A Survey of Streaming Graph Partitioning Algorithms

I. Introduction

In broad range of domains from online social networks to protein-protein interactions, data can naturally be modelled and represented as graphs. There are numerous works in literature to efficiently store and process such graph data. Scale of real-world graphs has already exceeded capabilities of single-node systems, which has lead to emergence of distributed graph processing systems in last decade such as Pregel [1], PowerGraph [2] and GPS [3]. Partitioning of graph into cluster of machines plays a crucial role in such systems, affecting both design and performance of the system. Execution of user programs and node-to-node communications are largely dependent on partitioning model of the system, influencing the computation model of the system as in case of PowerGraph [2]. From performance point of view, partitioning has an effect on overall network communication and computation workload of each machine. Minimizing the network communication and balancing computation workload across cluster is crucial to performance in any distributed system in general and unfortunately conflict each other. Researchers have been studying graph partitioning problem for many decades in different domains such as scientific computing, network analysis and high-performance computing. Recently, series of *Streaming Graph Partitioning (SGP)* papers have been published to overcome the limitations of existing work in big-data era [4]–[8]. Such systems treat graph as a stream of elements and make partitioning decisions on the fly without accessing the entire graph. Main advantage of such systems over traditional methods is their capability to scale very large graphs with limited resources.

Although significant number of streaming graph partitioning techniques have been proposed in recent years, there is a lack of understanding on how they perform in real-world applications, let alone systematic analysis and comparison of these techniques. Performance analysis in these papers are usually conducted in a limited context and are not representative performance of real-world applications. To clarify, these techniques are mostly deployed in either generalized large-scale data processing frameworks [4] or ad hoc implementations [5], which do not represent the performance characteristics of specialized distributed graph processing systems. Furthermore, most studies report only structural metrics and runtime statistics for PageRank, which can be seen as oversimplification considering the variety of real-world graph workloads. In this paper, we aim to provide overview of the work that has been done in streaming graph partitioning techniques in recent years. There is a considerable amount of work in this area and we do not aim the cover all work. Instead we aim to classify existing work based on their commonalities and differences, refer to some prominent examples from literature and provide systematic comparison of main approaches.

Objective of traditional k-way balanced graph partition is to create balanced vertex disjoint sets while minimizing the number of edge-cuts, i.e. number of edges of which endpoints lie in different partitions. NP-Hard nature of the k-way graph partitioning problem [9] caused researchers to work on traditional heuristics methods, with prominent example being multi-level methods [10], [11]. These methods usually rely-on structural metrics such as fraction of edge-cuts or number of vertices in each partition. Although such metrics might be representative in case of homogeneous cluster and uniform workload, they are naive and limited for real-world applications. Consider a online social network with a scale-free degree distribution; performance of the system rely on graph topology and edge/vertex access patterns rather than plain number of vertices per partition. Scale of the modern real-world graphs is another obstacle for these traditional heuristics. These methods rely on complete knowledge of the graph which usually does not fit into memory of a single-node and causes disk spills. Although there are some efforts on parallel/distributed graph partitioning algorithms such as ParMetis [12], these methods are usually complex and time-consuming. Streaming graph partitioning, on the other hand; performs a serial pass over the graph, maintains state information about the processed part of the stream and makes placement decisions on the fly. Hash partitioning, which simply creates a balanced disjoint partitioning based on a hash key, is a well-known, commonly adopted streaming partitioning policy due to its simplicity.

In 2012, Stanton and Kliot [4] formulated the streaming graph partitioning model for graph loading scenarios. They proposed ten different streaming heuristics and provided experimental performance analysis. Since then, many other works have adopted the streaming graph partitioning model and proposed different heuristics for different scenarios. All these methods have similar performance characteristics compared to traditional heuristics; (i) they maintain much smaller state in the memory which enables them to scale to very large graphs with limited resources, (ii) one-pass nature of streaming algorithms significantly reduces the partitioning time, (iii) they can provide significant improvements over hash partitioning, even comparable to traditional offline heuristics for some scenarios. In this paper, we study existing algorithms in two main dimensions. First, we analyze how graph is distributed across the cluster, in other words cut model. Edge-cut methods create vertex-disjoint subsets of the original graph where endpoints of an edge might be in different partitions whereas vertex-cut methods create edge-disjoint partitioning where some vertices may span multiple machines. Second, we classify streaming algorithms based on the input model i.e. vertex-streams and edge-streams. Former is a more informative but restrictive model which requires graph to be known in advance while latter is more preferable in terms of dynamic, growing graphs. In addition to these two dimensions, we analyze common characteristics of existing algorithms e.g. how communication cost is modelled or ideal graph ordering. More detailed taxonomy along with the summary of our findings can be found in Table II.

Rest of the paper is organized as follows. In next section, we provide an high level overview of distributed graph processing systems and graph partitioning problem in particular. Detailed analysis of edge-cut and vertex-cut methods is given in Section 3 and Section 4 respectively. Section 5 discusses how graph partitioning objective can be generalized to account for various application requirements with examples from literature.

