Map Reduce Based Optimized Frequent Subgraph Mining Algorithm for Large Graph Database

Sadhana Priyadarshini, Sireesha Rodda

Abstract: Distributed System, plays a vital role in Frequent Subgraph Mining (FSM) to extract frequent subgraph from Large Graph database. It help to reduce in memory requirements, computational costs as well as increase in data security by distributing resources across distributed sites, which may be homogeneous or heterogeneous. In this paper, we focus on the problem related complexity of data arises in centralized system by using MapReduce framework. We proposed a MapReduced based Optimized Frequent Subgrph Mining (MOFSM) algorithm in MapReduced framework for large graph database. We also compare our algorithm with existing methods using four real-world standard datasets to verify that better solution with respect to performance and scalability of algorithm. These algorithms are used to extract subgraphs in distributed system which is important in real-world applications, such as computer vision, social network analysis, bio-informatics, financial and transportation network.

Keywords: Distributed System, subgraph, support count, Graph Database, Mapper, and Reducer.

I. INTRODUCTION

Recently, the algorithm used to enhance the performance of graph data mining are classified into two groups. First, Graph Mining emphasis on searching those pattern are most frequent subgraphs in that graph. Second, Graph Partition that based on classification of a big graph database into smaller so that we can easily manage consecutively.

Due to the advent of new technologies, devices, and communication means like social networking sites, the amount of data produced by mankind is growing rapidly every year. As data size increasing very fast, the main challenges are to deal with graphs of big sizes that grow in terabytes or petabytes scale. To overcome these problem, we use graph division that reduce the complexity of graph mining algorithm, which helps to secure the most sensitive data, less cost used in memory, computation as well as in transmission during distributed system.

Distributed Graph Mining broadly classified into Agent based, or Client-Server model. Single-agent and Multi-agent are two sub groups of Agent-based model. Client-Server model is further classified into Classifier Based and Privacy Preserving model. We can use either Homogeneous or Heterogeneous Technique to calculate patterns between distributed data depending on datasets. The Privacy Preserving model used to protect our data from unauthorized exposure. It can be used either Cryptographic or Randomization techniques. The Cryptographic technique have better result than other one on the basis of accuracy and privacy [6].

In *Distributed Graph Mining*, researcher have to concentrate how to distribute the entire graph database and computation of algorithm over the entire network of similar or diverse types. Graph Database are easy to store in heterogeneous than homogeneous. Not only the cost associated with transmission, computation and memory can be less by distributing data mining, but also we can able to provide more data privacy on our database. In this paper, we mostly emphasis on extraction of data in form of subgraphs presented in heterogeneous sites .We follow *Decentralized Graph Mining* technique to make entire system can be distributed workload properly heterogeneous sites. Arabesque, a system for distributed graph mining, follows *"think like a vertex" (TLV)* programming paradigm [16], which provide a high-level filter-process computational framework consist of frequent subgraph mining, counting motifs, and finding cliques. Arabesque provide both graph computation and graph mining algorithm to run on the top of same infrastructure (i.e. Apache Graph [15]) by using Bulk Synchronous Processing (BSP) model.

In recent years, MapReduce becomes main model for computation on big data. It supports centralized data of distributed computing system. During big data analysis, it utilise the "*Distributed File System*" to improve input/output operations. The framework provide higher level of data abstraction and keep hides system level details from programmer, so that they can able more concentrate on problem oriented computation logic .Recently scientist are more emphasis on analyse and design of large network graph database to overcome major challenges arise in Big data like capturing data, storage, searching, sharing, transfer, analysis, presentation, etc. However, in various discipline's like, computational mining, computer biology, link spam detection, reachability and distance query indexing, use MapReduce[7] framework for generate densest subgraphs which more result able and sufficient compare to heuristic approach.

*Paper organization***.** The remainder of the paper is consist of following section. Section 2 establishment of previous work on distributed system. Section 3 provides fundamental concept and definitions,

Revised Manuscript Received on February 24, 2020.

Ms. Sadhana Priyadarshini, Ph.D. Scholar, Department of Computer Science and Engineering, GITAM (Deemed to be University), Vishakhapatnam, India

Dr. Sireesha Rodda, is a Professor in the Department of Computer Science & Engineering, GITAM (Deemed to be University) Vishakhapatnam, India.

whereas Section 4 describes overview as well as detail of our proposed Model. All the experimental analysis represent in Section 5 followed with conclusion in Section 6.

II. RELATED WORK

Frequent Subgraph Mining is recently studied research field in Distributed System. Broadly existing algorithm based on either Apriori-based or Pattern-growth methods which based on Breath-First Search (BFS) or Depth-First Search (DFS) respectively. Gonzalez, Low, Gu, Bickson, and Guestrin. discovered frequent subgraph mining by using Minimum Description Length (MDL), which can be implemented on both supervised concept learning and unsupervised pattern discovery and illustrate its scalability and effectiveness[6].

To make mining process faster Kuramochi and Karypis used heuristic algorithm GREW in canonical graphs. He proposed a mechanise for cutting in frequent subgraph, which focus on bigger subgraph candidate based on smaller subgraph and satisfy all criteria of frequent subgraph. Then he implemented with both BFS and DFS approaches and developed new algorithm HSIGRAM and VSIGRAM that used to find out frequent subgraph candidate [9].

