Standard Framework for Comparison of Graph Partitioning Techniques

Mudasser Iqbal[1], Dr. Saif-ur-Rahman[2]

[1,2]*Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST) Karachi, Pakistan*

[1]mudassar.iqb@gmail.com

[2]saif.rahman@szabist.edu.pk

Abstract – Graph Partitioning is used to distribute graph partitions across nodes for processing. It is very important in the pre-processing step for distributed graph processing. In Math and Computer Science, many different distributed graph processing solutions use different partitioning approaches. This research deals with the identification of issues associated with the different graph partitioning approaches. This research paper compared the different graph partitioning solution (GraphLab, ParMetis, PT-Scotch) by applying them on different real world datasets and obtained the I/O and partitioning variation between them using different technique. This paper describes the procedure of configuring the GraphLab on Ubuntu OS and applying partitioning and pagerank techniques on it. Pmetis and Kmetis are two graph partitioning algorithms used in ParMetis. These algorithms were on same graph for different numbers of partitions and obtained the I/O and partitioning comparison between Pmetis and Kmetis. Different vertex cut strategies are also discussed in this paper. In this paper, the behavior of PowerGraph and PT-Scotch was explored while working on a very large datasets.

***Keywords* – Graph databases, Graph Partitioning, Partitioning Solutions, Edge cuts and vertex cuts, Partitioning Techniques**

I. INTRODUCTION

Many large graphs have emerged in recent years e.g. Facebook, Twitter, LinkedIn Data and etc. The notable graph is the WWW which now contains more than 50 billion web pages and more than one trillion unique URLs [1]. One of the biggest challenges is how to partition a graph so that it can be deployed on a distributed system [1]. Graph partitioning makes it possible to partition a graph G into smaller graphs (g1, g2 ….gn), and it is useful in distributed architectures. Graph partitioning is very useful for load balancing while minimizing communication. It is NP complete problem because its solution is based on some heuristics. Graph partitioning techniques attempt to ensure following characteristics to create good quality partitions [2]:

- Number of nodes should be the same in every partition.
- Cross-partition edges of each partition should be reduced to a minimum number.
- Achieve spatial locality for graph processing.
- In case of shared memory in graph processing, memory contention should be minimized.
- Maximize parallelism.
- Communication and latency should also be reduced.

In graph, partitioning problems are defined on data in the form of G = (V, E), where V are vertices of graph and E are the edges in a graph connecting two vertex. Figure 1 shows a graph partitioning of a directed graph having 6 vertices V= {A, B, C, D, E, F} and 9 edges which are connecting them {(A, B), (B, D), (B, F), (C, E), (C, A), (E, A), (E, F), (F, A), (F, D)}. The vertices or set of vertices are partitioned into 3 partitions {M1, M2, and M3}. Edge list contains the outgoing edges from each vertex as each edge has been assigned a unique id by a partitioner and that unique id will be used to route a message across network. Vertex A espied to be replicated on different nodes so change in vertex A must be communicated to other nodes. Synchronization can be achieved in this manner.

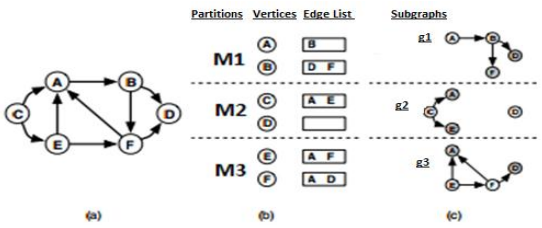

Fig. (1). Graph and Graph Partitioning [3]

Different graph partitioning solutions uses different partitioning techniques to partition a graph; For example, GraphLab partitions a graph using a vertex cut approach, while another graph partitioning solution ParMetis uses edge cut approach to partition a graph. Figure 2 illustrates the vertex cut and edge cut partitioning approaches.

