel programming models like MapReduce, as well as providing a cloud computing platform of Big Data services for the public. MapReduce is a batch-oriented parallel computing model. There is still a certain gap in performance with relational databases. Improving the performance of MapReduce and enhancing the real-time nature of large-scale data processing have received a significant amount of attention with MapReduce parallel programming being applied to many machine learning and data mining algorithms.

Data mining algorithms usually need to scan through the training data for obtaining the statistics to solve or optimize model parameters. It calls for intensive computing to access the large-scale data frequently. To improve the efficiency of algorithms, Chu et al. proposed a general-purpose parallel programming method, which is applicable to a large number of machine learning algorithms based on the simple MapReduce programming model on multi-core processors.

Classical data mining algorithms are realized in the framework, includes locally weighted linear regression, k-Means, logistic regression, naive Bayes, linear support vector machines, the independent variable analysis, Gaussian discriminate analysis, expectation maximization, and back-propagation neural networks. With the analysis of these classical machine learning

Author for correspondence:
Department of MCA, Nandha Engineering College (Autonomous), Erode-52.

algorithms, the computational operations in the algorithm learning process could be transformed into a summation operation on a number of training data sets.

MapReduce is useful in a wide range of applications, including distributed pattern-based searching, distributed sorting, web link-graph reversal, singular value decomposition, web access log stats, inverted index construction, document clustering, machine learning and statistical machine translation. The MapReduce model has been adapted to several computing environments like multi-core, many-core systems, desktop grids, volunteer computing environments, dynamic cloud environments, and mobile environments.

Frequent pattern mining has been a focused theme in data mining for over a decade. Abundant literature was dedicated to this area, making tremendous progress, including frequent itemset mining, sequential pattern mining and so forth. Frequent subgraphs are subgraphs found from a collection of graphs or single large graph with support no less than a user-specified threshold. Frequent subgraphs are useful at characterizing graph datasets, classifying and clustering graphs, and building structural indices. Differentiate the two aforementioned scenarios multi-graph and single-graph, and this paper focuses on frequent subgraph mining (FSM) on the single-graph setting.

Distributed FSM from single massive graphs is challenging, due to not only the special constraints of FSM algorithm design, but also the deficient support from existing distributed programming frameworks. First, an FSM algorithm computes the support of a candidate subgraph over the entire input graph. In a distributed platform, if the input graph is partitioned over various worker nodes, the local support of the subgraph is not much useful for deciding whether subgraph is globally frequent. Also, the support computation cannot be delayed arbitrarily, since candidate frequent subgraphs can be generated only from frequent subgraphs as per Apriori principle.

Additionally, although there are several existing models, including MapReduce , the de facto big data processing framework, they also do not accommodate graph algorithms. Among the distributed graph processing frameworks, Pregel is recognized for its scalability, flexibility, fault tolerance and a number of other attractive features. It is a vertex-centric programming model such that developers usually only need to submit processing scripts on vertices to the framework, which will handle the remaining issues such as graph partition and synchronization. However, it is suggested that structure-related computation may not fit Pregel naturally. Therefore, mapping a structure mining algorithm onto Pregel requires non-trivial efforts, since Pregel does not specify implementation details for self-defined functions [1-5].

RELATED WORKS

G. Liu et al., The authors realized that most of their computations involved applying a map operation to each logical “record” in their input in order to compute a set of intermediate key value pairs, and then applying a reduce operation to all the values that shared the same key, in order to combine the derived data appropriately. The major contributions of this work area simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

J. Dean and S. Ghemawat The goal of topic-based social influence analysis is to capture the following information: nodes’ topic distributions, similarity between nodes, and network structure. In addition, the approach has to be able to scale up to a large scale network. Following this thread, they first propose a Topical Factor Graph (TFG) model to incorporate all the information into a unified probabilistic model. Second, they propose Topical Affinity Propagation (TAP) for model learning. Third, they discuss how to do distribute learning in the Map-Reduce framework. Finally, they illustrate several applications based on the results of social influence analysis.

U. Kang et al., Graphs are ubiquitous: computer networks, social networks, mobile call networks, the World Wide Web, protein regulation networks to name a few. The large volume of available data, the low cost of storage and the stunning success of online social networks and web2.0 applications all lead to graphs of unprecedented size. Typical

graph mining algorithms silently assume that the graph fits in the memory of a typical workstation, or at least on a single disk; the above graphs violate these assumptions, spanning multiple Giga-bytes, and heading to Tera- and Peta-bytes of data.

A promising tool is parallelism, and specifically MAPREDUCE and its open source version, HADOOP. Based on HADOOP, here they describe PEGASUS, a graph mining package for handling graphs with billions of nodes and edges. The PEGASUS code and several dataset are at Unification of seemingly different graph mining tasks, via a generalization of matrix-vector multiplication.