II. Background

A. Distributed Graph Processing Systems

Bulk Synchronous Processing (BSP) [19] is a parallel programming model where computation is modelled as sequence of steps. In BSP Model, workers perform local computation at each step and synchronize through message passing after each step. MapReduce [20], the most prominent large-scale data processing framework based on BSP, has been used for batch processing of web-scale graphs. However, stateless nature of MapReduce model is ill-suited for iterative graph applications which resulted in specific graph processing frameworks. Pregel [1] is first large-scale BSP implementation with a graph specific API. Pregel employs vertex-centric programming model, where user defined compute function is iteratively executed over the vertices of graph and communication is performed through edges of the graph. Giraph [21] is an open-source implementation of Google's Pregel. GPS [3] employs Pregel's vertex-centric model and utilizes vertex migration for dynamic load balancing. GraphLab [22] employs an asynchronous communication model which can expedite convergence for various machine learning tasks. PowerGraph [2] and PowerLyra [18] factor vertex computation over edges to efficiently process natural graphs with power-law degree distribution. Interested reader is referred to [23] for a survey of distributed graph processing systems and to [24] for experimental comparison of Pregel-like vertex-centric systems.

TABLE I

CHARACTERISTICS OF STREAMING GRAPH PARTITIONING ALGORITHMS

	Algorithm	**Streaming Model**	**Comm. Cost**	**Parallelization**	**Graph Updates**	**Method**
Edge-cut	LDG [4]	Vertex	Cut Size	Inter-Stream Comm.	No	Greedy
	FENNEL [5]	Vertex	Cut Size	Inter-Stream Comm.	No	Greedy
	Restreaming LDG [6]	Vertex	Cut Size	Intra-Stream Comm.	Yes	Greedy
	Re-FENNEL [6]	Vertex	Cut Size	Intra-Stream Comm.	No	Greedy
	LOOM [13]	Vertex	Inter-Partition Traversal	Yes	No	Greedy
	BMI [14]	Vertex	Runtime	No	No	Greedy
	Leopard [15]	Edge	Cut Size	No	Yes	Dynamic
Vertex-cut	DBH [16]	Edge	Replication Factor	Yes	Yes	Hash
	Grid [8]	Edge	Repliation Factor	Yes	Yes	Constrained
	PowerGraph [2]	Edge	Replication Factor	Inter-Stream Comm.	Yes	Greedy
	HDRF [17]	Edge	Replication Factor	Inter-Stream Comm.	Yes	Greedy
Hybrid-cut	Ginger [18]	Edge	Replication Factor	Inter-Stream Comm.	No	Greedy

B. Balanced Graph Partitioning

Let $G(V, E)$ be a simple graph where V represents the set of vertices and E represents set of edges. Let $P = \{P_1, ..., P_k\}$ be a partitioning of the original graph G, $C(P)$ represent the total communication cost of partitioning and $w(P_i)$ represent the workload of a single partition P_i . Traditional (k, β) balanced graph partitioning problem can be generalized as;

$$\text{minimize } C(P) \text{ subject to: } w(P_i) \leq \beta * \frac{\sum_{j=1}^{k} w(P_j)}{k}, \forall i \in \{1...k\} \tag{1}$$

More precisely, objective is to generate k disjoint but complete partitions of the graph such that overall communication cost is minimized while each partition has approximately similar workload controlled by slackness parameter β . $\beta = 1$ represents the case where exact balance is required whereas $\beta > 1$ relaxes balanced constraint on partition loads.

This problem has been proven to be NP-Hard [9] and researchers have proposed heuristics methods to achieve an approximate solution. Those heuristics aims to achieve balanced vertex-disjoint partitions with minimal edge-cuts i.e. $w(P_i)$ is the number of vertices per partitions and $C(P)$ is the overall number of edges of which endpoints lie in different partitions. Among others, multi-level partitioning scheme have been successful and commonly adopted due to its quality partitions in reasonable time. METIS [10] and its family of partitioning softwares have been the most prominent example of such multi-level scheme and consists of three main phases. First, size of input graph is successively decreased during coarsening. Then smaller graph is partitioned and finally resulting partitioning is uncoarsen back to the original graph. METIS and similar heuristics require complete knowledge of the graph and this offline nature prevents them to scale large graphs. In addition, these methods are designed to work on static graphs and require re-partitioning of the complete graph in case of updates, which makes them unpractical for dynamic graphs. Irregular structure of the modern graphs arising from web or social networks impose new challenges for these traditional methods [25].

Several studies have been published addressing the aforementioned shortcomings of traditional heuristics. ParMetis [12] is an example of parallel methods that can process graphs larger than a memory of a single machine. However, they still require global knowledge of the graph and expensive inter-machine communication. Re-partitioning algorithms such as Hermes [26] focus on dynamic refinement of initial partitioning instead of re-partitioning whole graph in case updates. [25] studies how existing multi-level methods can be refined to partition natural graphs with power-law degree distribution.

C. Streaming Graph Partitioning

Streaming graph partitioning has originally been proposed by Stanton and Kliot [4] as a lightweight, scalable graph loader program which makes placement decisions while reading the input graph serially from the disk. Although streaming partitioners are usually inferior to offline methods in terms of partitioning quality, they can scale to much larger graphs with limited memory and significantly reduce partitioning time. These advantages enabled streaming graph partitioning approaches to gain popularity in both academia and industry, and series of papers have been published. Formal definition of streaming graph partitioning model is as follows.

