Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.18576/amis/140214

A Novel Distributed Approach for Frequent Subgraphs Mining Across Cloud Computing System (DistFsm)

M. Elshrkawey[1], *Hosam E. Refaat*[1,*] *and Hanan H. Amin* [2]

[1]Dept. of Information System, Faculty of Computers and Informatics, Suez Canal University, Egypt.
[2]Dept. of Computer Science, Faculty of Computers and Informatics, Sohag University, Egypt.

Received: 2 Aug. 2019, Revised: 12 Oct. 2019, Accepted: 25 Oct. 2019
Published online: 1 Mar. 2020

Abstract: In this paper, a novel approach known as DistFSM is presented for the FSM on a single graph. The DistFSM operation performed on a cloud computing system is framed on a set of heterogeneous clusters. Each cluster is a set of homogenous nodes. The input graph is converted into a sparse matrix. This matrix is partitioned horizontally into a sequence of non-equivalent chunks. Each chunk size is computed to be appropriate to the available worker resources in one of the clusters. In each cluster, the chunk is partitioned vertically into equivalent tasks. Each task is assigned to one of the worker nodes. The proposed partitioning method defined as the Hori-Vertical partition and aims to accomplish the load balancing among the different nodes in the different clusters. Each node performs its operation individually without any communication with other nodes. The non-equivalent chunks assigned to the different clusters allow them to finish their operation simultaneously. This strategy increases the resource usage by prohibiting or reducing the waiting time of the high-performance clusters. Finally, the results of all clusters are summarized and submitted to a distributed shared memory of the orchestration node to perform the required aggregation operations.

Keywords: cloud computing, Graph mining, Frequent subgraph mining, Parallel system, and Distributed Shared Memory.

1 Introduction

Data mining is the field of discovering and extracts significant and valuable knowledge from large databases [1]. Recently, there are different applications include various relationships among the objects that are presented in the form of the graphs [2]. The main reason for this development is the ability of the graph to save the complex and enormous data [3]. For examples, in the social networks, the nodes used to represent the users while their relationships are represented by the edges[4]. In the chemical structures, nodes are used to represent the atoms while edges are used to represent the bonds between them [5], and so on. Hence, the term "Big Graph Mining" has been occurred [3,7]. Moreover, a significant research area known as the Frequent Subgraph Mining (FSM) has been established [8]. The FSM is convenient in many domains of the real-world application. For instance, Social Structural Role Mining (SSRM) for dynamic social network has been introduced [9]. This model clarifies that graph mining can offer information concerning somebody's involvement rate amongst his social network and the impact on the interrelated community dynamics [10]. In addition, graph mining is implemented to decide personal compatibility [11]. This model aims to use the available information to measure the matching between the leader and the deputy in an organization, or between the students with their consultant.

Most FSM algorithms include two central phases: the production of the candidate subgraph patterns and the frequency computation [12]. The most imperative factor affects the performance of the FSM algorithm that has occurred when the same candidates are generated more than once. The production of redundant candidates should be avoided to get an efficient algorithm. The second phase is to compute the frequency of the candidates produced to determine the most frequent subgraph among them. In order to compute the frequency of a subgraph, it is required to find the number of isomorphic subgraphs in a database. Therefore, subgraph isomorphism is a vital problem for the applied algorithms due to the problem of NP-complete [13].

In addition, there are several issues considered by the algorithms implemented for FSM [14]. The first issue is

* Corresponding author e-mail: hosam.refaat@ci.suez.edu.eg

related to the type of the input graph. The input data set may be a single large graph or a set of small size graphs called transactions [15]. The frequency computation of a subgraph in the large single graph data set is dissimilar to the transactional dataset. In the transactional dataset, the number of transactions that contain this subgraph is used to compute the frequencies of the candidates. For the single graph-based FSM, the input data is an only one very large graph that has hundreds of thousands nodes. The FSM of the large single graph is depended on the enumerations of the subgraphs occurrence. The mining of the frequent subgraphs based on a single graph is faced by a lot of the difficulties and the huge computations through the matching process of the subgraphs that may be overlapped [16,17].These complications are originated through the necessary means that are required by two core operations. The first operation is the efficient productions of all subgraphs with various sizes. The second operation is the subgraph isomorphism assessment. Assume N and n be the number of vertexes of input graph G and subgraph S, respectively. Normally, the complexity of subgraph productions is $O(2^{N^2})$ and the supported assessment is O (Nn). Therefore, the whole complexity of an FSM algorithm is $O(2^{N^2}.N^n)$, which increases exponentially as the size of the graph database increases.

The second vital issue for the FSM is the categorization of the algorithmic approaches. Actually, there are two different algorithmic approaches for the FSM. They are the apriori-based and pattern growth approaches [18,19]. The purpose of both approaches is to find the frequently occurring subgraph from the available group of small graph sets or a single large graph. However, the two approaches are different in the way of mining the FSPs [20]. The apriori-based algorithms are initiated with small-sized graphs then extended them through joining subgraphs to get well-known subgraphs [21,22]. On the other hand, the algorithms based on the pattern growth approach are started by lengthening the frequent subgraphs beginning from minimal frequent subgraphs and the addition of one edge at each step up to when they are still frequent [23].

The third issue is considered through the graph representation method. Generally, the FSM algorithms use two different techniques to exemplify the graphs. They are the adjacency matrix and the adjacency list. Graphs should be demonstrated uniquely to enable the test of the subgraph isomorphism [24]. However, more than one matrix may be adjacent representing the same graph. So, novel methods are offered such as canonical adjacency matrix (CAM) [25] and min DFS code [26].

The final issue considered in this area has appeared when the existent datasets include very large graph data or classified as big graph data. In this case, the big graph data size cannot be fitted in the memory of a single machine. So, the FSM algorithms applied to the small graph database on a single machine should be modified to be appropriate to deal with the big graph data. This problem can be solved through two basic operations. In the first operation, the large graph database is partitioned into a set of applicable partitions. In the second operation, the partitions are distributed among set system nodes to be performed in a parallel manner by the applied FSM algorithm [27,28]. Actually, the graph partitioning methods are based on the layout of the database. The traditional partitioning methods can be either horizontally or vertically. In the horizontal layout of the database, each tuple includes row id within the items in the transaction [29]. Furthermore, the implementation of the database binds the database items with its tid list. In other words, each item is followed by the list of transaction ids that have an occurrence of this item [30]. In fact, the horizontal layout is the most practicable implementation of the databases.