Khan and Yan et [1] derived a proximity pattern concept that is counted by *pFP(probabilistic algorithm FP_growth)*, where the complexity of algorithm is lower than isomorphism graph technique. This algorithm transfer a complex graph mining problem to a simplified probabilistic item set miming pattern. They introduced an objective function to measure the *interestingness* of a proximity pattern and make the algorithm more efficient and effect to generate only the *top-k interesting pattern.*

Sun, Wang, Shao, and Li [19] ,developed the algorithm that make graph matching faster by changing function of index to become join and expand sungraph which lead to save memory requirement as well as index managing during frequent subgraph mining. They elucidated the algorithm, that support efficient subgraph matching for graph placed on a distributed memory stored. The algorithm use efficient graph exploration and massive parallel computing for query processing. This technique validity of performing subgraphs matching on web_scale graph data.

Zhao et al [20], derived SAHAD, an algorithm to overcome problem associated with extraction labelled subgraphs in network, which are isomorphic to template. The subgraphs are represented in form of tree by using Hadoop. The technique is able to find out motifs and computing graph lets frequency distribution. It can be able to run easily on Amazon EC2, without needs for system level optimization.

Han and Wen [8] proposed a model that based on the VIL (Vertex Identification List) in enumeration phase of solution candidate can be defined as pattern to be search.They proposed a new class of pattern names as frequent neighbourhood pattern, where a neighbour is a specific topological pattern in which a vertex (node) is embedded, and pattern is frequent if it is stored by a large position of nodes. The targets are clear semantics and are not limited to tree like shapes. The technique is feasible and unique ability to provide user with especially interesting pattern.

In 2009, Kang, Meeder, Papalexakis, and Faloutsos [17] developed PEGASUS algorithm by using MapReduce model to analysis big graph, they used *Page Rank* for combing the distance between modes and diameter of graph. *Bulk Synchronous Processing (BSP)* platform used to find the shortest path verification of bipartite Semi Clustering graph in 2010.

Teixeira et al [15] developed Arabesque system that generates process of extracting a very large number of subgraphs by using *a high-level filter-process* computational model. It solve the problem associated with frequent subgraph mining, counting motifs, and finding cliques. This system concentrates on scalability and provide customer-friendly simple programming API that allows non-experts to build workload.

Aparicio, D.,Ribeiro, and Silva, proposed a graph pattern mining engine for distributed graph processing system that is both fast and scalable to large graphs. They used *neighbourhood sampling* technique, which increases the probability that an estimator would actually find an instance of given pattern, hence need less estimator to get same accuracy. This algorithm also overcomes following challenges, general pattern, distributed setting, error-latency profile, and handling updates [12].

Mccune, Weninger, and Madey [11], tried to sort out the issues related to billion-node graph that exceed the memory capacity of standard machines are not well-supported by popular Big data tools. The new vertex-centric programming framework challenges one to "*think like a vertex "(TLAV)* and executes the customer-defined programs from the perspective of a vertex rather than graph. The timing, communication, execution model, and partitions are main milestone related to distributed algorithm.

The parallel methods, which are used to handle large graph fully rely on the expensive join operation which reduces the performance. Shao, Cui, Chen, Yao, and Yingxia [14], designed a parallel subgraph listing frame name *PSgl*, that repetitively eliminates the subgraph instances and *use divide-and-conquer* method to solve the subgraph listing. It purely based on graph traversal, and avoids the explicit join operations. The performance, scalability, and fault-tolerance of Pregel are already satisfactory for graph with billions of vertices.

Giuseppe et al[5], described three vital aspects of the proposed distributed framework, namely a distribution process based on a peer-to-peer communication, and dynamic partitioning of search space, and a novel receiver-initiated load balancing algorithm. It tolerates node failure and communication latency and supports dynamic resource aggregation. Its dynamic resource aggregation make feasible for large-scale, multidomain, heterogeneous environments.

Aparicio, Ribeiro, and Silva [12], derived a novel parallel method for subgraph counting geared towards multicores. They used *state-of-art g-tries* data structure, which is core of fastest sequential algorithm for subgraph counting. The g-tries are multiway trees, much like prefix tree that use common topologies in subgraphs in order to prune the search tree. They developed an efficient sharing mechanism that is able to stop, split and resume the execution of dynamically divide the search tree among threads.

The g-tries, a data structure specifically designed for discovering subgraphs frequencies.

Ribiro and Silva build a tree structure to summarize the structure of entire graph set, by combining the common prefix topologies in the same way a prefix tree. A g-tries is a multiway tree that can store a collection of graphs. They used sampling methodology capable trading accuracy for even faster execution times, additionally improving the potential of the developed data-structure [13].

Anchuri, Zaki, Barkol, Golan and Shamy [3], derived an algorithm to extract frequent approximate pattern in a single large graph database in the presence of a cost matrix between labels. They proposed label based repetitive pruning method to compute the representative set efficiently .It relied on k-top label and neighbor concatenated labels, which handle both arbitrary as well as binary cost matrices.

III. PRILIMINARIES

In this section, we explore some of definition and study related to our work. Table-I outlines the basic notations used throughout our paper. Section A illustrate topics related to *Frequent subgraph Mining*, whereas Section B emphasis on overview of MapReduce Framework.

A. Frequent Subgraph Mining (FSM)

Let, GD={GD_1, GD_2, GD_3…. GD_n }, $\forall_{i=1,2,3\ldots n}$ be a graph where n is number of graphs present in GD. We define each $GD_i \in GD$ as a quadruple $GD=(V_{gd},E_{gd},L_{gd},l_{gd})$,where $V_{gd},E_{gd},L_{gd},l_{gd}$ are set of vertices , edges, labels, and labelling function that map every vertices and edges to a single Label in L_{gd} respectively. For our proposed work, we consider undirected and connected graph only.