Let $S = \{e_1, e_2, ...\}$ be an ordering of of the graph $G(V, E)$ (note that a vertex stream can easily be transformed into a locality preserving edge stream by combining adjacency list of all vertices), $G^t(V^t, E^t)$

represent the processed subgraph at time t, $P = P_1, ..., P_k$ be the k partitions, P^t represent the partitioning of a graph $G^t(V^t, E^t)$ at time t and $h(e_t, P^t, P_i^t)$ be an algorithmic specific objective function that accounts for objectives of (k, β) balanced graph partitioning as in (1). Then, streaming graph partitioning can be generalized as;

$$\forall e_t \in S \text{ in order, assign } e_t \text{ to the partition } P_i \text{ where } P_i = \arg \max_{i \in \{1...k\}} \big(h(e_t, P^t, P_i^t)\big) \tag{2}$$

More precisely, objective is to assign each graph element e to partition P_i that will maximize the partitioning objective function $h(e_t, P^t, P_i^t)$ using partial information P^t and $G^t(V^t, E^t)$. Hash partitioning, which is a streaming graph partitioner in its simplest form, completely ignores current state of partitioning P^t and places each element e_i to a partition using a hash function.

III. Edge-cut Streaming Graph Partitioning

In this section we consider edge-cut streaming graph partitioning methods where original graph is fragmented into vertex-disjoint partitions. When endpoints of an edge lie in two different partitions, it has be replicated in both machines and constitutes an edge-cut. Similar to traditional balanced graph partitioning, most edge-cut SGP methods formulates communication cost in fraction of edge-cuts and partition load in number of vertices per partition. Formally;

$$C(P) = \frac{\sum_{i=1}^{k} |e(P_i, V - P_i)|}{|E|} \text{ and } w(P_i) = |P_i| \tag{3}$$

A. *Edge-cut SGP on Vertex Streams*

Adjacency list is a common format to represent graphs and adopted by many distributed graph processing systems such as Pregel [1] and GPS [3]. Each vertex v is stored with a list of neighbouring vertices. This representation is natural fit for graph loading scenarios where input graph is serially read from storage and each vertex is assigned to a partition on-the-fly. Therefore, many edge-cut SGP methods assume vertex stream input model.

Although streaming graph partitioning problem have been first defined in [4], hash based graph partitioning is considered as one of the simplest form of SGP and has been widely used in many systems. It distributes vertices across cluster by hashing vertex keys and resorts to random cut if a pseudo-random hash function is used. Hash partitioning achieves well balanced distribution however it completely ignores the graph topology or history of previous assignments. Therefore, hash partitioning completely ignores locality and is known to perform poorly in terms of edge-cut ratio. Given uniform random assignment of vertices to k cluster, expected edge-cut ratio is $(1 - \frac{1}{k})$.

Stanton and Kliot [4] formulate the traditional (k, β) balanced graph partitioning problem in streaming model and propose various heuristics for vertex assignment, e.g. range-based, hash-based and greedy methods. They report that Linear Deterministic Greedy (LDG) is the best overall across all datasets. LDG assigns vertex v to partition P_i with most number of neighbours while penalizing for partition size. Formally;

$$\arg\max_{i \in \{1...k\}} \left((|P_i^t \cap N(v)|) * (1 - \frac{|P_i^t|}{C}) \right) \tag{4}$$

where $C = \beta * \frac{|V|}{k}$ refers to the maximum capacity for a partition and $N(v)$ refers to set of vertices v neighbours. Multiplicative weights of LDG strictly enforces balance constraint.

Tsourakakis et al [5] formulates the same problem as a maximization problem and greedily assign incoming vertex v to a partition P_i that increases the objective function most. FENNEL relaxes the hard cardinality constraint in (1) by introducing a term in $C(P)$ which takes its minimum value when all partition sizes are equal, i.e. $|P_i| = \frac{|V|}{k}$, $\forall i \in \{1...k\}$. Using $c(x) = \alpha * x^{\gamma}$ for intra-partition cost, FENNEL selects the partition index for incoming vertex v by maximizing following objective function;

$$\arg\max_{i \in \{1...k\}} \left((|P_i^t \cap N(v)|) - (\alpha * \gamma * |P_i^t|^{\gamma - 1}) \right) \tag{5}$$

Parameter γ controls how much imbalance is tolerated. In an extreme case where $\gamma = 1$, partition balance is completely ignored and vertex is assigned to partition with most number of neighbours. On the other hand, $\gamma = 2$ resorts to the case where vertex is assigned to partition with least number of non-neighbours, a greedy heuristic originally proposed in [27].

Both LDG and FENNEL significantly improve the edge-cut ratio compared to hash partitioning and provide comparable performance to METIS, using much less resources. Experiments on Twitter Graph with 1.4 billion edges show that these streaming methods are approximately ten times faster than their offline counterpart, METIS [5]. However, parallelization remains as an issue. Both methods require each parallel worker to continuously communicate their share of partitioning. On the other hand, hash based methods do not require any synchronization and can be parallelized without communication overhead.