In this paper, a novel distributed approach for frequent subgraphs mining of a single large graph has been proposed. The proposed approach is implemented on a cloud computing system using a set of worker nodes distributed on a set of parallel heterogeneous clusters. However, within each cluster the worker nodes are homogenous. The proposed model offers a high-level computation based on two levels of master nodes and the set of parallel heterogeneous clusters connected through shared memory. The shared memory is employed to store the intermediate results to reduce the number of input/output operations. In addition, a novel partitioning method known as a hori-vertical method is proposed. It targets to adapt the partition sizes of the distributed subgraphs to be appropriate to the available processing power of each node. Furthermore, the last operation is aimed to achieve the load balancing among all nodes within each cluster. The rest of this paper is organized as follows. Section 2 describes the related studies that are based on the parallel and distributed system, especially for a single large graph. In Section 3, the proposed model and the proposed adaptive partitioning method are presented. Section 4 illustrates comparison analysis and performance evaluation. Finally, the concluding remarks are offered in Section 5.

2 Related Work

Recently, various algorithms have been proposed for a single large graph. These algorithms include SUBDUE [31], [32], GERM [33] and more recently GRAMI [34]. The majority of these algorithms are implemented in sequential execution that necessitates abundant time to mine large datasets. For instance, SUBDUE searches for the possibly frequent substructures in the graph, which can be compressed in perfectly. SiGraM uses the concentrated autonomous set-based approach to compute the support, which can be very costly. For GraMi, it uses a restriction satisfaction-based method to support computation for optimizing the storage of isomorphisms in memory by identifying automorphism groups.

On the other hand, the parallel and distributed computing methods are employed to speed up the computation. Two main frameworks are applied to perform parallel computing. They are the distributed Map-Reduce [35], and MPI (Message Passing Interface) [36]. Distributed map-reduce is based on employing the task which means to mine the frequency distribution of all vertex of the subgraphs up to specific k vertices. In contrast, for FSM, there is no constraint on the size of the sub-graphs to be mined, but the minimum support can be used to trim the search space [37]. Generally, the whole implemented parallel FSM algorithms that are based on the MapReduce are designed for graph transaction. They are not applied to the single graph. In addition, they suffer from the bottlenecks, since they spend a lot of time for moving the data/processes into and out of the disk during repeated execution of the algorithms. Moreover, several of these algorithms cannot be applied for mining through the subgraph extension [38,37]. The users should supply the size of subgraph as input. However, the methods that are established on the MPI such as DistGraph [36] have a preferred performance due to their dependence on the interconnected HPC (High-performance Computing) machines. Nevertheless, these methods suffer from the less availability of these machines for most of people. In this research, a distributed subgraph mining approach has been proposed. It is based on partitioning the input graph using a novel hori-vertical partition method. In 2012, Khadija Belbachir and Hafida Belbachir [41] proposed a parallel version for a sequential algorithm. The implementation of this parallelism doesn't require much communication between the nodes. But, the applied method uses a non-adaptive partitioning method. The database is divided horizontally into partitions of the same size based on the memory size. Therefore, it suffers from the inefficient use of the available system resources.

3 The Proposed Hori-Vertical Partition Method

In the proposed model, the input graph is passed by two basic operations methods. In the first operation, the input graph should be converted into a sparse matrix, as shown in Figure 1. In the sparse matrix, each 1 represents the existence of the edge between two vertices in the graph. On another hand, 0 means that there is no connection between the two vertices in the graph.

In the second operation, a novel partitioning method is offered to prepare the sparse matrix for mining processes. The sparse matrix is partitioned in two dimensions (horizontally and vertically). So, the partitioning method is known as the Hori-Vertical partition method (H-VP). In the horizontal dimension, the sparse matrix of the input graph is divided into a set of chunks, as shown in figure 2. The chunks are cut as a set of rows with different sizes. Each chunk is assigned to a specific cluster among all

Item-Vectors				
0	1	……..	0	1
1	0	……..	1	1
		. . . . .		
1	1	……..	1	0

Fig. 1: Sparse Matrix

heterogeneous clusters in the proposed model. Hence, the size of the chunk assigned to each cluster is determined to be appropriate to the all available resources of the workers within that cluster. Each chunk size depends on the available parallel node resources. Within each cluster, the assigned chunk is divided vertically into set of slices (tasks) to be distributed over the parallel nodes, as shown in figure 2. Each task is a set of rows and columns which represented by $w_1 w_2 \ldots . w_n$. Moreover, each task is cut with a specific size to represent the set of the itemsets that is executed on a specific worker. However, the size of each task is appropriate to the available resources of that specific worker. In the following subsection, the relation between the computation of chunk size and the available resource of a specific cluster is clarified. By the same way, the relation between the task size and the resource of a specific worker is illuminated through the proposed model details.

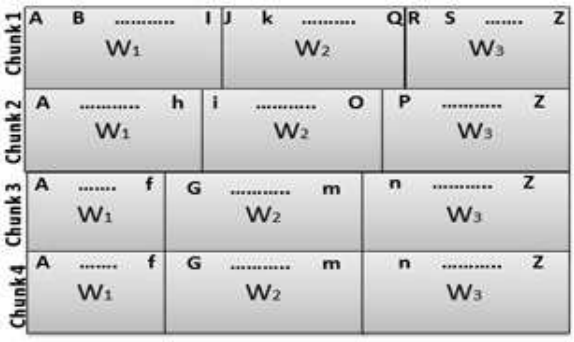

Fig. 2: Hori-vertical partitioning

However, the H-VP technique is based on dividing the database to be appropriate to the VMs (workers) capability. In the parallel execution procedures of the proposed model, the main task of each worker is to count the frequency of itemset in the assigned task individually. Generally, the H-VP is created to reduce the dependencies between parallel processing.