Table-I: List of Notations

Symbol	Description
GD	Graph Database(collection of graphs)
V_{gd}	Set of all vertices in GD
E_{gd}	Set of all edges in GD
n=|GD|	Number of graphs in GD
G_i	Number of subgraphs in GD_i
FE	Set of frequent Edges
CS	Candidate Set
τ	User given minimum threshold value
R	Result Set
S	Subgraph
N	Number of MapReduce machines(worker nodes)

Definition1:(Subgraph) Given a graph $S=(V`_{gd},E`_{gd},L`_{gd},l`_{gd})$ is said to be *subgraph* of another graph $GD=(V_{gd},E_{gd},L_{gd}, l_{gd})$, if there exit an injective function $\psi : V`_{gd} \rightarrow V_{gd}$ such that $\forall$ $(a,b) \in E_{gd}$ it must hold that $(l_{gd}(a)= l`_{gd} (\psi(a))) \wedge l`_{gd}(b)= l`_{gd}(\psi(a)) \wedge l_{gd}(a,b)= l_{gd}(\psi(a), \psi(b))$.

Definition2:(Subgraph Isomorphism)A subgraph isomorph-hism from subgraph $S=(V`_{gd},E`_{gd},L`_{gd},l`_{gd})$) to graph $GD=(V_{gd},E_{gd},L_{gd},l_{gd})$, is denoted by $G \preceq S$, which is an bijective function $\phi : V_{gd} \rightarrow V'_{gd}$ such that for every pair of vertices v_i, v_j if $(v_i , v_j) \in E_{gd}$ then $\phi(v_i), \phi(v_j) \in E'_{gd}$.

Definition 3:(frequency of subgraph)The frequency of subgraph S is calculated by number of times it present in GD. $f(S)=\{S|S \in GD \wedge S \subset GD\}$.

B. MapReduce Framework

MapReduce is a processing technique and a program model for distributed computing based on java. It consist of two tasks, Map and *Reduce.* The Fig.1 shows that, the Map takes input as a set of data and transform into a specific form where each element split into tuples (key/value pairs). Secondly Reduce, that takes the output from Map as input data and merge those data into a similar set of tuples. MapReduce programs implements in tree steps, namely map, shuffle and reduce [7].

- **Map:** The map or mapper task to organize input data which is in form of file dictionaries stored in *Distributed File System (DFS) or Hadoop Distributed File System (HDFS)*
  The input file is processed in mapper function line by line and generate several small chunks of data (i.e. key/value pair).
- **Reduce:** This step is consist of *Shuffle* and *Reduce*. The shuffle accumulate key-pair into a group of list of same key and sort it. Then individual key associated list of value sent to different machines (systems).The result of shuffle and sort sent to reducer, where reducer use *"reduce"* function each list of value and generate a unique key,final output <key, value> will be stored or display. The table-II illustrates how the Input and Output types of a MapReduce job are going to done (i.e. (Input) $<k_1, v_1> \rightarrow$ map $\rightarrow <k_2, v_2> \rightarrow$ reduce $\rightarrow <k_3, v_3>$ (Output)).

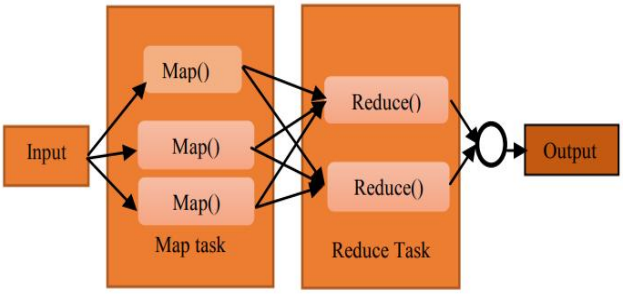


Fig. 1. Flow chart of MapReduce Framework

Table-II: Mapper and Reducer job

	Input	Output
Map	<k1,v1>	List<k2,v2
Reduce	<k2,list(v2)	List<k3,v3>

IV. OVERVIEW OF PROPOSED MODEL

In case of parallel computation of *Graph Mining*, we get faster result and better performance than sequential approach. However, in case of the large-scale graph processing, it is difficult to handle in parallel due to shortage of memory requirement, complexity of graph computational and privacy of data. In this paper, we proposed a model *MapReduced based Optimized Frequent Subgrph Mining (MOFSM)* shown in Fig.2, that used repetitive MapReduce framework with Optimized Frequent Subgraph Mining dynamically.

Homogeneous Classifier-based method, that based on *Ensemble Learning, Distributed association Rule Mining(DARM),MetaLearning, Knowledge Probing is* implemented for evaluation of pattern in distributed system. It is purely based on Client-Server model, where graphs are accumulated and processed at global level. Hadoop MapReduce is the processing component of Apache Hadoop that process the data parallel in distributed environment. MapReduce[7] can be implemented in Google, Apache Hadoop. In our proposed work, we use *Hadoop Distributed File System (HDFS)* for storing graph database.