S. Suri and S. Vassilvitskii In this paper, they discover patterns on near-cliques and triangles, on several real world graphs including a Twitter dataset and the "Yahoo Web" dataset, one of the largest publicly available graphs. To enable discoveries, they propose algorithm, an eigen solver for billion-scale, sparse symmetric matrices built on the top of HADOOP, an open-source MAPREDUCE framework. Their contributions are the following:

Effectiveness analyzes billion-scale real-world graphs and report discoveries, including a high triangle vs. degree ratio for adult sites and web pages that participate in billions of triangles. It chooses among several serial algorithms and selectively parallelizes operations for better efficiency. It uses the HADOOP platform for its excellent scalability and implements several optimizations such as cache-based multiplications and skewness exploitation. This results in linear scalability in the number of edges, the same accuracy as standard eigensolvers for small matrices, and more than a 76% performance improvement over a naive implementation.

R. Pagh and C. E. Tsourakakis All instances of a given "sample" graph in a larger "data graph," using a single round of mapreduce. For the simplest sample graph, the triangle, they improve upon the best known such algorithm. They then examine the general case, considering both the communication cost between mappers and reducers and the total computation cost at the reducers. To minimize communication cost, they exploit the techniques of for computing multiway joins (evaluating conjunctive queries) in a single map-reduce round. Several methods are shown for translating sample graphs into a union of conjunctive queries with as few queries as possible [6, 7].

They also address the matter of optimizing computation cost. Many serial algorithms are shown to be "convertible," in the sense that it is possible to partition the data graph, explore each partition in a separate reducer, and have the total work at the reducers be of the same order as the work of the serial algorithm. For data graphs of unrestricted degree, they show that there are convertible algorithms whose running time is of the same order as the lower bounds on number of occurrences of the sample graph that were provided by. They also offer better convertible algorithms when the degree of nodes in a data graph of m nodes is limited to $\sqrt{m}$.

METHODOLOGY

The proposed system includes the before nodes are assigned with graphs for Map process, the graphs are balanced such that all the nodes get correct number of graphs with nodes count. For example, two small graphs are given to Node A and one big graph is given to Node B. So, the map processes are completed in fewer intervals in all the nodes so that reduce phase can be started immediately.

The advantages of the system is, before sending input graph data to nodes, they are balanced. For example, two nodes are equal number of nodes and edges. Nodes complete the Mapper process in less intervals so that Reduce phase can be started with minimum delay. Overall time efficiency is increased.

GRAPH MODEL

Add Node

In this module, the node details such as Node Id, IP Address, X Position and Y Position are registered and the details are saved in 'Nodes' file.

Add Edge

In this module, the source and destination node id details are saved in 'Edges' file. Any number of nodes can be added as friend nodes to given nodes.

View Graph

In this module, the records from 'Nodes' are 'Edges' file are taken and the graph is displayed in Panel control graphically.

FREQUENT MINING PROCESS

In this module, mining frequent sub graph steps are carried out. Let G = {G1, G2, ..., Gn} be a graph database, where each Gi ∈G, ∀ I = {1... n} represents a labeled, undirected, simple (no multiple edges between a pair of vertices), and connected graph. For a graph g, its size is defined as the number of edges it contains. Now, t (g) = Gi : g ⊆ Gi ∈G, ∀ I = {1... n}, is the support-set of the graph g (here the subset symbol denotes a subgraph relation). Thus, t(g) contains all the graphs in 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 $\geq \pi^{min}$, where π^{min} is predefined/user-specified minimum support (minsup) threshold.

The set of frequent patterns are represented by *F*. Based on the size (number of edges) of a frequent pattern, we can partition *F* into a several disjoint sets, *Fi* such that each of the *Fi* contains frequent patterns of size i only.

The mining is carried out using Iterative Map-Reduce approach. FSM-H is designed as an iterative MapReduce process. At the beginning of iteration i, FSM-H has at its disposal all the frequent patterns of size i – 1 (*Fi* - 1), and at the end of iteration i, it returns all the frequent patterns of size i, (*Fi*) . Note that, in this work, the size of a graph is equal to the number of edges it contains. For a mining task if is the set of fre-quent patterns, FSM-H runs for a total of l iterations, where l is equal to the size of the largest graph in *F*. This module implements an FSM algorithm that follows a typical candidate-generation-and-test paradigm with breadth-first candidate enumeration.