Suppose that the graph database G is partitioned horizontally into h chunks and vertically into v partitions (tasks) at the same time. Then the database can be written as:

$$G = \bigcup_{i=1}^{h} \bigcup_{j=1}^{v} g_{i,j} \quad (1)$$

Lemma 1.*If the graph database D is partitioned horizontally and vertically simultaneously, for every frequent subgraph $x, y \in L_k$. The new subgraph $g = x \wedge y$ (join x and y) will not be frequent subgraph if x and y are not locally frequent subgraph in any of the same horizontal partition.*

*Proof.*Suppose $x, y \in L_k$, $g = x \wedge y$ and $g \in L_{k+1}$. Also, let $q_x = \{DB_{i,j} | x$ locally frequent subgraph $\}$ and $q_y = \{DB_{k,h} |$ y locally frequent subgraph $\}$. Also, $q_x \cap q_y = \emptyset$. However, the subgraph g should be counted in some transaction to be frequent ($q_x \cap q_y \neq \emptyset$). In other words, the candidate subgraph g should be locally frequent in at least one chunk.

4 The Proposed Mining Model

The model of the proposed approach is designed to enhance the operations of the frequent subgraphs deduced from the partitioning of the main input graph. In order to achieve this objective, distributed Frequent Subgraphs Mining (DistFSM) is proposed as shown in Figure 3. In this model, an adaptive load balancing technique is included. This proposed model is implemented among a set of heterogeneous clusters. Each cluster includes a set of homogenous VMs.

4.1 The Architecture of The Proposed Model

The architecture of a novel distributed approach for frequent subgraphs mining across cloud computing system is congregated in two main modules. They are the module of the orchestration node and the cluster module. The orchestration node module includes five main components. These components are the chunk creator (CC), the head-load balance (H-LB), the frequent aggregation, the candidate generation and the distributed shared memory (DSM). On the other hands, each cluster module includes four main components. They are the tasks creator, the VMs, the chunk frequency and the cluster-load balancing (CL-B).In the following subsection, the functions performed by each component in each module is clarified. The function of each component is explained according to its sequence in the whole operation of the proposed model.

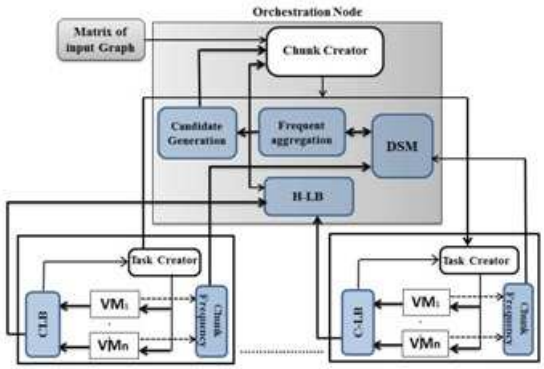

Fig. 3: The Adaptive Frequent Subgraphs Mining.

4.1.1 Chunk Creator (CC)

Before the start of CC operations, the input graph is converted into the sparse matric form. The CC includes two phases; Initial phase, and operational phase. In initial phase, the CC generates a set of the same size chunks, as shown in Algorithm 1. The number of chunks is equivalent to the number of the clusters. Each chunk is a set of rows. These chunks are identified as test chunks because their size is selected randomly. Each one of these chunks is passed to one of task creator (TC) in the parallel clusters. In the initial phase, the same size of the chunks is submitted to H-LB. On the other hand, the operational phases include two cases. In the first case, the chunks are cut from the original input sparse matrix. In the second case, the chunks are created from candidate aggregation after the count frequencies are performed on the first candidate itemsets C_1. In both cases of the operational phase, the chunk size of each cluster is computed according to the available resources of each cluster. The computation method of the sizes of the chunks is explained in the subsection of the cluster load-balance. In the following, the pseudo-code of the chunks creator is introduced.

4.1.2 Task Creator (TC)

The Task Creator (TC) receives its dedicated chunk and divides it into s set of equivalent size tasks. These tasks are distributed among the VMs of the cluster. The main objective of the VM is to count the frequencies of its dedicated candidate itemsets ($c_1^i \subset C_1$). In addition, each VM submits its consumed execution time of its assigned task to the cluster load-balance. In the following, the pseudo code of the Task creator is presented.

Algorithm 1 Chunk Creator Algorithm

input:
G //The big graph
CL
Output
HG //graph chunk (horizontal graph partition)

/* initializing the first candidate itemset.*/
if $(C_k \neq \phi)\&(k = 1)$ **then**
  $C_k = u$
end if
{ initial phase cluster loop}
for $(\forall cl_i \in CL)$ **do**
  $tstCh_i = createTstChunk(G)$
  $cl_i.taskCreator(tstCh_i)$
end for
$remGraph = G - \sum_{i=1}^{|CL|} tstCh_i$ //the remaining un-distributed partition of graph G
/* Operational phase */
$opSiz_i = HLB.ClusterPartitionSize(cl_i)$ /* receive the operational chunk size for cluster I from head load-balance */
//distributed the remaining of the graph
while $(remGraph \neq \phi)$ **do**
  for $(\forall cl_i \in CL)$ **do**
    $ch_i = createOprChunk(G, opSiz_i, C_k)$ /*create operational chunk for each cluster i */
    $cl_i.taskCreator(Ch_i, C_k)$ // send the chunk for the task creator i
  end for
end while
$K++$ // finding the next frequent itemsets
$C_k = candidateGenerator.Receive(K)$ /* take the k^{th} candidate Itemsets from the candidate generator */
while $(C_k \neq \phi)$ or$(remGraph \neq \phi)$ **do**
  for $(\forall cl_i \in CL)$ **do**
    $ch_i = createOprChunk(G, opSiz_i, C_k)$ /*create operational chunk for each cluster i */
    $cl_i.taskCreator(Ch_i, C_k)$ // send the chunk for the task creator i
  end for
  $remGraph = G - \sum_{i=1}^{|CL|} tstCh_i$
  if $(remGraph \neq \phi)$ **then**
    $C_k = CandidateGenerator.Receive(k)$ /* take the k^{th} candidate Itemsets from the candidate generator */
  end if
end while

4.1.3 The chunk Frequencies

The main objective of the chunk frequencies is to receive the count frequencies from the all VMs within the cluster. It aims to perform the local frequent itemset that is passed and stored in the distributed shared memory. In the following, the pseudo code of the chunks frequencies is presented.