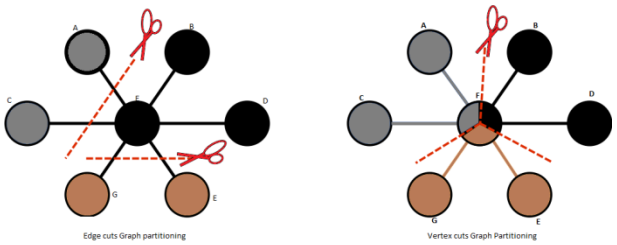

Fig. (2). Graph Partitioning Techniques [4]

Some graph partitioning libraries use coarsening strategies to minimize the graph and some use GAS (Gather, Apply and Scatter) strategy. To minimize communication requirements and balancing the memory load, distributed graph systems rely on the availability of a good graph partitioning. To meet the challenge of computation on power law graph, researchers introduce a new parallel graph abstraction that eliminates dependence on the degree of vertex program through direct exploitation of vertex-factor decomposition of GAS programs [5]. The GAS modal consists of three phases.

Gather: In Gather phase, adjacent vertices information is collected.

Apply: it update the value of the central vertex.

Scatter: it spreads the new value of the vertex on mirror nodes.

Gather function on each machine runs locally and then an accumulator is sent from every local machine or mirror node to the master node/copy. The apply function is executed on master machine and the updated value sent to every mirror machine. Finally, Scatter function runs parallel on every local machine. In vertex cut, partitioning vertex can be copied at multiple machines so gather & scatter functions will run in parallel on each node and the updated value or changed value is communicated across the network. Since each vertex is stored at multiple machines, the situation causes communication and storage overhead which can be improved by using different partitioning techniques and this paradigm is also a focus area of the research. A balanced p-way vertex cut approach can be used to achieve this objective by limiting the number of machines spanned by each vertex. If a vertex A is stored on 10 different machines then changes in vertex A must be replicated to all machines. Metis sequential graph partitioner uses Edge cut graph approach for graph partitioning. It coarsens the input graph until they become diminutive. It has three phases.

A. Coarsening

Coarsening is used to reduce the complexity of a graph and it is the first step in graph partitioning algorithms. The purpose of this step is to construct small graphs from the graph given as an input. Those small graphs preserve the connectivity information. For example, there is an edge (x, y) and a newly created node in coarsen graph which have weight of sum of the node x and y. When two (super) vertices are collapsed together, one of the (super) vertices is reused. In other words, one of the vertices is merged into the other. After a merger, the vertex that has been merged is not deleted. Its edges are deleted and it is declared inactive, but its vertex value is utilized to remember which vertex has been merged to it [3].

B. Initial Partitioning

In this step, a recursion bisection algorithm which repeatedly bisects a graph or sub graph is used for graph partitioning. Bisection algorithm uses breadth first traversal approach.

C. Refinement

In this phase, coarsens graph are projected back into the original graph. Coarsening is performed on original graph while refinement is performed on small coarsens graph.it improves the partitions quality. Partitioning completes after this step.

II. LITERATURE REVIEW

Rapidly growing social networks and other graph datasets require scalable processing infrastructure. Map Reduce; despite its popularity for big data computation, is limited at supporting iterative graph algorithms. As a result, a number of distributed, parallel and Sequential graph processing systems have been proposed including GraphLab, ParMetis and PT-SCOTCH [3]. Researchers have put an enormous amount of efforts into this area as the size of data is growing by two folds and it is not an easy task to manage the billions nodes of data. That's how graph partitioning is gaining its importance; therefore, to manage enormous amount of data, some partitioning techniques were needed. There are many approaches to cut a graph on cluster of machines p, A common approach is balanced p-way edge cut which assigns each vertex evenly to machines and number of edges connecting different nodes from different machines are minimized. Figure 3 illustrates examples of partitioning.

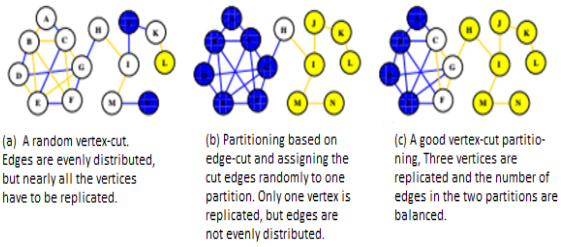

Fig. (3). Vertex Cut Strategies [5]

In random vertex cut, edges are randomly assigned to different nodes and nearly all vertices are replicated. This method has high replication factor. While in other method, edges are evenly assign to machines and it has very low

replication factor. Structure and data of a graph plays a central role in reducing a communication and ensures work balance [2]. How different machines or nodes interact when a graph is partitioned using vertex cuts is given in Figure 4.