The algorithm shows the frequent sub mining with breadth first search enumeration.

```
// 𝒢 is the database
// k is initialize to 1
Mining Frequent Subgraph(𝒢, minsup):
0.  Populate ℱ_1
1.  while ℱ_k ≠ ∅
2.    C_{k+1} = Candidate generation(ℱ_k, 𝒢)
2.    forall c ∈ C_{k+1}
3.      if isomorphism checking(c) = true
4.        support counting(c, 𝒢)
5.        if c.sup ≥ minsup
6.          ℱ_{k+1} = ℱ_{k+1} ⋃ {c}
7.    k = k + 1
8.  return ⋃_{i=1...k-1} ℱ_i
```

GRAPH REPRESENTATIONS

The simplest mechanism whereby a graph structure can be represented is by employing an adjacency matrix or adjacency list. Using an adjacency matrix the rows and columns represent vertexes, and the intersection of row i and column j represents a potential edge connecting the vertexes vi and vj . The value held at intersection < i, j > typically indicates the number of links from vi to vj . However, the use of adjacency matrices, although straightforward, does not lend itself to isomorphism detection, because a graph can be represented in many different ways depending on how the vertexes (and edges) are enumerated.

Subgraph enumeration

It might be classified into two categories: one is the join operation adopted by FSG and AGM and another one is the extension operation . The major concerns for the join operation are that a single join might produce multiple candidates and that a candidate might be redundantly proposed by many join operations. The concern for the

extension operation is to restrict the nodes that a newly introduced edge may attach to. Equivalence class based extension is founded on a DFS-LS representation for trees. Basically, a (k + 1)-subtree is generated by joining two frequent k subtrees. The two k subtrees must be in the same equivalence class.

An equivalence class consists of the class prefix encoding, and a list of members. Each member of the class can be represented as a (l, p) pair, where l is the k-th vertex label and p is the depth-first position of the k-th vertex's parent. It is verified all potential (k + 1)-subtrees with the prefix [C] of size (k − 1) can be generated by joining each pair of members of the same equivalent class [C].Equivalence classes can be based on either prefix or suffix.

Frequency counting

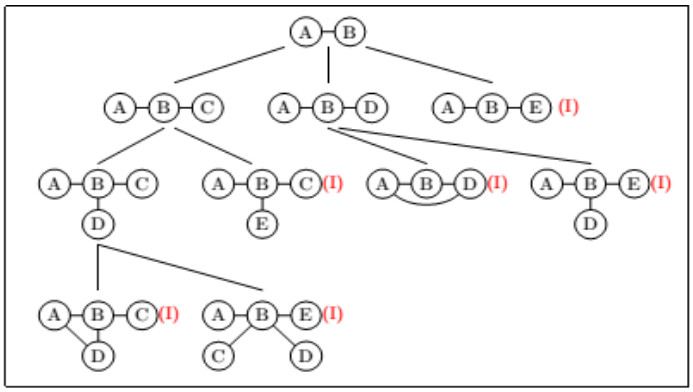
Two Methods are used for graph counting: Embedding lists (EL) and Recomputed embeddings (RE). For graphs with a single node they store an embedding list of all occurrences of its label in the database. For other graphs a list is stored of embedding tuples that consist of (1) an index of an embedding tuple in the embedding list of the predecessor graph and (2) the identifier of a graph in the database and a node in that graph. The frequency of a structure is determined from the number of different graphs in its embedding list. The other approach is based on maintaining a set of active" graphs in which occurrences are repeatedly recomputed.

Frequent sub-graph mining

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 ≥ πmin, where πmin is predefined/user-specified minimum support (minsup) threshold. The set of frequent patterns are represented by F. Based on the size (number of edges) of a frequent pattern, they can partition F into a several disjoint sets, Fi such that each of the Fi contains frequent patterns of size i only.

Mapreduce

MapReduce is a programming model that enables distributed computation over massive data. The model provides two abstract functions: map, and reduce. Map corresponds to the "map" function and reduce corresponds to the "fold" function in functional programming. Based on its role, a worker node in MapReduce is called a mapper or a reducer. A mapper takes a collection if (key, value) pairs and applies the map function on each of the pairs to generate an arbitrary number of (key,value) pairs as intermediate output. The reducer aggregates all the values that have the same key in a sorted list, and applies the reduce function on that list. It also writes the output to the output file Iterative MapReduce: Iterative MapReduce can be defined as a multi staged execution of map and reduce function pair in a cyclic fashion, i.e. the output of the stage i reducers are used as an input of the stage i + 1 mappers. An external condition decides the termination of the job. Pseudo code for iterative MapReduce algorithm is presented in Figure.



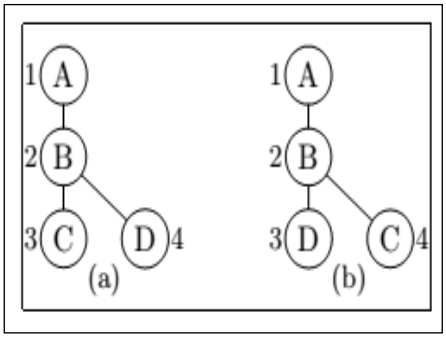
In this paradigm, the mining task starts with frequent patterns of size one (single edge patterns), denoted as F1.Then in each of the iterations of the while loop, the method progressively finds F2, F3 and so on until the entire frequent pattern set (F) is obtained. If Fk is non-empty at the end of an iteration of the above while loop, from each of the frequent patterns in Fk the mining method creates possible candidate frequent patterns of size k+1. These candidate patterns are represented as the set C. For each of the candidate patterns, the mining method computes the pattern's support against the dataset G.