Algorithm 2 Task Creator Algorithm

input:
HG /* graph chunk (horizontal graph partition)*/
C_k /* set of candidate */

$TG = null$ //task group
$k = 0$ // k Number of candidate in a task
$k = \frac{|C_k|}{|v|}$ /* where v is the number of VMs in the cluster.*/
$TG = createTasks(HG, k)$ /*create tasks has standard size*/

while ($TG \neq null$) **do**
  for $(\forall v_i \in v)$ **do**
    $t = null$
    $t = takeTask(TG)$ /* where $t \in TG$
    $InsertTasks(t, v_i.queue)$
  end for
end while

Algorithm 3 Chunk Frequency Algorithm

// loop on VMs cluster
for $(v_i \in v)$ **do**
  if $(v_i.Finish = true)$ **then**
    $C_k.localCount = t_i.condidCount(t, v_i.queu)$
  end if
end for
$DistributedSharedMemory.Write(C_k.localCount)$

4.1.4 The Cluster-Load Balancing

Actually, the load balancing is accomplished through a set of procedures that are performed after the completion of each task by its assigned VM. Each VM submits its consumed execution time to the Cluster Load-Balance (C-LB). In the C-LB, the total execution times of the all VMs in each cluster are computed as in the following equations.

$$TP_i = \sum_{j=1}^{m_i} T(VM_j) \tag{2}$$

Where:
$T(VM_j)$: is the total execution time of VM_j.
m_i: is number of VMs in cluster i.
TP_i : is the total of the execution time for cluster i.

Hence, λ_i is the expected execution time for each row in the chunk assigned to the cluster i, and can be computed in the C-LB as follows:

$$\lambda_i = \frac{TP_i}{|C|} \tag{3}$$

Where:
$|C|$: is the previous chunk size.
After these computations, the C-LB submits the expected execution time λ_i for each row in the chunk assigned to the cluster i to the Head-Load Balance (H-LB).

4.1.5 H-LB Algorithm

H-LB algorithm receives the set of values for expected execution times $(\lambda_i, \forall i)$ for all clusters in the system. The main objective of the H-LB is to make all clusters finish their work on their chunks simultaneously. So, the new chunk sizes are determined based on the slowest cluster in the system. Originally, the high performance clusters may be idle for waiting the slowest clusters to finish its work. Hence, the optimal solution for this problem is to create an adaptive job for each cluster. This means that the chunks assigned to the different clusters have different sizes. This strategy increases the resource usage by prohibiting or reducing as much as possible the waiting time of the high-performance clusters. Hence, the H-LB computes the new chunk size for each cluster in the operational phase as follows.

H-LB selects the normalized λ using the following equation: $\lambda = \max_i^n \lambda$; In other words, λ depends on the least cluster performance.

Where:

n: is the number of system clusters.

Hence, the new chunk size for each cluster can be computed as follows.

$$Z_i = \frac{\lambda}{\lambda_i} Z$$

Z: represents the previous chunk size or the chunk size for the previous iteration Z_i: represents the new chunk size for the next iteration

The new computed chunk size of each cluster is provided to the chunk creator to be used in cutting of the new chunks. Generally, the chunk size computation should be performed after the first execution of the test chunks. However, the chunk computations should be avoided as possible in the operational phase. The main reason of this avoidance is to minimize the predicted overhead due to the chunk size computation. Hence, the chunk sizes computations are not performed for each iteration. Actually, they are performed for one of two event occurrence. First, when any VM in any cluster is failed. Second, if an additional machine is added to any of the cluster. In this case, the C-LB computes the chunk sizes are submitted to the H-LB.

Algorithm 4 The pseudo code of the C-LB

1: **Output**
2: λ_i // the expected execution time for each row in cluster j
3: Collect the execution time for each VM (v_i^j) in the cluster $V^j (v_i^j \in V^j)$
4: Compute the total execution time $TP^j = \sum_{\forall i v_i^j}$
5: Compute the execution time for each row $\lambda_j = \frac{TP_i}{|C|}$
6: Send λ_j to H-LB
7: If allocated resources changed go to step 3

Algorithm 5 The pseudo code of the H-LB

1: **input:**
2: $\bigwedge = \{\lambda_1, ..., \lambda_n\}$ /* set of expected row execution time for each in cluster in the system */
3: Z //initial chunk size or the chunk size for the previous iteration
4: **Output**
5: $\xi = \{Z_1, ..., Z_n\}$ /* the chunk size for each cluster*/
6: $\lambda = \max_{i=1}^n (\lambda_i)$
7: **for** $(i = \{1, ..., n\})$ **do**
8: $Z_i = \frac{\lambda}{\lambda_i} Z$
9: **end for**
/* $\forall \lambda_i \in \bigwedge$ */
10: **if** (λ_i changes) **then**
11: Go to step 6
12: **end if**
13: send $\xi = \{Z_1, ..., Z_n\}$ to the Chunk Creator(CC)

4.1.6 Frequent Aggregation

By each iteration end, the frequencies counts of the candidate itemsets that represent the different tasks are sent to the Chunk Frequency (CF). The CF aggregates the frequent results that represent the whole chunk in a single chunk-frequent-vector and send it to be saved in the Distributed Shared Memory (DSM). For the successive iterations, the frequent results of each chunk are also represented by its own chunk-frequent-vector and send to the DSM. Hence, the main function of the frequent aggregation is to aggregate the similar frequent itemsets from the all successive chunk-frequent-vector. This operation of the chunk-frequent-vector aggregation is performed by the end of each repetition. After the whole chunk-frequent-vector of the all chunks has been completed, the total frequencies of the itemsets are determined by the frequent aggregation. Each frequency of the itemset is compared by the minimum support value(S) to determine the set of globally frequent itemsets (L_k). Hence, the frequent aggregation sends the globally frequent itemsets$(L_k)$to the Candidate Generation (CG). The CG uses the globally frequent $K-$ itemsets (L_k) to generate the new candidate $(K+1)-$ itemsets(L_{k+1}). Eventually, the new candidate $(K+1)-$ itemsets are sent to the Chunk Creator (CC) to use these candidate in the successive operations to generate the next and new chunks.

