An overview of recent graph partitioning algorithms

Chayma Sakouhi, Abir Khaldi, and Henda Ben Ghezala
ENSI, National School of Computer Sciences
Tunis, Tunisia

Abstract— *Distributed Graph Databases DGDB became increasingly important in many area , particularly in social networks. The DGDB systems require a good graph partitioner for an efficient distribution of graph data. The graph partitioner, aims to produce a balanced partitions while minimizing the communication cost between machines. Moreover, the graph partitioning method for graph databases should be distributed and also incremental to cope up with graph changes. There are several graph partitioning algorithms. In this work, we have classified them into three categories such as : static methods for offline partitioning, streaming method for dynamic graphs and OLTP methods for Online and Transactional DGDB. Then we have organized a comparative study between the different graph partitioning methods. We have concluded that, the partitioners used by such DGDB are not favorable, because according to our study, there are certain algorithms that outperform them.*

Keywords: Property graph, Graph Databases, Graph partitioning.

1. Introduction

Modeling a data as a graph have become important in many area. Graph has been used for performant data processing, data mining and for analytical purposes. It permits for an efficient querying of data and also to extract a new insights and help for making decisions. Graph become popular with social network analysis with the emergence of social media companies. In fact, several companies turn towards into graph modeling in order to cope with their needs. Examples would be Facebooks Open Graph, Twitters FlockDB, Googles Knowledge Graph, further, Google has its own graph computing system called Pregel, any many more.

Further, graph databases have been recently emerged. They are based on graph data model. This type of NoSQL graph databases overcomes the limitation of relational databases. Indeed, the benefits of using a graph data model in graph databases are numerous. It allows a natural modeling of graph data by introducing a query languages and operators for querying directly the graph structure, and also setting up a high level abstraction for storing graphs. Most of well known graph databases are available in their categories such as Neo4j [1], Orientdb [2],Trinity [3], Allegrograph[4], hypergraphdb[5], infinitegraph[6].

Graph databases are a powerful tool for graph-like queries. In fact it consists for serving continuous updates coming from clients simultaneously, and answering complex queries. Besides, processing, storing, querying such a large volume of data poses considerable challenges for their efficient processing. In addition, the graph size still growing exponentially over time, more particularly, in many applications and domain such as social networks. Hence, the large graphs require a distributed graph databases for efficiently managing a high concurrent queries and graph storage. Thus, distributed graph databases focus on distributing large graphs across a cluster of machines for parallel processing and querying of data. In reality, partitioning a dynamic and large graphs is non-trivial problem. Generally, the volume of graphs are massive, especially for property graphs which contains a rich metadata and the number of edges or relationships is too large. The graph partitioning algorithm should find an optimal distribution of data over the cluster. However, the partitioning of high quality must produce a balanced partitions (distributing equally the data between machines) and minimizing the number of edges cut (where its vertices belongs to two different partitions which increase the communication cost between machines). Therefore most of graph databases use random partitioning of data, while others use a sharding technique. However, few graph partitioning algorithms have been proposed to meet the challenge of distributed graph databases problems.

Throughout of this article, we will present an overview of different graph partitioning algorithms. We will classified them into three categories: (1) Static graph partitioning algorithms (2) Dynamic graph partitioning algorithms (3) OLTP graph partitioning algorithms. Then, we will introduce a comparative study between them based on a set of criteria. We restrict our evaluation to partitioning quality and system performance.

The paper is organized as follows: In section 2, we have explained current partitioning algorithms proposed for graph databases in detail. Section 3, presents the comparison of graph partitioning algorithms. Finally, in section 4, we draw conclusions.

2. Graph Partitioning Problems

The main goal of graph partition method is to divide a large graph into a set of disjoint partitions of equal size, while minimizing the communication cost between

partitions. Furthermore, dynamic graphs, highly required a dynamic / streaming partitioners, to cope up with graph changes and adapt dynamically their partitions instead of repeating the partitioning over again at each update.