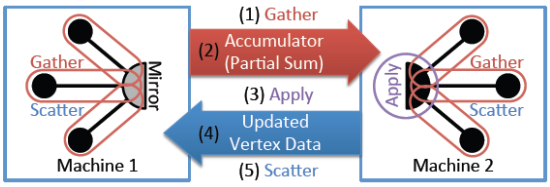

Fig. (4). GAS Composition [6]

A. Vertex cut strategies

There are three different strategies that can be used in vertex cuts graph partitioning.

1. Balanced p-way vertex-cut

Power graph allow single vertex to be stored on multiple machines. The storage overhead and communication across network can be improved. This can be achieved by evenly assigning edges to machines and each machine store only assigned edges. To achieve this objective, balanced p-way cut is used. PowerGraph also addresses the issues associated with the edge cut approach. The Power graph enables each vertex to be copied at multiple nodes. In vertex cuts, vertex program is divided along two machines with a single high degree point with gather and scatter functions running parallel to each other. It assembles the vertex data which is switched across the network grid. As the PowerGraph abstractions allow a single vertex program to be copied across numerous nodes, it can refine work balances and decrease cross network communication and storage load by equally giving edge data information to different nodes and allowing vertices to range to different nodes. Since each edge is not replicated on different nodes, it minimizes the cross network communication because the storage and network overhead require many machines spread on each vertex and any change on vertex must be replicated on all other machines. By restricting the number of vertices to copy on each vertex, storage and cross network communication overhead can be reduced. For every vertex in which replicas are multiplied, one specific node / machine is chosen as master copy and it enables the master version of the vertex data for read / writes operations. Then the left over copies of vertex are mirrored and maintains a local vertex data which is used for read only purpose. Partitioning a graph in three way vertex cut yields only two mirrors but only master data will change the data. After the changes are made on master copy they are immediately replicated to all mirrors. Edge cuts in power graphs are the major issue addressed in Vertex cuts. To break a graph instantly, high degree vertices must be cut into small fractions. The balance constraints assure that edges are evenly spread across the nodes and naturally improving work balance even in presence of vertices with a very high degree. Randomly assigning edges to machines is the easiest way to construct a vertex cut. Randomly placing edges which are fully parallel to data, almost perfect balances can be achieved on a larger graph and this can be applied to stream settings.

2. Greedy Vertex cuts

To improve vertex cuts random construction, the process of placing the edge should be redirected which minimizes the expected replication factor places next edges on the nodes [7]. When considering the placement task i + 1 edge after placing the leading edge i, a randomization is constructed. The targets are defined by using the conditional expectations [7]. The greedy algorithm does not work without the coordination of different nodes and there are two distributed implementations of it.

2A. Coordinated

In coordinated, greedy heuristic runs on every machine and update the distributed table continuously. To improve the performance, cashing is used which decrease the communication with the estimate of Ai (v).

2B. Oblivious

In oblivious, greedy heuristic runs independently on each node and each node/ machine maintains its own estimate of Ai and with no additional communication.

3. Balanced p-way Hybrid cut

Hybrid vertex-cut algorithm uses differentiated partitioning for low degree and high-degree vertices. Based on this approach, a new heuristic algorithm is derived called Ginger and is provided to further optimize partitioning for PowerLyra [7]. In hybrid cut, vertices evenly set edges to nodes and only copy vertices to construct a graph. The replication factor; the average copies for a vertex, is dependent upon the memory and network overhead. High degree in a regular graph brings upon a rupturing increase in vertex cuts already present. The overall goal of the existing vertex cuts is to decrease the number of vertices. Since the high degree vertices eventually need to be copied to the majority of the processors, the key is decreasing the low degree vertices. Unfortunately, the existing heuristic algorithm for vertex cuts is bias towards high degree vertex. Adopting differentiated partitioning to low and high degree vertices; an evenly balanced p- way hybrid cut is provided that prioritizes on decreasing the low degree vertices.