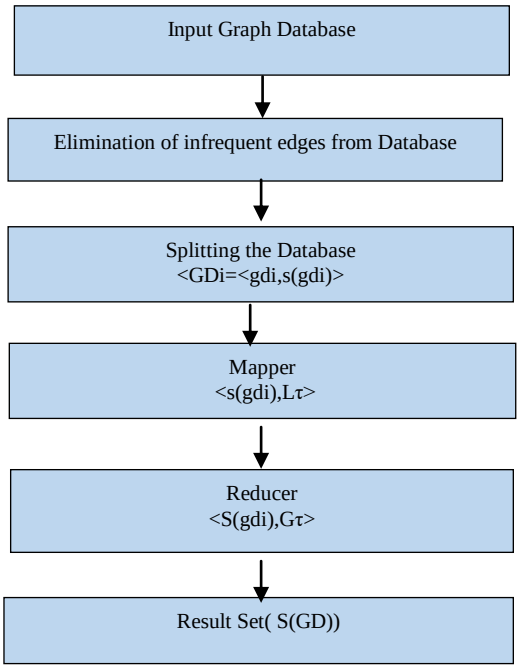


Fig 2. Framework model for MapReduced based Optimized Frequent Subgrph Mining (MOFSM).

Further, the pattern design consist of *Summarization* (eg. Inverted Index), *Recommendation* (eg. Sorting), *Classification* (eg. Top N Record) *and Analysis* (eg. Join, Selection). We use the analysis design pattern for our database. Here main precedence that, we can process data in parallel in distributed sites that leads to less computational time. Instead of moving entire graph database, we only shift processing logic to different site where actual data present.

In MOFSM technology, we generate frequent subgraph pattern from Graph Database if its support values equals to or greater than minimum threshold value (τ). Further, in case of dynamically division, the entire dataset is distributed over different sites in form of worker nodes. Therefore, it is difficult to estimate subgraph support count value in MapReduce model, as local support in respective division at worker nodes, are difficult form the support in global state communication. As Apriori-based algorithm, we can't postpone the support calculation. To overcome this problem, we introduced a new value to each node before partition which further used in worker nodes site for calculation of support count. At destination site, first we

Algorithm 1: Filter (GD, τ)

1. $F_e \leftarrow \varphi$
2. for each $e \in F_e$ do
3. Calculate f(e)
4. If(f(e))$\geq \tau$
5. then $F_e \leftarrow F_e \cup e$
6. return F_e

Algorithm 2: Geometric Two-Way Graph Division

Input: GD = (X, x, y, z)
Output: GD_1 and GD_2

1. Let xyzw = π (xyz)
2. Select the center point C_p of xyzw.
3. Perform conformal mapping to choose the largest circle LC_p on the unit sphere in R^{n+1}
4. Transform the LC_p into a sphere S in R^n using reverse conformal mapping.
5. Divide the sphere S vertices into two parts xyz_1, xyz_2.
6. Generate two graphs GD_1 and GD_2 are constructed from $xyzw_I$, $xyzw_2$ respectively.

check each node frequency value, if its value more that *threshold value (τ)*, then consider else discard it [20].

Our proposed work is based on four phases: *Splitting, Mapping, Shuffling, and Reducing*. We consider our database in form of a single large graph or set of small graphs. During Splitting phase, we use GMOFSM to split entire dataset into number of graphs GD_i, i=1,2,3..n,where n is number of workers nodes. Mapper used in mapping phase to extract subgraph with local frequency associated with each graph-id. During Shuffling phase ,each subgraphs is associated with its global frequency which become input to Reducer and generate all the frequent subgraphs whose frequency value equal to or greater than threshold value. Fig: shows the framework of different phases for generation of frequent subgraphs.

V. SPLITTING

In Splitting phase of our proposed work, the entire graphs (GD) divides into many partitions (GD_i).In algorithm 1,first we assign set of all frequent edges(F_e) as null in line-1.At each iteration we take individual edges, and computed its frequency f(e) inside the graph, if its value same or more than user given threshold value, then the edge added to F_e ,else discard(line2-5).During splitting section, each graph is associated with <key, value> pair, $GD_i, \forall_{i=1,2,3\ldots n,}$ where n is number of graphs present in GD_i. Now (key, value) pair generated by this phase as <graph-id> as key and <gd_i, F_e > as value for input to mapper [6].In our proposed framework, we derived Geometric Multi-way Division algorithm (Algorithm 2), which divide the entire graph database into n-disjoint graphs. Let a Graph Database (GD) in R^n of V vertices and E edges, we implement two-way division algorithm to get n-disjoint graph, where n is any positive integer. The X is an array of vertices pair that represents the edges among lattice vertices in G, such that, X = {{(x_1, y_1, z_1), (x_2, y_2, z_2)},…{(x_v−1, y_v−1, z_v−1), (x_v, y_v, z_v)}},where v is the number of vertices in R^n [13].In algorithm 2 (line 2) ,the standardized formulae of graph in eq^n(1) is used to calculate the center point of given points in sphere, where (x',y',z') and r are the center point and radius of sphere respectively.

$$(x-x')^2+(y-y')^2+(z-z')^2=r^2 \quad (1)$$

In case of conformal mapping, first, we use translation in the space is described by t_x,t_y,t_z. It is easy to this matrix realizes the equations:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$x=x'+t_x$

$y=y'+t_y$

$z=z'+t_z$ (2)

Second, we perform in $\mathbb{R}^3$, coordinate system rotations of the *x*-, *y*-, and *z*-axes in a counter clockwise direction when looking towards the origin give the matrices.

$$R_x(\alpha)=\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} \quad (3)$$

$$R_y(\beta)=\begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \quad (4)$$

$$R_z(\gamma)=\begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

Third, Scaling is describe by S_x,S_y,S_z. We see that this matrix realizes the following equations:

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & t_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$x_2=x_1 \cdot S_x$

$y_2=y_1 \cdot S_y$

$z_2=z_1 \cdot S_z$ (6)

To achieve better performance, the number of division also plays vital role .The different way of splitting show faster result, which we shown in experimental section. Before Spliting, we used a *"filtration"* technique to filter out infrequent edges from graphs which reduce further computation, transmission, i/o cost on worker nodes. In this paper, the input Graph Database (GD) split into many partition and the OFSM method iteratively use MapReduce.

mostly more than MapReduce jobs, therefore it optimized the desired result. By adding load balance technique in MapReduce task, we also achieve better result. Before Graph partition, input graph dataset go to pre-filling stage, where we remove all infrequent edges [14]. During scanning of graph database for entire GD, OFSM maintain a support-list from each edge to find out infrequent edges against the user-defined minimum threshold value.