Element	Definition
u, v	vertices $\in V$
n	$n = \|V\|$ number of vertices in graph G
$e(u, v)$	an edge $\in E$
m	$m = \|E\|$ number of edges in graph G
$N(u)$	set of Neighbors of vertex u

Table 1: Table of Notation

Graph partitioning consists to distribute graph vertices and edges into different partitions, stored on seperate servers. We consider a Graph G = (V, E) composed by a set of n vertices and m edges. Each vertex v, and edge e composed by a set of properties / attributes, values, id, labels etc. Each vertex v has a list of affiliated edges (also called by relationships) and a list of neighbors (affiliated vertices). In fact, there are two different technique to partition a graph.

Edge-cut consists to distribute equally the vertices V of a graph into a different servers / machines $V = \{P_1, P_2..P_k..P_K\}, P_k = \{v_{1,k}, v_{2,k}..v_{i,k}\}$ while allowing edges to span across multiple machines see figure 1. The edge-cut methods should produce a balanced partitions such that the size of each partition $\mid P_k \mid \leq \frac{n}{K}$, and minimize the number of cut edges which present a high communication cost between machines.

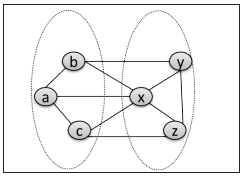

Fig. 1: Edge-cut method

Vertex-cut consists to distribute equally the edges of graph into into a different machines such that $E = \{P_1, P_2..P_k..P_K\}, P_k = \{e_{1,k}, e_{2,k}..e_{i,k}\}$. In fact, this method allows vertices to appear in more than one machine see figure 2. The partitioner which adopts vertex-cut should produces a balanced partitions such that the size of each partition $|P_k| \leq \frac{m}{K}$ and minimizes the communication cost by minimizing the number of replicated vertices.

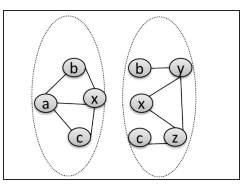

Fig. 2: Vertex-cut method

3. Partitioning Algorithms

We have distinguished several types of graph partitioning methods that we classified them into three categories. Some methods are designed for static and offline partitioning. The kind of this category does not support the graph changes over the time. Some algorithms are designed for dynamic graph partitioning. The methods of this category are enable to adapt their partitions with graph changes and support online partitioning due to streaming algorithms. The OLTP graph partitioning methods are the important category. They are designed for Online partitioning and for partitioning OLTP graph databases.

3.1 Static Graph partitioning algorithms

Static Graph partitioning algorithms are classical methods, are used since decades. These heuristic methods are created to deal with graph partitioning problems. Basically, they are designed for an efficient partitioning of static graph. We cite some of frequently used method, either by distributed systems or either by another algorithms as an initial partitioner.

Metis[7] is a well known partitioning algorithm in his category, build on MGP approach[8]. Despite, Metis is a static graph partitioning method, it has been proved, its efficiency in balanced graph partitioning. Basically, many recent works, such as Hermes [9], have been used Metis as initial partitioning algorithm. The basic idea behind the Metis is very simple. The large graph of million vertices and edges is coarsened down to a few hundred of vertices using a Maximum Matching, then, partition the small graph into k disjoint subsets. Finally, the partitions is projected back towards to the original graph. In order to improve the result of partitioning, a refinement algorithm is used to decrease the edge-cut. Hence Kernighan-Lin KL[10] algorithm implemented in Metis KL substantially decreases the edge-cut within a small number of iterations.