These greedy heuristics utilize history of previous assignments, which leads to poor decisions initially and more informed decisions towards the end of stream. Based on this observation, Nishimura and Ugander [6] propose the re-streaming model where same or similar graph is streamed recurrently and claim that re-streaming can close the gap between traditional offline heuristics and single-pass streaming methods. Re-streaming versions of LDG (re-LDG) and FENNEL (re-FENNEL) have access to partitioning results of previous streams and record the most recent assignments P_i^t over previous stream P_i^{t-1}, which leads to more informed decisions even in the beginning of the stream. In case of FENNEL, re-streaming model can be utilized to achieve exact balance. By increasing α in (5) with each iteration, re-FENNEL initially emphasis on high quality partitions and improves balance with each re-stream.

Nishimura and Ugander [6] experimentally show that re-streaming techniques can achieve quality partitioning comparable to METIS while using only $O(n)$ memory as single pass streaming algorithms. More importantly, re-streaming model enables parallelization without inter-stream communication. Parallel workers only need to communicate their portion of partitioning results between each re-stream, as opposed to continuous synchronization required for single pass methods.

These solutions assume stream of vertices with adjacency lists, which makes them suitable for graph loading scenarios. For systems which support adjacency list input format, edge-cut SGP techniques is a lightweight, scalable alternative to random-cut with much better partitioning quality. Indeed, these methods can even achieve METIS-compatible quality partitions with re-streaming. On the other hand, vertex streams with adjacency lists require complete vertex information to be present in advance, making such methods inappropriate for growing, dynamic graphs and other input formats like triple list serialziations of RDF graphs.

B. *Edge-cut SGP on Edge Streams*

Since dynamic graphs can naturally be represented as stream of edges, edge-cut SGP methods can be extended for edge streams to handle growing graphs. Although not commonly adopted, there exists some work on edge-cut SGP on edge-streams.

Filippidou and Kotidis study streaming graph partitioning for dynamic graphs and propose an adaptive technique based on graph stream summarization [28]. Initially, a new vertex u of incoming edge (u, v) is assigned to same partition as its adjacent vertex v if v has been placed previously or both assigned to the partition with minimum weight if they appear in stream for the first time. Formally, vertex u of edge (u, v) assigned to partition P_i;

$$i = \begin{cases} \arg\min_{i \in \{1...k\}} (|P_i^t|) & \text{if } u \text{ and } v \text{ appear for the first time} \\ i \mid i \in \{1...k\} \text{ and } v \in P_i^t & \text{if } v \text{ appeared earlier in the stream} \end{cases} \tag{6}$$

Meanwhile, a summary structure called Condensed Spanning Tree (CST) is maintained with each incoming edge (u, v) and is utilized to compute on-demand partitioning of the processed part of the stream. Unlike re-streaming methods, on-demand partitioning can re-partition the graph using the CST without processing the whole stream.

Huang and Abadi take a similar approach to deal with dynamic graphs and combine dynamic re-partitioning with replication [15]. Leopard initially makes use of modified version of FENNEL over edge stream and continuously revisits the initial placement decisions for migration. When endpoints of a new edge (u, v) lie in different partitions, they become candidates for re-assignment. Specifically, Leopard analyzes whether u should be migrated to v's partition or vice versa. In addition, Leopard integrates vertex replication into its re-partitioning scheme for fault tolerance and improved edge-cut ratio. However, this optimization is only applicable for read-only applications that do not update vertex values. Otherwise additional communication is necessary to keep all replicas in sync, which is called as replication factor and discussed more detailed in vertex-cut methods.

Although edge stream model is a better fit for dynamic graphs, it is less informative compared to vertex streams with adjacency lists. Therefore, greedy edge-cut SGP heuristics initially produce partitioning of lower quality compared to its vertex stream counterparts and need to revisit initial assignments. On the other hand, edge stream model is successfully adopted by many vertex-cut SGP methods as discussed in Section IV.

In general, edge-cut methods are preferable in applications where input graph is present in advance, especially for the vertex-stream input model. Partitioning algorithm is able to make more informed decisions on vertex placement when complete vertex information is available. Since edge-cut partitioners focus evenly distributing vertices of the graph, these methods can produce high quality partitioning for graphs with regular structure, i.e. relatively uniform degree distribution. However, natural graphs with power-law degree distribution pose significant challenges specifically because of high-degree vertices [2], [25]. Vertex-cut methods are able to achieve better quality partitioning for such graphs, as discussed in next section.