Algorithm 3: Frequent Subgraph Generation

Input: A graph g, user given support threshold τ value
Output: total frequent subgraphs set R
1. R← φ
2. Let FE be the set of all frequent subgraphs with all frequent edges of graph GD.
3. for each f ϵ FE do
4. R ← R U Subgraphextension(s,GD,τ,R, FE)
5. Delete f from FE and g
6. return R

Algorithm 4: Subgraphextension

Input: subgraph s, a graph g, min-sup τ, a set of frequent edges FE, result set R
Output: all frequent subgraphs of g that extend s

1. Result(R)← φ
2. Candidate set(CS)← φ
3. for each edge e in FE do
4. Let extension in S` be the extended of e by adding frequent edge e'
5. If S` ⊄ CS, then CS←CS U S`
6. for each x ϵ CS do
7. if isomorphic check(x) =true and x contains repeated e ,then
8. Calculate Upper_Bound of x, UB(x).
9. If UB(x) ≥ τ and x.sup ≥ τ do
10. R ← R U {x}
11. else
12. discard x from CS
13. return R

VI. MAPPING

The map function used to generate a list of values of (key-value) pair from splitting phase. The algorithm for finding FSM can be sequential or non-sequential on a single large graph database or set of small graphs. Basically both candidate generation and support calculation methods are needed to pick up required subgraphs. We start with single-edge pattern(p_i).At each iteration of while loop in algorithm OFSM ,the k+1 subgraphs is generated by mapping two k-disjoint subgraphs by adding either forward edge or back edge. There is chances of duplicates subgraphs generation which we overcome by isomorphism checking. We use *min_dfs_code* for it. There are more than one generation path for each candidate pattern, we extract only valid candidate path. The technique, we use as follows: A valid generation path whose insertion order of edges matches with the edge ordering in min_dfs_code. In algorithm 4, we performed first isomorphic test, then *Upper-Bound* for elimination method to find out support value, which filter out some of infrequent edge before support count [12].The biggest feasibility support value is called as its UpperBound, UB(x).The computation cost is reduced by discarding infrequent edges which further reduce by calculating Upper Bound of edges. The {key, value} pair used to pass in distributed phase to different sites.The key is consist of graph identification number and value that is associated with subgraph, edge extension embedding which applied on map function. The *Candidate Generation* is implemented by combining two (n)-size subgraph to produce (n+1)-size subgraph. We use *core identification, join and down-word closure property* of support condition to remove the repeated one.

In this phase, it reads all subgraphs associated with graph-id. All the technologies used *branch_and_bound* to find out locally frequency of subgraph. Then we add a *"dummy root"* node over the single edge subgraphs that correspond to empty subgraphs.

Each node extends the graph of its present nose(s) by adding single edge, a child node subgraph of its parent. We use top-down approach to traverse the tree and calculate each node support value. If a visited node is frequent correspond to graph and frequency are output and the process continues with child node. Otherwise, we pruned all its child node and their descendants.

Algortihm 5:Mapper

1.$S(gd_i) \leftarrow \varphi$
2.for each $gd_i \in GDi$ do
3. $S(gd_i) \leftarrow$ Frequent Subgraph Generation(gd_i, τ) U $S(gd_i)$

VII. SHUFFLING

During *Shuffling*, we reduce the unnecessary I/O and communication overhead by decomposing the output in the map function into following ways: for each subgraphs present in gd_i, we shuffle in such a way that each subgraph is associated with all support value in entire graph database (GD_i) and make list of subgraphs with all support value in different worker node.

VIII. REDUCING

In this phase, we calculate global frequency of each subgraph by summarize all local frequency present in individual worker node. The output of shuffling and sorting of key-value pair are input to reducer, then we calculate the global support(gl(f))value of each candidate subgraph pattern by aggregating the support count generated in graph partition, If its value more than or equal to given threshold value (τ),then reducer result appropriate key-value pair in HDFS else discard it.

Further, a graph is consider as candidate if it is *locally frequent* of any worker node. To avoid false candidate

we check the condition; if any graph in any worker node is not reported as candidate by any machine, then frequency of GD in G_i must be satisfy the following inequality:

$$Sc(GD) = \sum_{k=0}^{n} Sc(Gi) < \sum_{i}^{n} \tau.\, ni = \tau.n_i \qquad (7)$$

Algorithm 6: Reducer

1. g(s))← φ
2. for each s ϵ $s(gd_i)$
3. calculate gl(f)
4. if gl(f) ≥ τ then
5. g(s) ← g(s) U s
6. else discard it.

IX. EXPERIMENTAL RESULT AND DISCUSSION

In this section, we represent the experimental, result that show the performance of MOFSM for resolving the extraction of frequent subgraphs on a large graph datasets. All the experimental we conducted on Intel (R) CPU 3.10 GHz PC with 4 GB RAM running on 32-bit windows operating systems. We use following four real-world graph datasets.