5 Experiment and Result Discussion

All experiments are conducted on two heterogeneous clusters formed on a cloud computing system [43]. The first cluster includes 4 homogenous workers nodes. Each node resource comprises of 5 GB of RAM, Intel (R) Xeon (R) CPU with 1.8.00 GHz. The second cluster includes 4 homogenous workers nodes. Each node resource

comprises of 3 GB of RAM, Intel (R) Xeon (R) CPU with 2.2 GHz. The whole nodes in both clusters are run on Windows server 2008 R2 Enterprise. In addition, the resources of the orchestration node are formed of 8 GB of RAM, Intel (R) Xeon (R) CPU with 3.8 GHz. The whole implemented algorithms are coded in Matlab 2012. The proposed approach DistFsm is compared with different research methods such as traditional parallel FSG [42], Belbachir [41] and DistGraph methods [36]. The comparisons among the different methods are performed on a different database set for a large single graph. These databases include ERDOS981, ERDOS971, ERDOS992, ODLIS, California search engine [40], PDB1 [44], Paten [39] and YouTube [6]. The different characteristics of the different datasets which include the size of graphs, the number of edges in the dataset are described in Table 1. The performance of the compared methods and the proposed approach DistFsm are determined by measuring the execution times in seconds against the minimum support threshold at different values. The result values of the different comparisons methods using the different datasets are also shown in Table 1.

The results of the comparisons among the different methods that were performed on the different databases are shown in the following sequence of figures 4. For instance, Fig. 4 shows the performances result for using the database of ERDOS981. Fig.5 shows the performances result for using the database of ERDOS971. Fig. 6 shows the performances result for using the database of ERDOS992. Fig. 7 shows the performances result for using the database of ODLIS, and so on. The figures show that, the performance of the proposed algorithm DistFSM is faster than the traditional parallel FSG, Belbachir, and DistGraph algorithms for the all values of minimum supports. In addition, the number of results grows exponentially when the support threshold Θ decreases. Thus, the running time of all compared methods also grows exponentially. Moreover, when the database size is increased as in the YouTube dataset and lower minimum support Θ (0.01), the minimum execution time for the all compared methods is accomplished by the DistFSM. On other hand, as the size of the database is decreased as in the EDROS981 and the minimum support Θ (0.6) is increased, the DistFSM is still the fastest among the all compared methods due to less waste of the resource employment in the DistFSM method. As a numerical example that demonstrates this important result, for the dataset "ODLIS" with minSup = 0.6. The run time of traditional parallel FSG is 0.046225 (s), Belbachir is 0.008072 (s), DistGraph is 0.000452 (s), while that of DistFSM is 0.000353 (s), thus DISTFSM algorithm doesn't waste the time in the mining to get the all frequent subgraphs compared with other methods, as shown in Fig 7.

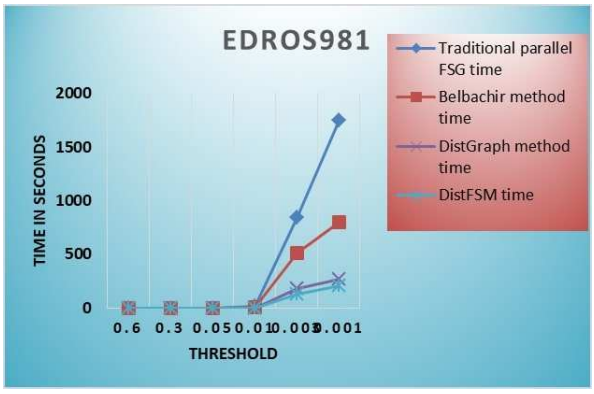

Fig. 4: Runtimes of Traditional parallel FSG, Belbachir, DistGraph and DISTFSM methods for ERDOS981 dataset

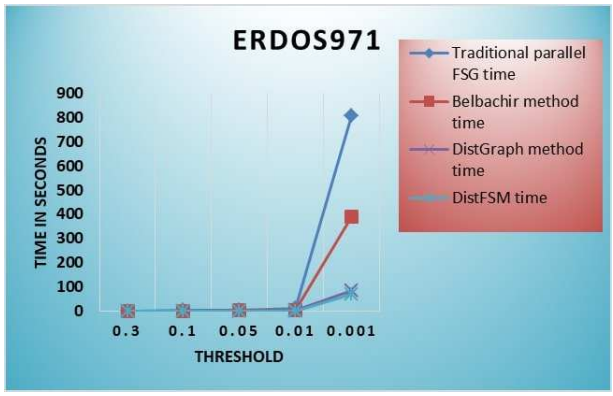

Fig. 5: Runtimes of Traditional parallel FSG, Belbachir, DistGraph and DISTFSM methods for ERDOS971 dataset

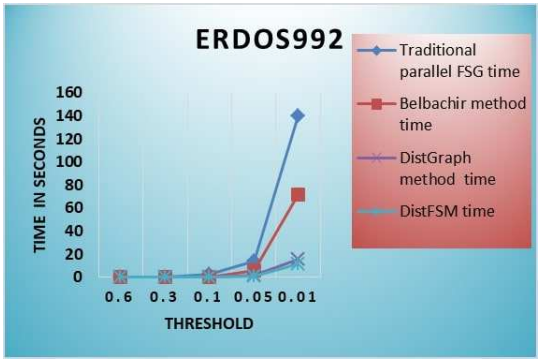

Fig. 6: Runtimes of Traditional parallel FSG, Belbachir, DistGraph and DISTFSM methods for ERDOS992 dataset

Table 1: Characteristics of the experimental results time (in second)