IV. Vertex-cut Streaming Graph Partitioning

An alternative to traditional vertex-disjoint graph partitioning is to distribute edges across the cluster and produce edge-disjoint partitioning of the original graph, called vertex-cut graph partitioning. In this section, we consider vertex-cut streaming graph partitioning algorithms where adjacency list of a vertex might be distributed to multiple machines. Such vertices needs to be replicated in multiple machines and number of machines spanned is called replication factor. Similar to edge-cut methods, most existing vertex-cut SGP algorithms consider only structural metrics and formulate communication cost in replication factor and partition load in number of edges. Let $A(v) \subseteq \{1...k\}$ represent the set of partitions in which vertex v is replicated and $e(P_i) \subseteq E$ represent set of edges which are assigned to partition P_i. Then communication cost and partition load of vertex-cut partitioning P is formally defined as;

$$C(P) = \frac{\sum_{v \in V} |A(v)|}{|V|} \text{ and } w(P_i) = |e(P_i)| \tag{7}$$

A. Vertex-cut SGP on Vertex Streams

Although a vertex stream with adjacency lists is more informative model than an edge stream, it is more restrictive in terms of applicability as discussed in previous sections. Edge stream model is more appropriate for dynamic, growing graphs, as well as RDF graphs where data is represented as list of triples. In addition, each adjacency list in a vertex stream can be expanded to an edge list which corresponds to a locality preserving edge stream when combined. Therefore most existing work on vertex-cut SGP methods assume edge-stream input model with few exceptions. [29] discusses an hybrid input model where vertices arrive with some of its edges. [30] constructs vertex-cut partitioning over a degree-ordered vertex stream. However, it is not classified as a streaming method since it requires entire stream to be consumed before partitioning can be produced.

One system that can be considered under this section is PowerLyra and its hybrid streaming graph partitioning method Ginger [18]. PowerLyra Ginger differentiates between high-degree and low-degree vertices and employs edge-cut for low-degree vertices whereas in-edges of high-degree vertices are partitioned via vertex-cut. Hybrid-cut streaming partitioning in Ginger performs two pass over the vertex stream. Ginger employs FENNEL like heuristics to minimize replication factor of low-degree vertices in the first phase by assigning vertex v and its in-edges to partition P_i that minimizes the expected replication factor. Unlike FENNEL, Ginger incorporates both number of vertices and edges into its objective function. Formally,

$$\arg \max_{i \in \{1 \ldots k\}} \left((|P_i^t \cap N(v)|) - c(\ \frac{1}{2}(|P_i^t| + \frac{|V|}{|E|} * |e(P_i^t)|) \ ) \right) \tag{8}$$

where $c(x) = \alpha * x^\gamma$ is the balance term in FENNEL's objective function (5). High-degree vertices whose in-degree is higher than user supplied threshold are identified in the first phase. Then Ginger re-assigns in-edges of those vertices by hashing on source vertex. In this way Ginger preserves the locality of low-degree vertices while distributing high-degree vertices across the cluster. However, the additional re-assignment phase makes Ginger inapplicable for applications with growing dynamic graphs.

B. Vertex-cut SGP on Edge Streams

The simplest solution for vertex-cut partitioning is to distribute edges across cluster by using a hash function on some attributes of endpoints, e.g. concatenation of vertex ids. Although hash partitioning produces partitions with exact balance, it resembles random partitioning where each vertex is possibly replicated across many partitions. Considering that main challenge of power-law graphs are those few vertices with very high degree, one can focus on replicating only those high degree vertices. Degree Based Hashing (DBH) is built on this idea and assigns an edge to a partition by hashing the vertex with smaller degree [16]. Formally, given an hash function $h(v)$ which hashes vertex v to a partition $i \in \{1...k\}$, edge (u, v) is assigned to partition i;

$$i = \begin{cases} h(u) & \text{if } d(u) < d(v) \\ h(v) & \text{otherwise} \end{cases} \tag{9}$$

DBH distributes adjacency list of high degree vertices around the cluster whereas those of low-degree vertices are more likely to be placed together. Such a scheme reduces the expected replication factor as skew increases, however it relies on degree information to be present beforehand. Another approach to reduce replication factor is to put an upper bound on number of replicas per vertex. Jain et. al. employ constrained sets and virtually organizes the partitions on a 2D grid [8]. All partitions that are in same row or column with partition P_i constitute P_i's constrained set. When a new edge (u, v) is consumed, both endpoints are hashed to some partitions P_i and P_j. For any two vertices (u, v), it is guaranteed that constrained sets of P_i and P_j have at least two intersected partitions and one with the smallest size is selected for edge (u, v). Such a scheme upper-bounds replication factor with $2 * \sqrt{n} - 1$. Using two dimensional torus topology further reduces upper-bound to $1.5 * \sqrt{n} + 1$ [8].

Gonzalez et. al. introduces a simple greedy heuristic for vertex-cut SGP which has been adopted in PowerGraph [2]. Using the previous assignment history, PowerGraph utilizes following simple set of rules for edge (u, v); (i) if both u and v appear for the first time then edge is assigned to partition with smallest size, (ii) if both have appeared before and $A(u)$ and $A(v)$ intersect, then edge is assigned to partition with smallest size in intersection, (iii) if A(u) and A(v) do not intersect (one might be empty meaning that vertex appears for the first time), then edge is assigned to partition with smallest size in union of two sets. An equivalent formulation of such rules is to maximize following objective function for each arriving edge (u, v):

$$\arg\max_{i \in \{1...k\}} \big((|V(P_i^t) \cap \{u, v\}|) + (1 - \frac{|e(P_i^t)|}{C})\big) \tag{10}$$