Majority edges independently belong to their assigned vertex and destination of an edge to stay away from multiple copies of the edges. Low degree vertices are adopted by hybrid cuts to equally give low cut with in edges to nodes by mixing up their target vertices. In a high degree vertices case, high cuts are adopted by hybrid cuts to spread all edges by mixing their source vertices. Later, hybrid cuts task is to make copies and build local graphs as regular vertex cuts. In hybrid cuts, all vertices along with their edges are given a low degree vertex. The partition created by hybrid cut only delivers four mirrors and collects good even loaded balances.

On skewed graphs, greater issues of hybrid cuts are given with edge and vertex cuts. Firstly, the replication factor given for hybrid cut is extremely low. In low degree vertices, all edges are gathered with their assigned vertex; therefore, no need to form replicas for them. In a high degree vertices case, the higher bound of increased copies is caused by adding a high degree vertex evenly to number of partitions like machines. Unlike the degree of the vertex which shuns an expanding replication factor in edge and vertex cuts. Secondly, hybrid cut gives unidirectional permission locally which when used by hybrid computation model to decrease communication cost in run time. Thirdly, hybrid cut is very accurate in graph ingress because it is hash based partitioning for both high and low degree vertices. Finally, partitions created by hybrid cuts are naturally balanced on vertices and edges. Random placing of low degree vertices leads to balance of vertices which are almost like balancing the edges for low edge vertices.

III. Graph Partitioning

The problem of a finding an optimal partition is NP-complete as a result many approximate solutions have been proposed [1]. Following are the list of efficient graph partitioning techniques**.** As the size and amount of collected data and computing power grows, it become very difficult for modern datasets to fit on one single node. Efficient, parallel and distributed algorithms are required for large scale data. That is where graph partitioning solutions came in and provides a solution to these problems.

A. GraphLab

GraphLab is an open source distributed computation framework developed in C++ and it is used to perform data mining tasks. GraphLab perform partitioning by cutting vertices instead of cutting edges. In vertex cut, a high degree vertex can be split across multiple machines. GraphLab differentiate between vertex data shared with adjacent vertices and edge data shared with all neighbors. GraphLab works with the undirected graph and does not differentiate between graph edges. GraphLab uses GAS modal for the partitioning of graphs and run an update function for a vertex on multiple machines in parallel instead of running an update function on single machine. It uses de-randomization strategy for partitioning instead of a randomly constructing a vertex cuts. In de-randomization strategy, vertex is equally assigned to machines. Purpose of this is to avoid the cross machines communication.

B. Graphlab versions

Three are three different graph lab versions.

1. *Shared-memory, multicore parallel GraphLab*

Let suppose a Graph G = (Vertices, Edges, and Data) where V represents Vertices (set of vertices), E represents Edges (set of edges) and D is the user-defined data and data can be associated with each vertex Dv: v in V or can be associated to each Edge (Du). Graph structure is static while data is mutable. It was a stateless procedure which updates data only within the scope of a vertex and schedules or execute the future updates on other functions. The drawback in this procedure is simultaneous execution of two update functions which can result in collision / race condition.

2. *Distributed GraphLab*

In distributed GraphLab, graph is partitioned into many sets and each set executes on separate machine of clusters. Delta cache consistency has been introduces which keeps the vertex consistent. If a lock on plot is required on an edge or vertex on the boundary which has to do in both partitions involved; therefore, locking becomes distributed locking. Grpahlab solve fault tolerance issue by check pointing.

3. *Power Graph*

PowerGraph combines the feature from GraphLab and Pregel [8]. The problem is many graphs have a power law connectivity distribution. There are popular vertices with many edges as well as unpopular vertices with fewer edges. Typically, a Zipf distribution of in / out degree. This causes huge problems for GraphLab as partitioning becomes highly imbalanced. Since some partitions have high degree of nodes while some has very few. The situation leads to imbalance computation. Power graph innovate few things. In first innovation, PowerGraph perform partitioning by vertex cuts instead of edge cuts which pick one partition for each edge and allow high degree vertices to be copied at multiple machines. In second invention, power graph runs update function in parallel instead of to run it on single vertex, this lead high degree vertex to be updated parallelized. In third invention, PowerGraph use balanced way partitioning instead of random partitioning which equivalence the load on each partitions. PowerGraph also introduced the concept of delta caching, and purpose of this concept is to only trigger those vertices on other machines where changes need to be communicated. It also maintains the result of gather phase which allow skipping the gather phase in subsequent iterations. List of active vertices is maintained by PowerGraph engine. Power engine set the order of active vertices and executed them according to that order. PowerGraph introduced a new concept known as Balanced p-way vertex-cut and Greedy vertex cuts approaches.