Patent citation network managed by the *National Bureau of Economic Research*. The data collected on period from 1st Jan 1963 to 30th Dec. 1999 including 3,923,922 patents .The citation graph include all citation made by patents granted during that period, totalling 16,522,438 citations.

Twitter tweets dataset collected 467 million twitter post from 20 million during 1st Jan 2009 to 31st Dec 2009. We use only calculate 20-30% of public tweets published on Tweeter during that specific time span. Each public tweet consist of Author, Time, and Content information. We consider 17,069,982 user associated with 476,553,560 tweets.

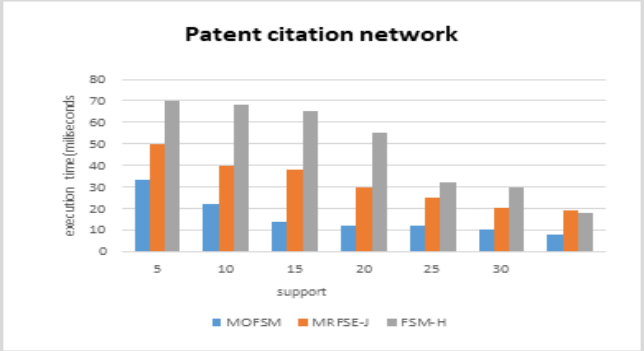
Amazon Website dataset based on "Customer who bought this item also brought feature" of amazon website. We consider only largest connected component which consist of 334863,925872 numbers of nodes and edges respectively.

Google Web graph dataset use nodes and edges to represented webpages and hyperlinks respectively. The data was released in 2002 by Google as a part of Google Programming Contest which consist of 875713 web pages and 5105039 hyperlinks.

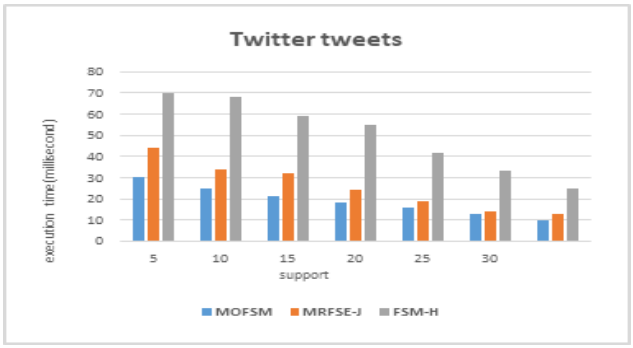
Table-III: Statistics of MapReduced based Optimized Frequent Subgraph Mining with min_sup value 8.

Data sets	No of Node	No. of Edge	Run Time (min)	Mapping Time (min)	Reducing Time (min)	Comm. Over n/w (GB)
Patent citation network	3,923,922	16,522,438	40.8	12.5	49.3	57
Twitter tweets	17,069,982	476,553,560	69.75	17.9	56.7	81
Amazon Website	334863	925872	22.75	5.69	21.07	3.2
Google Web	875713	5105039	30.5	7.28	21.84	6.9

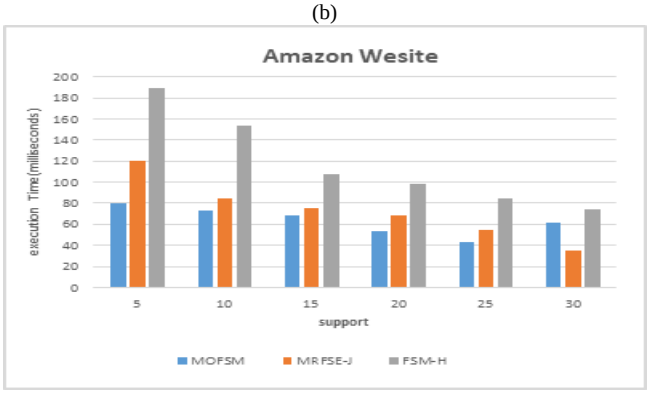
In this section, we compare efficiency and effectiveness of our proposed methods with existing one. In Fig.3, we vary minimum support threshold (τ) from 5 to 30 with fix number of data node to 8.The fig. illustrate that as number of τ increases the execution time decrease. Our result approximate nearly 5%, 15% improve than MRFSE-J and FSM-H respectively.



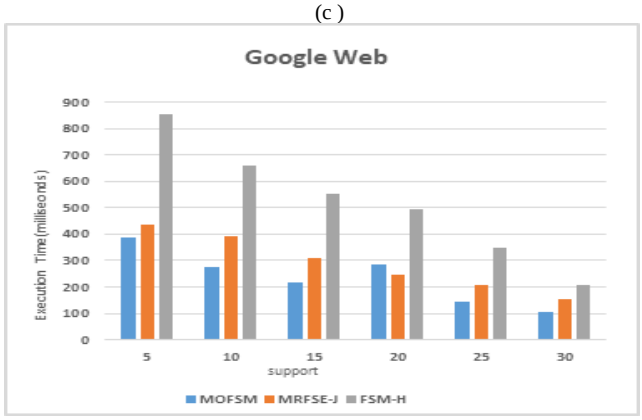
(a)



(b)



(c)