We present a distributed algorithm **Ja-Be-Ja**[11] designed to deal large scale distributed graphs, in oder to produce more balanced graph partitions and minimize edge cut. Indeed, the basic idea behind Ja-Be-Ja is very simple : The algorithm initializes randomly a partition for each vertex of the graph. Indeed, the system defines the energy function of each vertex and also the energy of the graph system. The vertex energy is defined as the number of neighboring vertices belongs to different partition, defined as: $\forall u \in V = \{u_1, u_2..u_n\}$

$$energy(u) = |N_u - N_u(u(p))| \quad (1)$$

Whereas, the energy of the system is the sum of the energy of the vertices. Further, it is defined as the number of edges between their incident vertices with different partition (equivalent to edge-cut), defined as:

$$energy(G, u) = \frac{1}{2} \sum_{u \in V} energy(u) \quad (2)$$

Then, the algorithm apply heuristic local search in oder to produce lower energy states (min-cut). However, the local search operator is executed by all vertices in parallel: each vertex attempts to change its partition to the most dominant partition among its neighbors. In order to preserve the size of the partitions, the vertices can not change their partition independently. Instead, they only swap their partition with one another. If the partition exchange results in a total lower energy of the system then the two vertices swap their partition. Otherwise, they keep it. Finally, The swap processes are periodically repeated by all the vertices in parallel, and when no more swaps take place in the graph, the algorithm is converged.

Ja-Be-Ja-vc [12] is a distributed partitioning algorithm. It is another extension of Ja-Be-Ja based on vertex-cut method. This algorithms adopts Ja-Be-Ja algorithm while the computation is based at the level of edges instead of vertices. In fact, the algorithm assigns randomly a set of edges to differents partition as an initial step while allowing vertices to be replicated into many partitions. Then it defines the energy function for each edge and for each partition. According to the energy of the system, which refers to the total energy of edges, the algorithm iteratively improves the initial partitioning, by applying a local search algorithm.

DFEP [13] , is a distributed algorithm. It is implemented on distributed system such as Hadoop and Spark. The algorithm is based on edge-partitioning to produce more balanced partitions. Instead of assigning vertices over instances, DFEP distributes the edges over partitions to avoids the edge cut problems. The algorithm based on the concept of buying the edges through an amount of funding where the computation is considered at the edges. It starts with a defined number of partition k and the program executed in parallel. At the initialization step, all partitions received a set of funding or units which is equals to $|E|/k$ while $|E|$ is the number of total edges. Then, each partition selects randomly a candidate vertex v to affect it, the amount of unit $M_i[v]$ from partition i, and all edges initialized as unassigned, Then, the program organized as a sequence of round where each iteration composed by three steps. The first step is executed at each vertex. In fact, for each partition i, if the vertex v has a positive amount of units $M_i[v] > 0$, the vertex will propagates the units equally between its outgoing edges that are free or owned by that partition i. Thus, for each eligible edge e, it received an amount of $M_i[v]/|eligible|$ units. The second step executed at each edge. In effect, the free edge e is bought by the partition that gives an attractive offer of funding, The win partition i loses one unit of funding on the edge, and the rest is divided equally between the vertices composing the edge $\forall v \in N(e)$. For each vertex of the edge receives an amount of $M_i[v]/2$. Otherwise, the loser partitions that participate for funding the edge e sends back the sum of units equally between the vertices S participated for the funding of the edge e at the partition i, $M_i[e]/|S|$. In the third step, each partition i receives an amount of funding according to its number of edges bought E_i as a matter of fact, to support the small partitions to buy more edges, and reach the balance between the partitions. The distribution of funding is proportional switch the size of partition, while the average funding is concluded using the following formula:

$$AVG = \sum_{i \in [1..k]} |E_i|/K \tag{3}$$

The funding function $funding = min(10, AVG/E_i)$ is distributed between the vertices which has a positive amount at the partition i. The program loops while all edges are bought and assigned

3.2 Streaming graph partitioning algorithms

Static graph partitioning algorithms are classical heuristic methods. They are basically, used by recent systems as an initial partitioner. The disadvantage of this category is that, not only they are didied for static graphs whose does not support the dynamic changes over time, but also they are offline graph partitioning algorithm. Otherwise, in this case, the dynamic graph must be repartitioned again to adopt changes. In this context, dynamic graphs require a streaming graph partitioning algorithm. However it is crucial to have a an efficient graph partitioning method for dynamic graphs.