Dataset	Num of Vertex	Num of Edges	Minimum (?) support (Θ)	Traditional time (in second) parallel FSG	Belbachir method	DistGraph method	DistFSM time
ERDOS981	347	483	0.6	0.038353	0.004271	0.000758	0.000544
			0.3	0.088042	0.009905	0.003761	0.0027
			0.05	0.597076	0.063127	0.026694	0.021032
			0.01	16.20034	2.936635	0.847732	0.608628
			0.003	840.2437	511.7998	180.7032	129.735
			0.001	1746.229	798.0491	268.206	204.691
ERDOS971	448	472	0.3	0.089511	0.00911	0.003992	0.003146
			0.1	0.449326	0.043532	0.015995	0.012169
ERDOS971	448	472	0.05	1.095346	0.258724	0.01651	0.012786
			0.01	8.292601	2.211906	0.134122	0.105615
			0.001	809.8722	389.7837	85.17391	69.08383
ERDOS992	524	6100	0.6	0.237611	0.026358	0.003997	0.003026
			0.3	0.337428	0.038415	0.012885	0.009251
			0.1	2.857993	0.381867	0.042332	0.033247
			0.05	13.71754	5.884974	1.628219	1.308947
			0.01	139.8394	71.38091	15.75073	11.30821
ODLIS	2896	2909	0.6	0.046225	0.008072	0.000452	0.000353
			0.3	0.084817	0.01882	0.005622	0.004503
			0.01	3.690695	0.056807	0.020393	0.016491
			0.05	8.691193	4.182621	0.674939	0.484572
			0.001	233.3989	136.2786	58.10711	47.31791
California search engine	16150	9664	0.6	1.090441	0.122692	0.003348	0.003028
			0.3	16.60339	5.707027	1.287254	1.028796
			0.1	257.4042	111.4921	31.88925	24.28754
			0.01	659.5966	350.9257	122.4421	98.36775
			0.001	839.8848	533.1789	217.317	176.613
PDB1	20,226	83,356	0.3	19.77699	14.23943	8.998529	6.891181
			0.1	290.6977	209.3024	132.2675	101.292
			0.01	879.0151	632.8909	399.9519	306.2879
Paten	2,942,15	14,275,931	0.3	478.3892	344.4431	217.6671	166.6921
			0.1	1567.239	1128.412	713.094	546.0956
			0.01	4714.769	3394.634	2145.22	1642.835
YouTube	4,584,572	23,236,009	0.3	969.16	753.7952	549.9683	268.0095
			0.1	5597.589	4590.264	3636.903	1253.603
			0.01	9923.387	8424.798	5060.141	3109.306

6 Conclusion And Future Work

In this paper, DistFSM model is introduced for finding the frequent subgraphs mining from a single large graph database. The proposed model is implemented over a cloud system. It distributes the load over the heterogeneous cloud resources in a balanced manner. The input graph is formed in a sparse matrix form. The sparse matrix is partitioned using a new partitioning strategy called "Hori-Vertical". This strategy is aimed to accomplish the load balancing among the different nodes in the heterogeneous clusters. First, the matrix is partitioned horizontally into a set of non-equivalents chunks. Each chunk has the appropriate size to the cluster available resources. Subsequently, the chunk is partitioned vertically into equivalent tasks. These tasks are independent tasks, which can be processed in the cluster VMs without communication overhead among these nodes. The non-equivalent chunks assigned to the different clusters allow them to finish their operation simultaneously. This strategy increases resource utilization over heterogeneous resources by eliminating the idle time for the different performance clusters. Finally, the results of clusters are aggregated and submitted to a distributed shared memory in an orchestration node to perform the global aggregation operations. At last, the experiments show that our methods outperform the DistFSM model. In future work, apply DistFSM model on the stream of data, which can be real-time insights in Big Data.

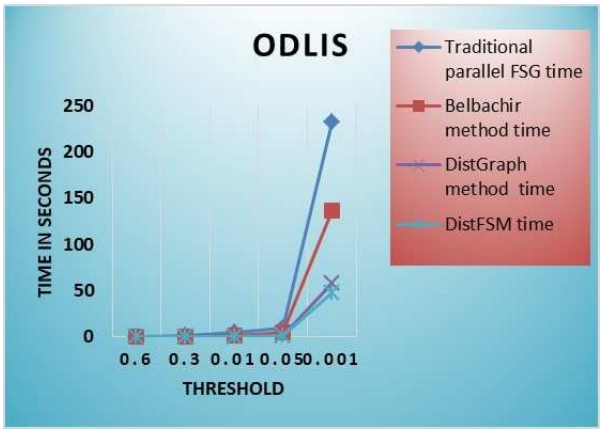

Fig. 7: Runtimes of Traditional parallel FSG, Belbachir, DistGraph and DistFSM methods for ODLIS dataset

Fig. 8: Runtimes of Traditional parallel FSG, Belbachir, DistGraph and DistFSM methods for California search engine

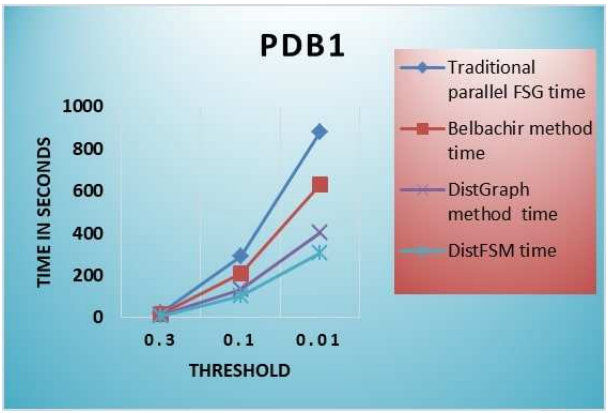

Fig. 9: Runtimes of Traditional parallel FSG, Belbachir, DistGraph and DistFSMmethods for PDB1 dataset