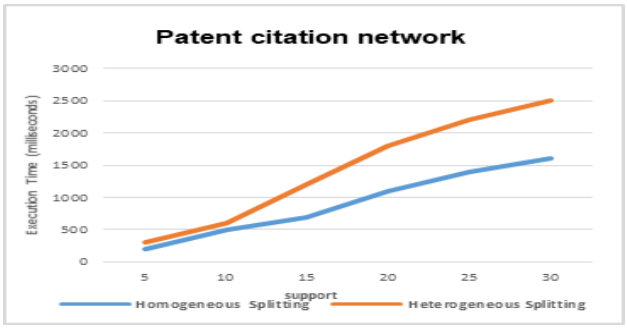


(d)

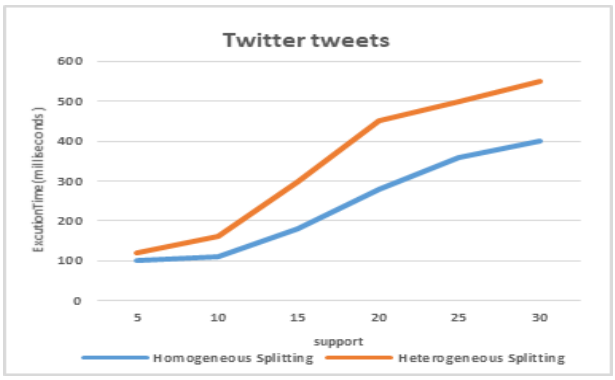
Fig.3: Relationship between different minimum support values with execution time four different datasets.

We illustrate the performance of our proposed model in different categories of partition in Fig.4 by using heterogeneous and homogeneous strategies of graph splitting, we concluded that performance of randomly division method is better than equal size division in all four datasets. The Table-III represents how the different datasets affects the data communication over the network in Gigabytes. The communication size is directly proportional to number of nodes in each database.

During execution of MapReduce job in Hadoop, the number of reducer we use played vital role. When devaluing data (output) in HDFS, a MapReduce job go along with a convention of naming the output file with the key record "part". Reducer calculates the number of "part" will be produced to hold the result of job. If we use only one reducer, then entire result stored in a single file. As we use repetitive use of MOFSM, where result of current task is set as input to next task, the number of reducer plays outstanding effects on the runtime of MOFSM.



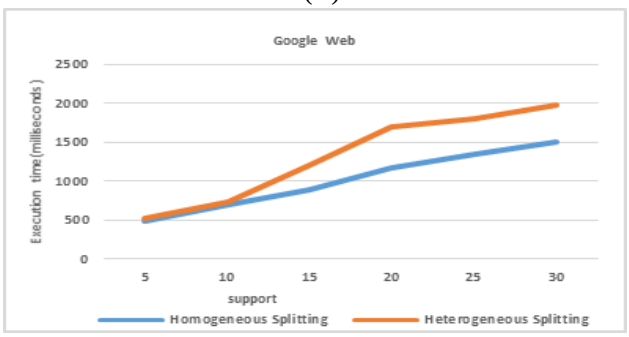
(a)



(b)

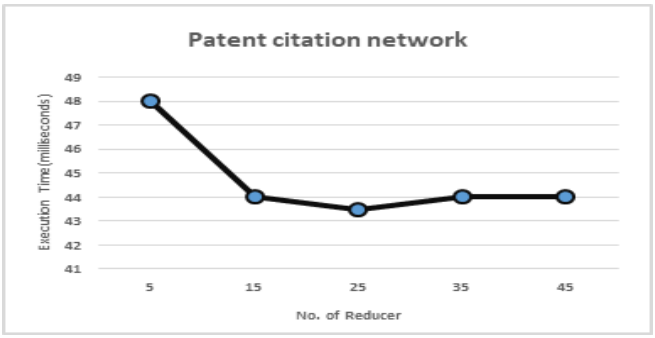
If we use less number of reducers, then result of output files stored in larger size which will be a burden over the network during transmission of data nodes. On other hand, if we use large number of reducers, then there is chances of creation of output files of zero size. Further, this zero size output file creates overhead for next stage of mappers.

(c)

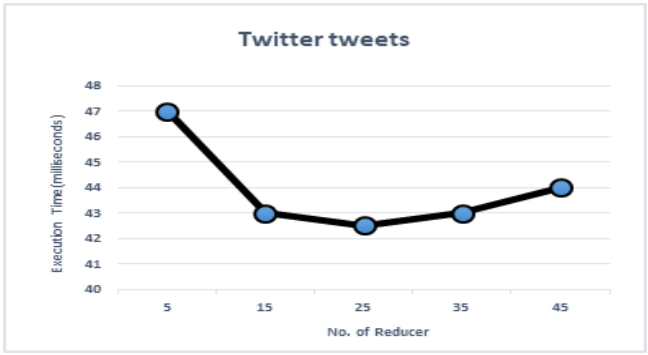


(d)

Fig.4 Relationship between different Splitting techniques with its execution time four different datasets.

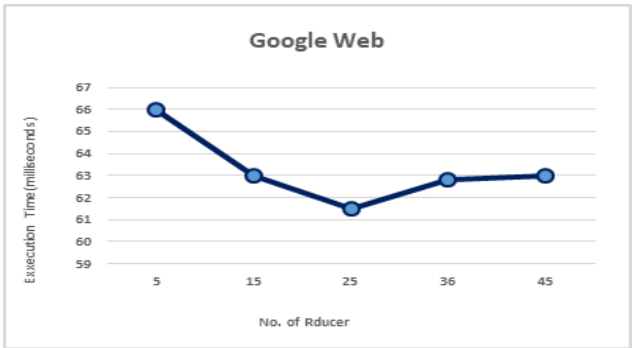


(a)



(b)

(c)



(d)

Fig.5: Relationship between numbers of Reducer used with execution time four different datasets.