Fennel [14] is a streaming graph partitioning framework. Fennel aims to efficiently partition a dynamic graph online. The objective function of fennel is straightforward. In one hand it consists of finding a balanced graph partition that minimizes the number of edges cut. In the other hand, optimize the partitioning by an heuristic assignment of vertices. However, Fennel adopts on-pass streaming algorithm. It consists of greedy assignment of vertices to partitions as follow: it assigns each new vertex to an optimal partition such that respect the objective function 4. Hence, the global objective function is based on a set of accounting costs. Thus, for each partition (vertex partition) it defines the inter-partition cost c_{out} and the intra-partition cost c_{in}. However, the inter-partition cost refers to the total number of edge-cut $|e(P_i, G \backslash P_i)|$. Whereas, the intra-partition cost refers to balance cost.

$$f(P) = c_{out}(|e(P_1, G \backslash P_1)|...|e(P_k, G \backslash P_k)|) + c_{in}(|P_i|..|P_k|) \tag{4}$$

Therefore, the partitioning algorithm is based on solving the optimization problem of $f(P)$ by minimizing the cost of total cut and balance cost between different partitions. Subsequently, the vertex is assigned to the optimal partition such that does not maximize the global objective function. As a result, the heuristic framework Fennel distributes on streaming the vertices over different partitions or clusters while maintaining all partitions balanced and minimizing the number of edges cut.

DynamicDFEP [15] is a distributed edge partitioning approach for large dynamic graphs. The algorithm is designed

to deal with the challenge of large scale of graph where graph changes its data over time. DynamicDFEP can be used to partition large graphs and also to update the partitioning when new edges and vertices are added or removed. In reality,a graph has already been partitioned, re-running a graph partitioning algorithm from scratch, just because a few vertices or edges have been added, is extremely inefficient. However, new vertices and edges should still be processed and assigned to the right partitions. The main idea behind DynamicDFEP is simple. It is used to overcome the re-partitioning problem. It is based on DFEP algorithm to partition a graph and it suggested three alternative methods to adopt graph changes. The partial partitioning is designed not to completely re-partitioning of the graph for every incoming vertex or edge, but just the partitioning is considered only for the subset of new dataset. The program initializes the subgraph and then executes a sequence of iterations to partition the sub datasets. This approach applies DFEP starting from the current partitioned graph. Since it starts with funding already distributed in the graph, it will need a very small number of iterations before all the new edges and vertices have been partitioned. Due to adopt the dynamic changes in graph, DynamicDFEP is designed to deal with dynamic updates. DynamicDFEP, introduces heuristic method based on rest of DFEP partitioning outputs. Because, the first partitioning algorithm keeps track of the amount of units that each partition has committed to each vertex and each edge. The UB-Ins method exploits the existing funding in the outgoing vertices of each new edge in order to assign it to the best partition. In fact, it assigns the edges e to the partition that gives a large amount of unit through the outgoing vertices v of edges.

3.3 OLTP graph partitioning algorithms

In recent years, few recent works moved towards distributed graph databases partitioning. These algorithms have been designed for distributed graph databases systems DGDB. The DGDB consist for serving complex queries managing OLTP operations, storing a large volume of data etc. It is a quite difficult find an optimal partitioner for DGDB that partition a graph while efficiently adapt with graph updates. We present the graph databases partitioning methods available in the literature.