Petroni et. al. exploit the skew in degree distribution for real world graphs and propose degree-aware greedy heuristic for vertex-cut SGP, called HDRF [17]. Similar to DBH, HDRF motivates replicating those few number of high degree vertices and aims to preserve locality of low-degree vertices. HDRF improves upon PowerGraph heuristic by incorporating degree information into the objective function. Since obtaining vertex degrees in from an edge stream requires pre-processing, HDRF accumulates partial degree information in a table and normalizes partial vertex degrees to $\theta(u)$ and $\theta(v)$ with each incoming edge (u, v). Resulting objective function is;

$$\arg\max_{i\in\{1...k\}} \big(g(v, P_i) + g(u, P_i) + \lambda * (1 - \frac{|e(P_i^t)|}{C})\big) \text{ where } g(v, P_i) = (1+(1-\theta(v))) * \mathbb{1}_{A(v)}(P_i) \quad (11)$$

By introducing $g(v, P_i)$ in its objective function, HDRF favours the partitions of lower-degree vertex over those of high-degree vertex. Moreover, HDRF introduces the term λ which accounts for the importance of imbalance in score computation. In original PowerGraph objective function in (10), balance term is always smaller than one and only used to break ties. Such a scheme might result in placing all edges into a single partition in case of an ordered stream. On the other hand, HDRF with $\lambda > 1$ avoids such problem by gradually increasing the importance of balance in objective function.

In case of dynamic graphs where partition capacity C is not known in advance, greedy objective functions can be modified for balanced partitioning at any given time. Given that min and max represent the current minimum and maximum partition sizes, substituting balance term with $\frac{max\ -\ |e(P_i^t)|}{max\ -\ min}$ promotes immediate balance.

Both hash based and constrained methods are embarrassingly parallel meaning that they can be parallelized without any communication or dependencies. On the other hand, greedy methods need some state to be shared among workers, similar to greedy edge-cut SGP heuristics. In particular, a distributed table with values of $A(v),\ \forall v \in V$ needs to be maintained.

Vertex-cut methods can achieve more balanced workload distribution by evenly distributing edges across cluster, especially in real world graphs with power-law degree distribution. By distributing the computation of high degree vertices across multiple workers, vertex-cut methods strive to avoid straggler vertices which might appear in edge-cut model. Furthermore, [7] and [2] show that number of replicas in a vertex-cut partitioning is lower than edge-cut partitioning along the same partition boundaries meaning that vertex-cut can lower the communication cost. Combined with edge-stream input model, these methods are favourable for modern applications with dynamic, scale-free graphs. On the other hand, vertex-cut methods rely on system support to allow vertex computation to span multiple machines. Although few recent systems such as PowerGraph [2], PowerLyra [18] and GraphX [31] provide such support, it complicates the systems design. Therefore, most existing Pregel-like distributed graph processing systems rely on traditional edge-cut model which is considered more intuitive for ”Think like a Vertex” paradigm.

V. Streaming Graph Partitioning with Generalized Cost Models

In previous sections we provided classification of existing literature on streaming graph partitioning based on cut and input model and described prominent examples for each quadrant of this streaming graph partitioning taxonomy. One common characteristics of all these methods is that they focus on optimizing structural metrics such as number of edges lie in partition boundaries (edge-cut methods) and replication factor (vertex-cut methods). Optimizing such measures might not necessarily minimize network communication nor achieve balanced workload distribution if graph is not accessed in uniform manner or cluster does not have homogeneous structure. In this section, we focus on how partitioning objective function can be modified to account for different cost models, workload and cluster characteristics.

Nishimura and Ugander [6] study more generalized types of balance for edge-cut SGP methods. In particular, they argue that re-streaming versions of LDG (4) and FENNEL (5) can be modified to account for more generalized notion of balance such as number of edges or any positive weights. By substituting $|P_i^t|$ with $x_i^t = \sum_{v \in P_i^t} a_v$ in (4) and (5), these methods can guarantee balanced partitioning on any positive vertex attribute a_v. In particular, balancing number of edges promotes balanced computation across cluster in applications where complexity of vertex computation is proportional to degree [4].

Bourse et. al. studies the performance characteristics of edge-cut and vertex-cut partitioning with and without message aggregation [7]. Most distributed graph processing systems support aggregation of messages between two partitions P_i and P_j in order to reduce network communication. It is shown that expected communication cost of vertex-cut partitioning is lower than that of edge-cut partitioning in case of message aggregation. [7] proposes a heuristic method for vertex-cut SGP with message aggregation which tries to optimize communication volume rather than replication factor.

[13] proposes a workload-aware greedy heuristics for edge-cut SGP problem by modelling communication cost in terms of probability of inter-partition traversal for a given subgraph matching workload. LOOM employs frequent subgraph mining techniques to discover patterns in a given workload and identifies matches over a sliding window. Using a greedy heuristics based on LDG in (4), LOOM assigns these matches to partitions which will reduce the possibility of inter-partition traversals for given subgraph pattern matching workload.

Methods we discussed so far assume a homogeneous cluster where each machine has identical capabilities. LeBeane et. al. [32] challenge this assumption and propose to adapt existing vertex-cut SGP heuristic to account for cluster heterogeneity. CPU power or memory capacity of a node is used to define notion of skew factor which determines the amount of data each partition is assigned to. [32] presents modified version of Grid [8], PowerGraph [2] and Ginger [18] vertex-cut SGP methods where amount of data in each machine is proportional to skew factor.