If the support is higher than the minimum support threshold (minsup), the given pattern is frequent, and is stored in the set Fk+1. Before support counting, the method also ensures that different isomorphic forms of a unique candidate patterns are unified and only one such copy is processed by the algorithm. Once all the frequent patterns of size k + 1 are obtained, the while loop continues. Thus each iteration of the while loop obtains the set of frequent patterns of a fixed size, and the process continues until all the frequent patterns are obtained.

Isomorphism checking

Given a frequent pattern of size k, this step adjoins a frequent edge (which belongs to F1) with c to obtain a candidate pattern d of size k + 1. If d contains an additional vertex then the added edge is called a forward edge, otherwise it is called a back edge; the latter simply connects two of the existing vertices of c. Additional vertex of a forward edge is given an integer id, which is the largest integer id following the ids of the existing vertices of c; thus the vertex-id stands for the order in which the forward edges are adjoined while building a candidate pattern. In graph mining terminology, c is called the parent of d, and d is a child of c, and based on this parent-child relationship they can arrange the set of candidate patterns of a mining task in a candidate generation tree.

FSM algorithms also impose restriction on the extension nodes of the parent pattern so that redundant generation paths can be reduced. One such restriction that is used in the popular gSpan algorithm is called rightmost path generation that allows adjoining edges only with vertices on the rightmost path. Simple put, "right most vertex" (RMV) is the vertex with the largest id in a candidate subgraph and "right most path" (RMP) is the shortest path from the lowest id vertex to the RMV strictly following forward edges.



Pattern	Occurence List (OL)
A–B	1 : [(1, 2)] ; 2 : [(1, 2)]
B–D	1 : [(2, 4)] ; 2 : [(2, 3)] ; 3 : [(1, 2)]
B–E	2 : [(2, 5)] ; 3 : [(1, 3)]
A–B–D	1 : [(1, 2), (2, 4)] ; 2 : [(1, 2), (2, 3)]
A–B–E	2 : [(1, 2), (2, 5)]

CONCLUSION

In In this paper MapReduce framework has been used to mine frequent patterns where the transactions in the input database are simpler combinatorial objects frequent subgraph mining on MapReduce, however, their approach is inefficient due to various hortcomings. The most notable is that in their method they do not adopt any mechanism to avoid generating duplicate patterns. This cause an exponential increase in the size of the candidate subgraph space; furthermore, the output set contains duplicate copy of the same graph patterns that are hard to unify as the user has to provide a subgraph isomorphism routine for this amendment.

It is believed that almost all the system objectives that have been planned at the commencement of the software development have been net with and the implementation process of the thesis is completed. A trial run of the system has been made and is giving good results the procedures for processing is simple and regular order.

By analyzing the sequential version of parallelization of distribution load balacing algorithm is researched and implemented based on parallel programming model. Experimental results show that the proposed parallel algorithm for frequent sub graph is feasible and improvement in the allocation speed is extremely obvious. The application eliminates the difficulties in the existing system. It is developed in a user-friendly manner. The application is very fast and any transaction can be viewed or retaken at any level. Error messages are given at each level of input of individual stages.

REFERENCES

[1] G. Liu,M. Zhang, and F. Yan, “Large-scale social network analysis based on Mapreduce,” in Proc. Int. Conf. Comput. Aspects Soc. Netw., 2010, 487–490.
[2] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” Commun. ACM, 51, 2008, 107–113, 2008.
[3] U. Kang, C. E. Tsourakakis, and C. Faloutsos, “Pegasus: A petascale graph mining system implementation and observations,” in Proc. 9th IEEE Int. Conf. Data Mining, 2009, 229–238.
[4] U. Kang, B. Meeder, and C. Faloutsos, “Spectral analysis for billion-scale graphs: Discoveries and implementation,” in Proc.15th Pacific-Asia Conf. Adv. Knowl. Discov. Data Mining, 2011, 13–25.
[5] S. Suri and S. Vassilvitskii, “Counting triangles and the curse of the last reducer,” in Proc. 20th Int. Conf. World Wide Web, 2011, 607–614.
[6] R. Pagh and C. E. Tsourakakis, “Colorful triangle counting and a mapreduce implementation,” Inf. Process. Lett., 112(7), 2012, 277–281.
[7] F. Afrati, D. Fotakis, and J. Ullman, “Enumerating subgraph instances using map-reduce,” in Proc. IEEE 29th Int. Conf. Data Eng., 2013, 62–73.