The Incremental Online Graph Partitioning algorithm **IOGP** [17] is designed for distributed graph databases. The main goal of IOGP is to optimize the performance of such graph databases. However, the algorithm is enable for handling OLTP operations and partitioning data in one-pass streaming. For instance, if the graph performs an insertion queries of new vertices or edges, the algorithm places new arrived data in appropriate partitions. Further, one of IOGP capabilities, it could allow a physical migration of data in order to keep balance between partitions and minimizes the cut edges by spliting vertices with large number of affiliated edges. The partitioning method is organized into three stages. Initially, the algorithm defines a data structures to maintain the states of vertices in each partition and to optimize partitioning quality. These data are stored in memory for high latency, and can be restored from full scan of gra databases in case of failure. IOGP defines four edge counters for each vertex, $alo(v)$ and $ali(v)$ store the number of actual local outgoing/incoming edges of v while, $plo(v)/pli(v)$ store the number of potential local outgoing/incoming edges of v. The $loc(v)$ function returns server id, that actually store v, also each server maintains a size counter, which indicates the vertices and edges number. The first phase is described by quiet stage. Using deterministic hashing function, IOGP assigns the new vertex to destination server, and stores new edges with their incident vertices. Consequently, each new edge $e(u, v)$ is inserted twice, one for outgoing edge list of u and for incoming edge list of v. During this stage, the algorithm updates edge counters (alo, ali, plo, pli) while serving vertex and edge insertions. For instance, once a new edge $e(u, v)$ is inserted a sequence of updated are maintained. However, after adding $e(u, v)$ to list of outgoing edges of u, the algorithm verify the location of v $loc(v)$. If the vertex u and the destination vertex v are stored in the same server, then it increases $alo(u)$ and $ali(v)$ by 1, otherwise, it increases $pli(v)$. Likewise, on the server that stores the destination vertex (s_v), counters are updated accordingly. In order to maintain a balanced partitions, the algorithm introduced a vertex reassigning stage. The principle is simple: moving the vertex to the destination partition that stores most of its neighbors. IOGP employs Fennel heuristic score, to determine the best partition with high score to hold a vertex v. For this reason, IOGP exploits the counters parameter to efficiently compute the best location. The score function calculated from equation 5, will indicate whenever moving a vertex like v from its server to another will increase or decrease score.

$$\begin{aligned} score_{s_i} = & max\{2 * (plo(u)_{s_i} + pli(u)_{s_i}) \\ & - 2 * (alo(u)_{s_u} + ali(u)_{s_u}) \\ & + [size_{s_i} - size_{s_u}]\} \end{aligned} \tag{5}$$

Migrating vertices is non-trivial procedure. However, a set of updates are performed in-memory during the vertex reassignment. In fact, the two partitions maintain, in the same time, the data structure related to vertex migration like u. In the original server s_u, all actual local outgoing/incoming edges $alo(u)$ and $ali(u)$ turn into potential local outgoing/incoming edges $plo(u)$ and $pli(u)$, contrary in target server which turn all potential edges of u into local. Basically, all vertices connected with vertex u,also maintained an updates. For instance, in the original server, all us incoming/outgoing edges are reduced by 1, because their vertex u is migrated, while, it increased by 1 in the target server. IOGP dives into splitting edge step when the number of edges for vertex u exceeds MAX-EDGES. During the

splitting stage, IOGP distributes an outgoing edge together with its target vertex and places an incoming edge together with source vertex. Once a vertex u enters in splitting stage, it will never moved again.

In this context, we can cite another distributed graph databases, it's Neo4j. Essentially, Neo4j used a technique called cache sharding which consists of routing each request to databases instance. Furthermore, every instance in the cluster contains a full copy of the graph. Hence, **Hermes** [9] was designed as an extension of the open source Neo4j. Thus, the main goal of Hermes is straightforward: Hermes apply a multi level partitioning algorithm such as Metis to partition the graph over multiple servers as initial step. However, Metis was designed for static offline partitioning. Therefore, Hermes has been introduced the lightweight repartitioner to maintain high quality partitions and adapts any changes in the graph (updates, insertion etc.). Furthermore, the lightweight repartitioner algorithm aims to incrementally improve an existing partitioning by decreasing edge-cuts while maintaining almost balanced partitions.