C. ParMetis

The most widely used graph partitioner is the sequential partitioner Metis from Karypis [4]. There are several parallel graph partitioners: ParMetis, PT-Scotch and etc. All of these implementations are explicitly parallel programs for distributed-memory architectures. ParMetis is a parallelization of Kmetis using MPI. In ParMetis, each process owns a random portion of the input graph [4]. ParMetis is a library that is used for partitioning of irregular large graphs using multilevel graph partitioning algorithms. Traditional graph partitioning algorithms perform a partitioning directly on original graph and are very slow and do not produced a good quality partitions. Multilevel algorithms work entirely in different way. They reduced the

size of the original graph by collapsing the vertices and edges. Multilevel algorithm coarsen the input graph, perform initial partitioning on small coarsen graph and then project these smaller graph into original graph in lase refinement phase. In this way, Metis produced high quality partitions for large graphs. Metis partition the graphs in the presence of a multiple balancing constraints. Constrains can be weight associated with the domain or associated with the storage. Metis provides a library which is written in C ++ in which different functions are exposed. Those functions can be used to perform a partitioning on input graph. Metis also provide the graph check class which take graph as an input and return whether the graph format is inappropriate or not. Format of the command is given below; write below on command line interface:

Graphch graphFile; where graph file is the input graph.

D. PT-SCOTCH

PT-Scotch is a library used for sequential and parallel graph partitioning. It also use the multilevel heuristics for graph partitioning. In multilevel graph, heuristics graph is first coarsening (divided into smaller parts) then performed the partitioning on that smaller parts. PT-Scotch applies the bisection algorithm iteratively. The key point that differentiates the PT-Scotch from ParMetis is ParMetis performs one level of coarsening despite the number of partitions desired whereas PT-Scotch performs [Log2n] cycles of coarsening and refinement. It only computes bisection on each step resulting in recursively repeating the same step. PT-Scotch use "divide and conquer" approach which recursively breaks a problem into more sub problems which can be of the same (related) or non-related type. It breaks the problems into small pieces until they become simple enough to solve. Then combine the result of the problems to yields a single solutions or result.

IV. EXPERIMENTAL SETUP

This Research work is based on experimental setup in which following step has been taken to achieve the research objective:

A. Dataset

For this research, live journal dataset has been used which is provided by SNAP (Stanford Network Analysis Project). It is a graph mining library and general purpose network analysis. It provides graph with millions of vertices and edges. A live journal graph was downloaded which consists of 4847571 nodes and 68993773 edges.

B. Tools / OS

Tools and Operating system that has been used for experimental work are;

Ubuntu is an open source OS and in this research work, it is required to run GraphLab. It is required for compiling GraphLab and installs the build-essential package.

Step 1:

Installation and configuration of GraphLab

Step 2:

For GraphLab, in first step, the large dataset was selected for experiment. Live general dataset was downloaded from Stanford Large Network Dataset Collection. Dataset statistics are as below:

Nodes: 4847571

Edges: 68993773

Step 3:

After confirming the dataset, the chunks of a large graph file were created and the size of each chunk is approximately 150 MB using following command.

Split -l 10000000 soc-LiveJournal1.txt

Where soc-LiveJournal1.txt is the graph file stored in directory dataset.

Step 4:

Pagerank and Partitioning algorithms were executed on dataset using the following commands on Ubuntu command line editor.

./pagerank --graph=./dataset/soc-LiveJournal1_t10m.txt --format="snap"

./partitioning --graph=./dataset/soc-LiveJournal1_t10m.txt --format="snap"

V. RESULTS

Following are the results obtained from executing Kmetis and Pmetis for different number of machines / partitions.