Xu et. al. address the cluster heterogeneity for edge-cut SGP methods in more sophisticated and studied way and propose several greedy heuristics which strive to minimize total execution time [14]. Proposed framework profiles the heterogeneous environment to estimate computing and communication capabilities of each node and requires users to provide workload specific cost function for average computation and communication per vertex. Based on these parameters $cost^t_{comp}(P_i)$ and $cost^t_{comm}(P_i)$ are computed, which represent the computation and communication time estimations of partition P_i at time t. Among others, the best performing heuristic Balanced Min-Increased (BMI) assigns vertex v to a partition P_i with minimum additional overhead while penalizing for current total cost. Formally;

$$\arg\min_{i\in\{1...k\}} \left(\Delta cost^t_{comp}(P_i) + \Delta cost^t_{comm}(P_i) + (cost^t_{comp}(P_i) + cost^t_{comm}(P_i))^\lambda\right) \tag{12}$$

VI. Experimental Results

In this section we conduct a comprehensive experimental study of streaming graph partitioning strategies and analyze the performance characteristics of different partitioning policies on distributed graph databases. In this paper, we are particularly interested in run-time performance of different streaming graph partitioning policies on real-world interactive workloads. Section VI-A describes our experimental setup whereas rest of the section presents our findings.

A. Experimental Methodology

1) Cluster Setup: We conducted our experiments on a local cluster connected with a 10Gbit/s Ethernet. Each machine has 32GB RAM and 12 2.1 GHz physical cores. We varied number of machines from 2 to 32 for scalability experiments. Default number of 4 machines is used for all other experiments unless otherwise stated.

We have set on using a distributed graph database rather than distributed graph processing engine in order to run real-world interactive workloads. Our choice of distributed graph database is TitanDB[1].

2) Datasets and Workloads: LDBC Social Network Benchmark [33].

LDBC SNB Interactive workload consists of three types of queries; (i) complex read-only traversal that touch significant portion of the graph, (ii) short traversal , (iii) update stream which concurrently inserts new vertices and edges. Benchmark specification defines a query mix which is representative of

[1]http://titan.thinkaurelius.com/

real social network workload. In addition to standard query mix, we use following to better analyze the effect of partitioning strategy on query performance;

- Point lookups where a single vertex or edge is retrieved based on a predicate
- Neighbourhood queries where 1-hop or 2-hop neighbourhood is retrieved for a given start vertex.
- Single-source, single-target shortest path queries
- BFS Traversals
- Read-only LDBC SNB Interactive Mix
- Read-write LDBC SNB Interactive Mix

3) Metrics: Primary performance metrics used in our experiments can be summarized in three main categories;

- **Partitioning Quality** Structural metrics that define the quality of partitioning result. Edge-cut ratio defines the fraction of edges which lie in partition boundaries for edge-cut methods whereas replication factor defines the average number of mirror vertices per vertex for vertex-cut methods. Standard deviation of vertex/edge count per partition is used to
- **Algorithm Performance** Performance of the streaming graph partitioning algorithm during data ingress. Ingress time is measured as time it takes to load a graph into the graph database. In addition, we measure the resource utilization of the partitioning algorithm itself, i.e. memory usage.
- **Run-time Performance** Effect of the partitioning method on performance of distributed graph database. In particular, we are interested in throughput and latency of the system as well as resource utilization.

VII. Conclusions

This paper focused on providing high level view of streaming graph partitioning approaches for distributed graph processing systems. We analyzed existing approaches in two main directions; (i) cut model describes how input graph is distributed across cluster and (ii) stream model describes how input graph is consumed by partitioning algorithm. In addition we discussed how streaming graph partitioning can be generalized to consider runtime aspects of distributed graph processing such as workload characteristics or cluster structure.

In summary, streaming graph partitioning algorithms are scalable and efficient alternative to traditional offline graph partitioning methods. One pass nature of streaming approaches enables them to scale much larger graphs with limited resources. Furthermore, streaming methods can naturally handle growing, dynamic graphs without re-partitioning the entire graph. However, most of the existing approaches ignores run time characteristics of real-world graph processing systems. In future, we plan to conduct a systematic experimentation on existing methods in real-world deployments. In particular, we plan to better understand run-time performance of existing algorithms on distributed graph processing systems under real-world workloads.

REFERENCES

[1] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2010, pp. 135–146.

[2] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Proc. 10th USENIX Symp. on Operating System Design and Implementation*, 2012, pp. 17–30.

[3] S. Salihoglu and J. Widom, "Gps: A graph processing system," in *Proc. 25th Int. Conf. on Scientific and Statistical Database Management*, 2013, pp. 22:1–22:12.

[4] I. Stanton and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proc. 18th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2012, pp. 1222–1230.

[5] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "Fennel: Streaming graph partitioning for massive scale graphs," in *Proc. 7th ACM Int. Conf. Web Search and Data Mining*. ACM, 2014, pp. 333–342.

[6] J. Nishimura and J. Ugander, "Restreaming graph partitioning: Simple versatile algorithms for advanced balancing," in *Proc. 19th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2013, pp. 1106–1114.