The lightweight repartitioner algorithm implemented in Hermes system is performed as follow: After initially partition the graph using metis, Hermes incrementally maintains lightweight repartitioner. The algorithm aggregates the auxiliary data from each partition. The auxiliary data collects the total weight of each partition and its relevant vertices such as vertex weight and the number of its neighbors belongs to different partitions. Any updates in the graph generates a set of updates of auxiliary data too. Lightweight repartitioner is composed by two phases. The first phase is splitted into two stages. The first stage is organized as a sequence of iterations. In each iteration the algorithm employs the auxiliary data to select the eligible vertices for logical migration and the target destinations. However, The algorithm selects the candidate vertex to choose the target partition for migration. In fact, migrating the vertex v from partition source P_s to partition target P_t if P_s will not turn into underloaded less than $2-\alpha$ times the average partition weight. Thus, the P_t will not turn into overloaded more than α maximum allowed imbalance. In addition, the migration is accepted when P_s is overloaded and there is a maximum gain of moving the vertex v to the target partition P_t which indicates the number of edge-cut decreased. As a result, the algorithm returns for each vertex the target partition with the gain. In the second stage, the selected vertices are logically migrated. In fact, it updates the auxiliary data related to both partitions (source partition and target partition) and the auxiliary data related to the moved vertices and their neighbors. The next phase, performs a physical migration of data as a final step of lightweight repartitioner algorithm. Indeed, the set of vertices selected for migration, from the previous stage, are received by their target partitions. Behind the scene and between the two steps, the algorithm maintains a local copy of market vertices and edges, to avoid the lose of data via network. Once the moved vertices and edges are received by their final destination, all replicated data are removed.

Some graph databases systems adopts a distributed architecture for storing and managing large scale graph data in parallel. Trinity, Titan, OrinetDB etc are examples for some distributed graph databases system. By default, Trinity, Titan, employ a random partitioning strategy that randomly assigns vertices to a set of machines. to avoid complex graph partitioning method. In addition, Orientdb system supports a sharding per class for data storage. For example, it creates for each class a cluster of machines to stores graph data. In fact, random partitioning and sharding have a limitations such as unbalanced distribution of data and high communication via networks.

4. Discussion

In this section, we consider a comparison study between the graph partitioning algorithms presented in the previous sections. Our study is based on a set of metrics to evaluate (1) the partitioning quality of an algorithm and (2) performance evaluation.

4.1 Discussion partitioning quality

The main goal of any graph partitioning algorithm is to divide large graph into disjoint partitions to optimize computation runtime. Therefore, high quality graph partitioning depends on certain criteria, should be considered during the partitioning, described as follow:

- **Graph type:** we distinguish several types of graph partitioning algorithm. The static methods, they are classical method designed for static graph. Once the graph is modified, the algorithm require to repartition the entiere graph. Dyanmic partitioning methods, are designed for streaming processing of dynamic graph. OLTP partitioning methods, they are designed for OLTP graph databases. Graph databases require a dynamic graph partitioning algorithm to optimize the OLTP operations online.
- **Balance:** One of the most common creteria in graph partitioning problem is to distribute the data equally over the cluster of machines. Due to the large volume of data, processing a data into single machine afflicted the performance quality of processing large scale graphs.However, partitions size need to be balanced to exploit the speedup of parallel processing/computing over different machines. Furthermore, distribute data storage over a cluster of machines is highly recommanded, to balance the work load between machines. Therefore, the partitioning technique shoulds produces a balanced partitions to avoid the underloaded and overloaded partitions.
- **Communication cost:** Distributing the data over different partitions, produces a disconnected data. However,

there are 2-ways of partitioning techniques. That we can group them into two categories. The first category of algorithms employs a vertex partitioning, they are initially dividing data for computation by assigning vertices to partitions while allowing edges to span multiple machines. For consequence, the partitioning method requires to reduce the number of edge cut. Whereas, the second category of algorithms employs edge partitioning. Radher than the vertex partitittioning, these kind of methods distribute the edges over several partitions as an initial step. It is possible that vertices could appear into different partitions. For these reason, the algorithms require to reduce the number of replicated vertices (vertex cut). As a result, when the number of cut (edge / vetex) is large, the communication cost between partitions increased also.

- **Streaming:** Another major challenge in large-scale graph partitioning problem is an efficient processing of dynamic graph. Due to the rapidly growth of data, graphs evolved dynamically over the time. Especially, social networks are large graphs that performs frequent updates each second. Thus, a good partitioner should be incremental,at a given moment, it adopts its partitions according to graph changes to avoid the repartitioning of entirety graph.