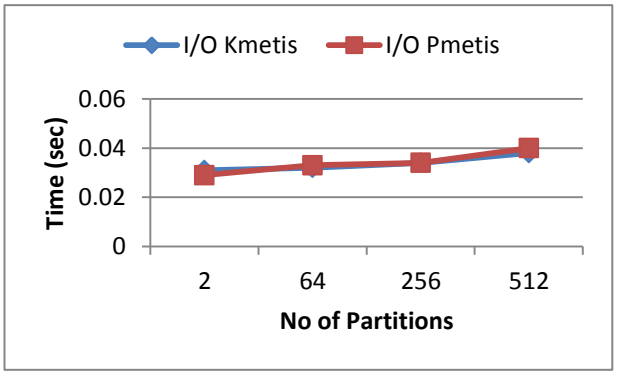

Fig. (5). I/O Comparison of Kmetis and Pmetis

Figure 5 represents the graphical comparison of I/O Kmetis and I/O Pmetis. There is not much difference at this level. Partitioning comparison between Kmetis and Pmetis is given below.

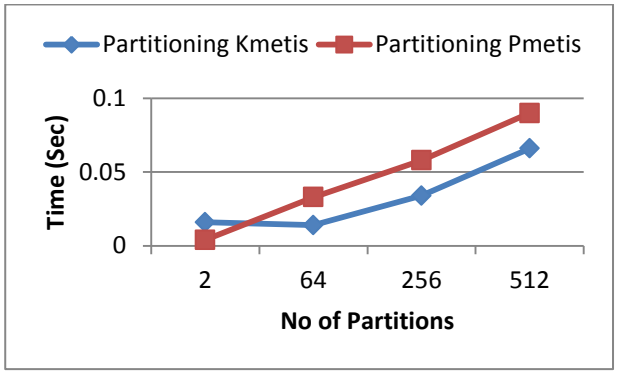

Fig. (6). Partitioning Comparison of Kmetis and Pmetis

Figure 6 represents the partitioning comparison of Kmetis and I/O Pmetis. It is evident from above Figure that Pmetis performs better when numbers of partitions are small while Kmetis performs better when numbers of partitions are greater than 16.

Table 1. Kmetis and Pmetis for 2 way partitioning

Kmetis for 2 partitions			
Graph details	# of vertices	# of Edges	partitions
	15606	45878	2
2way Partitioning	# on Edge cuts	Balance	
	141	1.03	
Timing information	I/O Time	Partitioning Time	Total Time
	0.031	0.016	0.047

Pmetis for 2 partitions			
Graph details	# of vertices	# of Edges	Partitions
	15606	45878	2
2 way Partitioning	# on Edge cuts	Balance	
	142	1	
Timing	I/O Time	Partitioning	Total
information		Time	Time
	0.029	0.004	0.035

Table 1 shows I/O and partitioning result for 2 partitions. A graph having vertices 15606 and Edges 45878 when partitioned using Kmetis and Pmetis produced above results and above results are showing that when numbers of partitions are two, Pmetis perform better than Kmetis. It can be seen that both I/O and Partitioned results for Pmetis taking less time than Kmetis.

Table 2. Kmetis and Pmetis for 64 way partitioning

Kmetis for 64 partitions			
Graph information	# of vertices	# of Edges	Partitions
	15606	45878	64
64 way Partitioning	# on Edge cuts	Balance	
	2840	1.03	
Timing information	I/O	Partitioning	Total
	0.032	0.014	0.048

Pmetis for 64 partitions			
Graph information	# of vertices	# of Edges	Partitions
	15606	45878	64
64 way Partitioning	# on Edge cuts	Balance	
	2919	1	
Timing information	I/O	Partitioning	Total
	0.033	0.033	0.069

Table 2 shows I/O and partitioning result for 64 partitions. A graph having vertices 15606 and Edges 45878, when partitioned using Kmetis and Pmetis produced above results and above results are showing that when numbers of partitions are 64, Kmetis outperform Pmetis. It can be seen that both I/O and Partitioned results for Kmetis taking lesser time than Pmetis.