[7] F. Bourse, M. Lelarge, and M. Vojnovic, "Balanced graph edge partition," in *Proc. 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2014, pp. 1456–1465.

[8] N. Jain, G. Liao, and T. L. Willke, "Graphbuilder: Scalable graph etl framework," in *Proc. 1st Int. Workshop on Graph Data Management Experiences and Systems*, 2013, pp. 4:1–4:6.

[9] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory of Computing Systems*, vol. 39, no. 6, pp. 929–939, 2006.

[10] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. on Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[11] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," in *Proc. 9th Annual Int. Conf. on Supercomputing*, ser. Supercomputing '95. New York, NY, USA: ACM, 1995. [Online]. Available: http://doi.acm.org/10.1145/224170.224228

[12] G. Karypis and V. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *J. Parall. and Distrib. Comput.*, vol. 48, no. 1, pp. 71–95, 1998.

[13] H. Firth and P. Missier, "Workload-aware streaming graph partitioning," in *Proc. 5th Int. Workshop on Querying Graph Structured Data*, 2016.

[14] N. Xu, B. Cui, L. Chen, Z. Huang, and Y. Shao, "Heterogeneous environment aware streaming graph partitioning," *IEEE Trans. Knowl. and Data Eng.*, vol. 27, no. 6, pp. 1560–1572, 2015.

[15] J. Huang and D. J. Abadi, "Leopard: lightweight edge-oriented partitioning and replication for dynamic graphs," *Proc. VLDB Endowment*, vol. 9, no. 7, pp. 540–551, 2016.

[16] C. Xie, L. Yan, W.-J. Li, and Z. Zhang, "Distributed power-law graph computing: Theoretical and empirical analysis," in *Advances in Neural Information Proc. Systems 27, Proc. 28th Annual Conf. on Neural Information Proc. Systems*, 2014, pp. 1673–1681.

[17] F. Petroni, L. Querzoni, K. Daudjee, S. Kamali, and G. Iacoboni, “Hdrf: stream-based partitioning for power-law graphs,” in *Proc. 24th ACM Int. Conf. on Information and Knowledge Management*, 2015, pp. 243–252.

[18] R. Chen, J. Shi, Y. Chen, and H. Chen, “Powerlyra: Differentiated graph computation and partitioning on skewed graphs,” in *Proc. 10th ACM SIGOPS/EuroSys European Conf. on Comp. Syst.*, 2015, p. 1.

[19] L. G. Valiant, “A bridging model for parallel computation,” *Commun. ACM*, vol. 33, no. 8, pp. 103–111, Aug. 1990.

[20] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in *Proc. 6th USENIX Symp. on Operating System Design and Implementation*, 2004, pp. 137–149.

[21] (2016, Oct.). [Online]. Available: http://giraph.apache.org/

[22] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, “Distributed graphlab: A framework for machine learning in the cloud,” *Proc. VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[23] R. R. McCune, T. Weninger, and G. Madey, “Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing,” *ACM Computing Surveys*, vol. 48, no. 2, p. 25, 2015.

[24] M. Han, K. Daudjee, K. Ammar, M. T. Özsu, X. Wang, and T. Jin, “An experimental comparison of Pregel-like graph processing systems,” *Proc. VLDB Endowment*, vol. 7, no. 12, pp. 1047–1058, 2014.

[25] A. Abou-Rjeili and G. Karypis, “Multilevel algorithms for partitioning power-law graphs,” in *Proc. 20th Int. Parallel & Distributed Processing Symp.* IEEE, 2006, pp. 10–pp.

[26] D. Nicoara, S. Kamali, K. Daudjee, and L. Chen, “Hermes: Dynamic partitioning for distributed social network graph databases.” in *Proc. 18th Int. Conf. on Extending Database Technology*, 2015, pp. 25–36.

[27] V. Prabhakaran, M. Wu, X. Weng, F. McSherry, L. Zhou, and M. Haradasan, “Managing large graphs on multi-cores with graph awareness,” in *Proc. USENIX 2012 Annual Technical Conf.*, 2012, pp. 41–52.

[28] I. Filippidou and Y. Kotidis, “Online and on-demand partitioning of streaming graphs,” in *2015 IEEE Int. Conf. on Big Data*. IEEE, 2015, pp. 4–13.

[29] C. Xie, W.-J. Li, and Z. Zhang, “S-PowerGraph: Streaming Graph Partitioning for Natural Graphs by Vertex-Cut,” *ArXiv e-prints*, Nov. 2015.

[30] D. Margo and M. Seltzer, “A scalable distributed graph partitioner,” *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1478–1489, 2015.

[31] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, “GraphX: graph processing in a distributed dataflow framework,” in *Proc. 11th USENIX Symp. on Operating System Design and Implementation*, 2014, pp. 599–613.

[32] M. LeBeane, S. Song, R. Panda, J. H. Ryoo, and L. K. John, “Data partitioning strategies for graph workloads on heterogeneous clusters,” in *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 56.

[33] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz, “The ldbc social network benchmark: interactive workload,” in *Proc. ACM SIGMOD Int. Conf. on Management of Data*. ACM, 2015, pp. 619–630.