4.2 Discussion performance evaluation

The quality of partitiong method can be measured by another criteria related to algorithm performance. Further, we added the following metrics:

- **Stability:** In order to minimize the communication cost as minimum as possible, and keeping balance between partitions, some graph partitioning algorithms perform a data migration between partitions. Some algorithms adopt a physical migration of data via networks between machines. Hence, it could decrease system performance, losing data caused by network collision etc. In fact it introduces a significant cost.
- **Scalability:** The algorithm should be distributed. However, the algorithm executed independently on each server, where each part of the graph is processed seperatly and in parallel over a computational system.
- **In-memory** Graph processing is performed in memory. It increases system performance, speedup of runtime processing.
- **Convergence:** The algorithm converges in less number of iterations.
- **Complexity:** Complex graph partitioning algorithms impact system performance.
- **Runtime:** The algorithm completes its partitioning in less runtime.

4.3 Comparison

We discuss the comparison between the different graph partitioning algorithms, based on performance criteria discussed in above section.

As mentioned in tables 2, all graph partitioning algorithms produce a balanced partitions. Almost of these algorithms, reduce the communication cost between partitions. However, according to experimental studies affected by different works, we note there are significant variation between the different algorithm. For instance, according to [17], in the context of edge-cut ratio, IOGP is better than Fennel and Metis. In addition, in [9] the porcentages of edge-cut for Hermes Lightweight repartitioner and Metis is slightly comparable. In fact, Metis requires large amount of memory for runtime execution. Indeed, the amount of memory required by Metis scales according to number of edges and coarsen/uncoarsen stages. Further, The number of migrating edges and vertices in Metis are high comparing to Hermes, which actually, impact system performance.

Ja-Be-Ja produces a balanced partitions but with communication cost 10 times higher than DFEP, as declared in [15]. In addition, DFEP requires less number of iteration to converge rather than Ja-Be-Ja as shown in table 2. Ja-Be-Ja used a swapping technique to fetch for a best partner to swap their partitions. In terms of performance, this operation is very expensive, because each vertex scans its neighbors list to find a swapping partner, otherwise it scans all vertices of graph. In contrast, in [12], Ja-Be-Ja-vc has a better result than DFEP and Ja-Be-Ja. It produces a vertex-cut less than DFEP while their partition size are nearly close. Furthermore, Ja-Be-Ja-vc outperformes its old version based on edge-cut method. It produces a highly balanced partitions and min-cut than Ja-Be-Ja.

As illustrated in table 2 and in [14], we note that Fennel outperforms Metis capabilities. In effect, for the twitter graph contains more that 1.4 billion of edges, Fennel partition the graph in 40 minutes while acheiving a balanced partitions and generated 6,8 % edges cut. Whereas, Metis took 4 hours to produce balanced partitions with 11,98 % of edges cut.

From the point of view of performance, Metis is the worst algorithm compared to others. In addition, the static algorithm DFEP, the streaming framework Fennel, and the DynamicDFEP, are not designed for online partitoning such as OLTP graph databases. These distributed algorithms maintain a high quality partitioning, inasmuch as, balance, min communication cost, converge in less number of iteration and requires less runtime. By nature, these algorithm does not perform a physiscal migration of data, instead of Metis, JaBeJA, IOGP, Hermes. The common important features of DFEP, Fennel, DynamicDFEP is to avoid the data migration during graph partitioning or over graph updates. In particular, IOGP and Hermes, those recently systems are designed for Distributed Graph Databases. They produce a high quality graph partitioning relative to performance criteria. Despite