Table 3. Kmetis and Pmetis for 512 partitioning

Kmetis for 512 partitions			
Graph details	# of vertices	# of Edges	Partitions
	15606	45878	512
512 way Partitioning	# on Edge cuts	Balance	
	9860	1.05	
Timing information	I/O Time	Partitioning Time	Total Time
	0.038	0.066	0.109

Pmetis for 512 partitions			
Graph details	# of vertices	# of Edges	Partitions
	15606	45878	512
512 way Partitioning	# on Edge cuts	Balance	
	6942	1.02	
Timing information	I/O Time	Partitioning Time	Total Time
	0.04	0.09	0.135

Table 3 shows I/O and partitioning result for 512 partitions. A graph having vertices 15606 and edges 45878 when partitioned using Kmetis and Pmetis produced above results and above results are showing that when numbers of partitions are 512, Kmetis outperform Pmetis. It is evident that both I/O and Partitioned results for Kmetis taking lesser time than Pmetis. After above experiments, it has been proved that when number of partitions are high, only practical approach is Kmetis, while for small number of partitions, Pmetis is better approach.

VI. CONCLUSION

It has been identified that PowerGraph produce better partitioning then ParMetis and PT-Scotch. PowerGraph uses the Vertex cut approach for graph partitioning while ParMetis and PT-Scotch uses the edge cuts approach. It also has been identified that ParMetis uses two different algorithms Kmetis and Pmetis. Both the algorithms were executed on same dataset and it has been observed that Kmetis outperformed Pmetis. But for small number of partitions (e.g. less than or equal to 10), Pmetis performance is better than Kmetis. For large number of partitions Kmetis is the practical approach and should be followed.

ParMetis and PT-Scotch use the same edge cut (Coarsening, initial partitioning and Refinement) strategy for graph partitioning. ParMetis and PT-Scotch both uses Recursive bisection algorithms. The key difference which has been identified between ParMetis and PT-Scotch is ParMetis applies Recursive bisection algorithm on coarsens graph only once while PT-Scotch apply this algorithms recursively on resultant coarsen graph due to which it perform slower for large graphs and ParMetis performs better than PT-Scotch for large number of graph.

VII. ACKNOWLEDGEMENT

I am very much thankful to ALLAH who bestowed upon me the courage and knowledge needed for this study.

I would like to express my deepest appreciation to all those who provided me the possibility to complete this research work. I want to appreciate Dr. Saif ur Rehman for the guidance. He helped me on every difficult track and provided sufficient feedback to complete this research.

REFERENCES

[1] L. Wang, Y. Xiao, B. Shao and H. Wang "How to partition a billion-node graph" In *Proceedings of 30th IEEE International Conference on Data Engineering (ICDE 2014)*, 2014.

[2] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson and C. Guestrin "PowerGraph: Distributed Graph-parallel Computation on Natural Graphs", In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, 2012, pp: 17-30.

[3] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda and J. McPherson "From "Think Like a Vertex" to "Think Like a Graph"", In *Proceedings of the VLDB Endowment (PVLDB)*, 2013, vol. 7, no. 3.

[4] N. Jain, G. Liao, and T. L. Willke "GraphBuilder: A Scalable Graph ETL Framework", In *Proceeding of First International Workshop on Graph Data Management Experiences and Systems (GRADES '13)*, 2013, Art. 4.

[5] Y. Low, "GraphLab: A Distributed Abstraction for Large Scale Machine Learning", Ph.D. dissertation, Car. Mel. Univ. PA, USA, 2013.

[6] Gupta "An Evaluation of Parallel Graph Partitioning and Ordering Softwares on a Massively Parallel Computer", IBM T. J. Watson Research Center, NY, Tech. Rep. RC 25008 (W1006-029), 2010.

[7] R. Chen, J. Shi, Y. Chen, H. Guan, B. Zang and H. Chen "PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs" Institute of Parallel and Distributed Systems, Tech. Rep. IPADSTR-2013-001, 2013.

[8] X. Sui, D. Nguyen, M. Burtscher, and K. Pingali "Parallel Graph Partitioning on Multicore Architectures", In *Proceedings of the 23rd international conference on Languages and Compilers for Parallel Computing (LCPC' 10)*, 2010, pp: 246-260.