Hence loading an input file is costly. In fig.5, we take different number of mappers (i.e.5, 15, 25, 35, 45) and calculate runtime for MOFSM .As per the fig, we conclude that 25 is the best choice for number of reducer in our proposed model.

X. CONCLUSION

In this paper, we derive how to perform FSM in a distributed system. We defined a MapReduce model that we use OFSM for extraction of frequent subgraphs .We also analyse how to perform extraction on different type of networks system for a large scale graph database. For experiment analysis, we use all datasets with different minimum support value on both random and equal division. We make a comparative analysis with existing techniques, which conclude that MOFSM is significantly better performance that existing one. To get effective and correct results, we have to select proper candidate generation algorithm with correct partition method. In further, we plan to extend our work to extend our result on large database.

REFERENCES

1. A.Khan, X.Yan, and K.-L.Wu(2010),'Towards Proximity Pattern Mining in Large Graphs', in Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, Indianapolis, Indiana, USA, 2010, pp. 867–878.
2. Anand Iyer,Zaoxing Liu, Xin Jin(2018),'Towards Fast and Scalable Graph Pattern Mining HotCloud'18', Proceedings of the 10th USENIX Conference on Hot Topics in Cloud Computing, July 2018.
3. Anchuri, P.,Zaki, M. J.,Barkol,O.,Golan,S. and Shamy M.(2013), 'Approximate graph mining with label costs'. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2013).
4. D. J. Cook and L. B. Holder (2006), Mining graph data. John Wiley & Sons, 2006.
5. Di Fatta, G., and Berthold, M. R.(2006), Dynamic load balancing for the distributed mining of molecular structures. IEEE Transactions on Parallel and Distributed Systems 17, 8 (2006).
6. Gonzalez, J. E., Low, Y., Gu, H., Bickson, D,Guestrin, C. (2012), 'PowerGraph: Distributed graph-parallel computation on natural graphs'. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (2012).
7. Hill S, Srichandan B, and Sunder Raman R(2012). 'An iterative MapReduce approach to frequent subgraph mining in biological datasets'. In Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine (2012).
8. J. Han and J.-R. Wen(2013), 'Mining Frequent Neighborhood Patterns in a Large Labeled Graph', in Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management, San Francisco, California, USA, 2013, pp. 259–268.
9. M. Kuramochi and G. Karypis(2005), 'Finding frequent patterns in a large sparse graph',Data Min. Knowl. Discov., vol. 11, no. 3, pp. 243–271, 2005.
10. Malewicz G, Austern, M. H., Bik A. J., Dehnert, J. C., Horn, I., Leiser N., Czajkowski, G.(2010), 'Pregel: A system for large-scale graph processing'. In Proceedings of the ACM SIGMOD International Conference on Management of Data (2010).
11. Mccune, R. R., Weninger, T., and Madey, G.(2015) 'Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing', ArXiv: 1507.04405 (2015).
12. Oliveira Aparicio, D., Pinto Ribeiro , P. M., and Silva, F(2014), 'F. M. A. Parallel subgraph counting for multicore architectures'. In Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications (2014).
13. Ribeiro, P., and Silva, F(2014),'G-Tries: A data structure for storing and finding subgraphs.'Data Mining and Knowledge Discovery 28, 2 (2014).
14. Shao, Y., Cui, B., Chen, L., Ma, L., Yao, J., and Xu, N.(2014) 'Parallel subgraph listing in a large-scale graph'. In Proceedings of the ACM SIGMOD, International Conference on Management of Data (2014).
15. Teixeira, C. H. C., Fonseca, A. J., Serafini, M.,Siganos, G.,Zaki,M.J.,and Abounga(2015), 'A. Arabesque: A system for distributed graph mining - Extended version'. Technical Report, Qatar Computing Research Institute, 2015.

16. Tian,Y., Balmin, A., Corsten,S.A.,Tatikonda, S. and Mcpherson, J.(2013), ‘ From “think like a vertex” to “think like a graph’. Proceedings of the VLDB Endowment 7, 3 (2013).
17. U. Kang, B. Meeder, E. E. Papalexakis, and C. Faloutsos, “Heigen: Spectral analysis for billionscale graphs,” Knowl. Data Eng. IEEE Trans. On, vol. 26, no. 2, pp. 350–362, 2014.
18. Yan. X., Han J(2002). ‘gSpan: Graph-based substructure pattern mining’. In Proceedings of the IEEE International Conference on Data Mining (2002).
19. Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li(2012), ‘Efficient subgraph matching on billion node graphs’, Proc. VLDB Endow., vol. 5, no. 9, pp. 788–799, 2012.
20. Z. Zhao, G. Wang, A. R. Butt, M. Khan, V. A. Kumar, and M. V. Marathe(2012), ‘Sahad: Subgraph analysis in massive networks using hadoop’, in Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, 2012, pp. 390–401.

AUTHORS PROFILE

Ms. Sadhana Priyadarshini, is a Phd scholar in Department of Computer Science and Engineering at GITAM (Deemed to be University), Vishakhapatnam, India She completed MTech(CSE) from SQA University in 2010.Her research interests in field of Data Mining.

Dr. Sireesha Rodda, is a Professor in the Department of Computer Science & Engineering, GITAM (Deemed to be University). She has 17 years of research experience in the fields of Artificial Intelligence, Data Mining and Machine Learning. She has more than 30 papers published in referred journals