Algorithm	Graph	Balance	Min-cut	Streaming	Stability	Scalability	Convergence	Runtime	Complexity
Metis	Static	yes	yes	no	no	no	no	no	no
Ja-Be-Ja	Static	yes	yes	no	no	yes	no	no	no
Ja-Be-Ja-vc	Static	yes	yes	no	no	yes	no	no	no
DFEP	Static	yes	yes	no	yes	yes	yes	yes	yes
Fennel	Streaming	yes	yes	yes	yes	yes	yes	yes	yes
DynamicDFEP	Dynamic	yes	yes	yes	yes	yes	yes	yes	yes
Hermes	OLTP	yes	yes	yes	no	yes	no	yes	no
IOGP	OLTP	yes	yes	yes	no	yes	no	yes	no

Table 2: Comparison between graph partitioning algorithms

the fact that, these system employ a data migration during computation in order to maintain balanced distribution of data and minimize the number of replicated data to reduce the communication costs. Consequently, they can impact system performance, especially, data is transmitted from one partition to another over network. In fact, it depends on network bandwidth.

5. Conclusion

In this study, we presented three generation of graph partitioning methods. The first is a classical methods which designed for static graph partitioning. The second category has been proposed for dynamic graph partitioning problems. The third category has been introduced for distributed graph databases problems. We have also described their implementation details. In addition, we have presented a comparative study between the different graph partitioning algorithms based on performance criteria. According to our evaluation, there are a slightly comparable difference between all graph partitioning algorithms. However, IOGP and Hermes Lightweight repartitioner algorithms outperform the other algorithms. They produce a high quality graph partition despite, they employ a physical migration of data. We believe that Hermes, IOGP and distributed graph databases partitioning algorithm could improve the partitioning quality, while replacing the physical migration of data by one-pass streaming of data. For instance, the partitioning algorithm provides an heuristic method, to decide the optimal partition to store it instead of using data migration during OLTP transactions and executed queries.

References

[1] [Online]. Available: https://neo4j.com/
[2] [Online]. Available: http://orientdb.com/orientdb/
[3] B. Shao, H. Wang, and Y. Li, "Trinity: A distributed graph engine on a memory cloud," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 505–516.
[4] [Online]. Available: https://allegrograph.com/
[5] B. Iordanov, "Hypergraphdb: a generalized graph database," in *International conference on web-age information management*. Springer, 2010, pp. 25–36.
[6] [Online]. Available: http://www.objectivity.com/products/infinitegraph/
[7] G. Karypis and V. Kumar, "Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0," 1995.
[8] B. Hendrickson and R. W. Leland, "A multi-level algorithm for partitioning graphs." *SC*, vol. 95, no. 28, 1995.
[9] D. Nicoara, S. Kamali, K. Daudjee, and L. Chen, "Hermes: Dynamic partitioning for distributed social network graph databases." in *EDBT*, 2015, pp. 25–36.
[10] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.
[11] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, "Ja-be-ja: A distributed algorithm for balanced graph partitioning," in *Self-Adaptive and Self-Organizing Systems (SASO), 2013 IEEE 7th International Conference on*. IEEE, 2013, pp. 51–60.
[12] F. Rahimian, A. H. Payberah, S. Girdzijauskas, and S. Haridi, "Distributed vertex-cut partitioning," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2014, pp. 186–200.
[13] A. Guerrieri and A. Montresor, "Dfep: Distributed funding-based edge partitioning," in *European Conference on Parallel Processing*. Springer, 2015, pp. 346–358.
[14] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "Fennel: Streaming graph partitioning for massive scale graphs," in *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM, 2014, pp. 333–342.
[15] C. Sakouhi, S. Aridhi, A. Guerrieri, S. Sassi, and A. Montresor, "Dynamicdfep: A distributed edge partitioning approach for large dynamic graphs," in *Proceedings of the 20th International Database Engineering & Applications Symposium*. ACM, 2016, pp. 142–147.
[16] C. Battaglino, P. Pienta, and R. Vuduc, "Grasp: distributed streaming graph partitioning," in *1st High Performance Graph Mining workshop, Sydney, 10 August 2015*, 2015.
[17] D. Dai, W. Zhang, and Y. Chen, "Iogp: An incremental online graph partitioning algorithm for distributed graph databases," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2017, pp. 219–230.