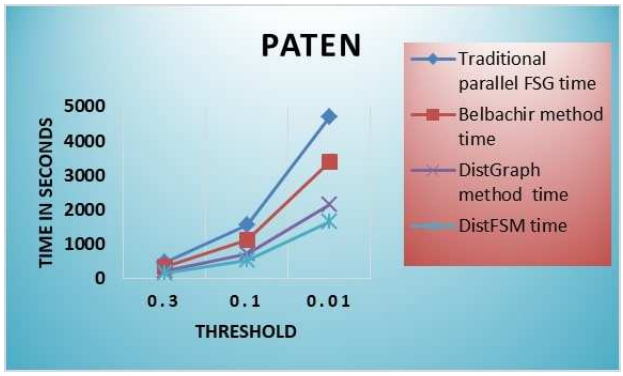

Fig. 10: Runtimes of Traditional parallel FSG, Belbachir, DistGraph and DistFSM methods for PATEN dataset

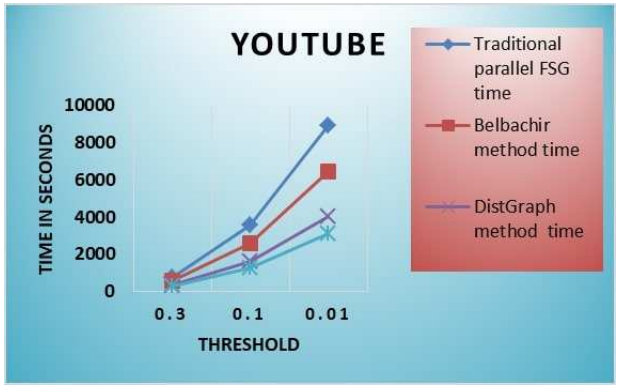

Fig. 11: Runtimes of Traditional parallel FSG, Belbachir, DistGraph and DistFSM methods for YouTube dataset

References

[1] J. Han, J. Pei and M. Kamber, “Data mining: Concepts and Techniques”, Morgan Kaufmann Publishers is an imprint of Elsevier, LONDON UK, 279-323, 2011.
[2] E. Abdelhamid, M. Canim, M. Sadoghi, B. Bhattacharjee, Y.-C. Chang, and P. Kalnis: “Incremental frequent subgraph mining on large evolving graphs”, IEEE Trans. Knowl. Data Eng., vol. 29, no. 12, 2710–2723, 2017.
[3] I. Atastina, B. Sitohang, G.A.P. Saptawati, and V. Moertini: ”A review of big graph mining research,” IOP Conference Series: Materials Science and Engineering., 180, 12-56 , 2017
[4] A. Brandstädt: ”Efficient domination and efficient edge domination: A brief survey,” presented at the Conf. Algorithms Discrete Appl. Math., 10743, 1–14, 2018.
[5] T. Meinl, M. Worlein, I. Fischer and M. Philippsen: “Mining molecular data sets on symmetric multi-processor systems”, Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics (SMC’06), IEEE, vol. 2, pp. 1269-1274, 2006.
[6] The Youtube dataset (netsg.cs.sfu.ca/youtubedata)
[7] U. Kang, L. Akoglu, and D.H. Chau: “Big graph mining for the web and social media: algorithms, anomaly detection, and applications,” WSDM ’14: Proceedings of the 7th ACM

International Conference on Web Search and Data Mining, 677–678, 2014.
[8] T. Alam, S.A. Zahin, M. Samiullah, and C.F. Ahmed: ”An Efficient Approach for Mining Frequent Subgraphs,” presented at the International Conference on Pattern Recognition and Machine Intelligence Intell, 486–492, 2017.
[9] A. Abnar, M. Takaffoli, R. Rabbany, and O.R. Zaiane: ”SSRM: Structural social role mining for dynamic social networks,” in Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on, 289–296, 2014.
[10] A. Pak and P. Paroubek : ”Extracting Sentiment Patterns from Syntactic Graphs”, In Social Media Mining and Social Network Analysis: Emerging Research; IGI Global: Hershey, PA, USA, 1–18, 2013.
[11] F. Qiao and H. Wang: ”Computational Approach to Detecting and Predicting Occupy Protest Events”, In Proceedings of the 2015 International Conference on Identification, Information, and Knowledge in the Internet of Things (IIKI), Beijing,China, 94–97, 2015.
[12] K. Lakshmi and T. Meyyappan:” A comparative study of frequent subgraph mining algorithms”, International Journal of Information Technology Convergence and Services, 2 (2), 23-29, 2012.
[13] M.R. Garey and D.S. Johnson: ”Computers and Intractability; A Guide to the Theory of NP-Completeness”, W. H. Freeman & Co., New York, USA , 29-45, 2002.
[14] C. Jiang, F. Coenen and M. Zito: “A survey of frequent subgraph mining algorithms”. Knowl. Eng. Rev., 28, pp. 75–105, 2013.
[15] C. Jiang, F. Coenen, and M. Zito, ”Frequent Subgraph Mining on Edge Weighted Graphs,” presented at Bach Pedersen T., Mohania M.K., Tjoa A.M. (eds) Data Warehousing and Knowledge Discovery. DaWaK 2010. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 6263, 77–88, 2010.
[16] S. Ghazizadeh and S. Chawathe: ”Seus: Structure extraction using summaries”, International Conference on Discovery Science, Springer,71-85,2002.
[17] M. Kuramochi and G. Karypis: ”Finding frequent patterns in a large sparse graph”, Data Mining and Knowledge Discovery, 11(3), 243-271, 2005.
[18] A. Dhiman and S.K. Jain, ”Frequent subgraph mining algorithms for single large graphs—A brief survey”, presented at 2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA), Dehradun, India, Spring, 1–6, 2016.
[19] E. Abdelhamid, M. Canim, M. Sadoghi, B. Bhattacharjee, Y.-C. Chang, and P. Kalnis: ”Incremental frequent subgraph mining on large evolving graphs”, IEEE Trans. Knowl. Data Eng., 29 (12), 2710–2723, 2017.
[20] C. Aggarwal and H. Wang, ”Graph data management and mining: A survey of algorithms and applications”, in Managing and Mining Graph Data. Advances in Database Systems, Boston, MA, USA: Springer,40, 13–68, 2010.
[21] M. Kuramochi and G. KarypisG: ”An efficient algorithm for discovering frequent subgraphs”, IEEE Transactions on Knowledge and Data Engineering, 16(9), 1038- 1051,2004.
[22] C. Jiang, F. Coenen and M. Zito, ”A survey of frequent subgraph mining algorithms”, The Knowledge Engineering Review, 28(1), 75-105, 2013.
[23] X. Yan, J. Han: ”Close graph: mining closed frequent graph patterns,” Proceedings of the Ninth ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 286-295,2003.
[24] T. Meinl, M. Worlein, I. Fischer and M. Philippsen: ”Mining molecular datasets on symmetric multi-processor systems”, Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics (SMC’06), IEEE, 2, 1269-1274, 2006.
[25] J. Huan, W. Wang and J. Prins: ”Efficient mining of frequent subgraphs in the presence of isomorphism”, Proceedings of the 2003 International Conference on Data Mining, IEEE, 549-552, 2003.
[26] X. Yan and J. Han, ”gSpan: graph-based substructure pattern mining”, Proceedings of International Conference on Data Mining, IEEE, 721-724, 2002
[27] M. Bhuiyan and M. Hasan: ”An iterative mapreduce based frequent subgraph mining algorithm”, IEEE Transactions on Knowl- edge and Data Engineering, 27(3), 608-620, 2015.
[28] M. Sangl and P. Bhavsar: ”gSpan-H: An iterative mapreduce based frequent subgraph mining algorithm”, International Journal of Advance Research and Innovative Ideas in Education, 2(5), 169-177, 2016
[29] A. Savasere, A. Omiecinski and S. Navathe: ”An Efficient Algorithm for Mining Association Rules in Large Databases”, In Proc.Of the 21st Int’l VLDB Conference, Zurich, Switzerland, 432-444, 1995.
[30] M.J. Zaki, S. ParthaSarathy and M. Ogihara, W. Li: ”New Algorithms for Fast Discovery of Association Rules”, KDD: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, 283–286, 1997.
[31] B. Holder, J. Cook and S. Djoko: ”Substructure discovery in the SUBDUE system”, Proceedings of the AAAI Workshop on Knowledge Discovery in Databases , 169-180,1994.
[32] M. Kuramochi and G. Karypis: ”Finding Frequent Patterns in a Large Sparse Graph”, Knowl. Discover, 11(3), 243–271, 2005.
[33] M. Berlingerio, F.Bonchi,B. Bringmann and A. Gionis: ”Mining graph evolution rules”, In Machine Learning and Knowledge Discovery in Databases; MIT Press: Cambridge, MA, USA, 115–130, 2009.
[34] M. Elseidy, E. Abdelhamid, S. Skiadopoulos and P. Kalnis: ”GraMi: Frequent Subgraph and Pattern Mining in a Single Large Graph”, Proc. VLDB Endow, 7(7), 517–528, 2014.
[35] K. Wang, X. Xie, H. Jin, P. Yuan, F. Lu and X. Ke: ”Frequent Subgraph Mining in Graph Databases Based on MapReduce”, In Advances in Services Computing, Proceedings of the 10th Asia-Pacific Services Computing Conference (APSCC2016), Zhangjiajie, China, 16–18 November 2016; Springer: Berlin, Germany, 464–476, 2016.
[36] N. Talukder and M.J. Zaki: ”A distributed approach for graph mining in massive networks”, DataMin. Knowl. Discov, 30(5), 1024–1052, 2016.
[37] S. Shahrivari, S. Jalili: ”Distributed discovery of frequent subgraphs of a network using MapReduce”, Computing, 97(11), 1101–1120, 2015.
[38] Y. Liu, X. Jiang, H. Chen, J. Ma, X. Zhang: ”MapReduce-Based Pattern Finding Algorithm Applied in Motif Detection for Prescription Compatibility Network”. In: Dou Y., Gruber R., Joller J.M. (eds) Advanced Parallel Processing Technologies. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg , 5737, 341–355, 2009.

[39] The Patent dataset (www.nber.org/patents).
[40] http://vlado.fmf.uni-lj.si/pub/networks/data.
[41] K. Belbachir and H. Belbachir: "New algorithm for data mining on Grid Computing", International Review on Computer and Software (IRECOS), Praise Worthy Prize, 11(12), 1105-1121, 2016.
[42] K. Rajdeep, Puneet and Deepika: "Apriori Algorithm for Mining Frequent Patterns using Parallel Computing: Survey", International Journal of Advanced Research in Computer Science and Software Engineering, 6(5), 822-824, 2016.
[43] http://www.ec3-egypt.com.
[44] RCSB Protein Data Bank (www.rcsb.org/pdb).

Mohammed A. El-Shrkawey received his B.Sc. in Electrical engineering from the Military Technical Collage, Cairo in 1987. He received his M. Sc. in Computer Engineering from Faculty of Engineering, Al Azhar University, Cairo in 2002. He received his Ph. D. in Network Security from Faculty of Computers & Informatics, Cairo University in June 2007. He is currently a lecturer in Faculty of Computers & Informatics, Suez Canal University. Ismailia, Egypt. His current research interests are Networks, Modeling, simulation, and Image Processing.

Hosam E Refaat has graduated from the Faculty of Science, Assuit University, Egypt, in 1998. In October 2006, he finished his Master degree in the field of distributed systems from the faculty of Science, Cairo University, Egypt. Currently, he is a lecturer in Faculty of Computers & Informatics, Suez Canal University. Ismailia, Egypt. His current research interests are Parallel Systems, Cloud Computing, Edge Computing, and Data mining.

Hanan H. Amin has graduated from the faculty of Science, Sohag University, Egypt, in 2007. In 2012, she received her M.Sc in computer Science , from faculty of Science, Sohag university. In 2018, she finished her Ph. D. in graph mining and cloud computing , from faculty of Science, Sohag university. Currently, she is a Lecturer in Computer Science, faculty of Science, Sohag university. Her current research interests are Parallel System, Neural Networks, Cloud Computing , Graph theory and Graph mining.