Clustering-based Partitioning for Large Web Graphs

*Deyu Kong, †Xike Xie and ‡Zhuoxu Zhang
*†‡University of Science and Technology of China
{*cavegf,‡zzx371479326}@mail.ustc.edu.cn, †xkxie@ustc.edu.cn

***Abstract*—Graph partitioning plays a vital role in distributed large-scale web graph analytics, such as pagerank and label propagation. The quality and scalability of partitioning strategy have a strong impact on such communication- and computation-intensive applications, since it drives the communication cost and the workload balance among distributed computing nodes. Recently, the streaming model shows promise in optimizing graph partitioning. However, existing streaming partitioning strategies either lack of adequate quality or fall short in scaling with a large number of partitions.**

In this work, we explore the property of web graph clustering and propose a novel restreaming algorithm for vertex-cut partitioning. We investigate a series of techniques, which are pipelined as three steps, streaming clustering, cluster partitioning, and partition transformation. More, these techniques can be adapted to a parallel mechanism for further acceleration of partitioning. Experiments on real datasets and real systems show that our algorithm outperforms state-of-the-art vertex-cut partitioning methods in large-scale web graph processing. Surprisingly, the runtime cost of our method can be an order of magnitude lower than that of one-pass streaming partitioning algorithms, when the number of partitions is large.

***Index Terms*—Web Graphs, Streaming Partitioning**

I. Introduction

The scale of graphs grows with an unprecedented rapid pace, including web graphs, social graphs, biological networks, and so on. Big graphs are often measured in terabytes or petabytes, with billions or trillions of nodes and edges. To cope with the big graph challenge, many distributed graph system are developed, such as [1], PowerGraph [2], GraphX [3], GraphLab [4], and PowerLyra [5]. In these systems, a big graph is partitioned into a predefined number of subgraphs, which are stored in distributed nodes. Each node of the distributed graph system operates on its subgraph in parallel, and different nodes are communicated and synchronized with message-passing. Therefore, the quality, efficiency, and scalability of graph partitioning algorithms are found to be imperative ingredients for bulk synchronous iterative processing in distributed systems. Because it affects the workload balancing and communication overheads, and thus has a direct effect on on large-scale graph system performance.

There are two mainstream graph partitioning strategies, *edge-cut* [6]–[11] and *vertex-cut* [12]–[15] partitioning, both of which are to optimize objectives of load-balancing and min-cut (for either edges or vertices), so that the overall performance of distributed graph systems can be improved. The vertex-cut partitioning strategy evenly assigns graph edges to distributed machines in order to minimize the number of times that vertices are cut. Theoretically and empirically, vertex-cut partitioning is proved to be significantly more effective than its counterpart for web graph processing [2], [16], because most real graphs follow power law distributions [17].

Despite many works done, the problem of effective graph partitioning on practical distributed graph system is still open.

The problem of graph partitioning has been widely studied in the past decade. For vertex-cut partitioning, there are two categories, a) *offline distributed algorithms* that load the complete graph into memory [9], [13], [15], and b) *online streaming algorithms* that ingest edges as streams and perform on-the-fly partitioning based on partial knowledge of the graph [2], [12], [14], [18], [19]. Offline algorithms do not scale well for distributed graph systems, with the tremendous increase of data volumes. For example, METIS [9] requires more than 8.5 hours to partition a graph with about 1.5 billion edges to only 2 partitions [6]. Online streaming algorithms consist of hashing-based methods (e.g. DBH [14], Hashing [2]) and heuristic-based methods (e.g. Greedy [2], HDRF [12]). The characteristics of vertex-cut streaming algorithms are summarized in Table I.

TABLE I
Vertex-cut Streaming Partitioning Algorithms

Algorithm	Time Cost	Quality
Hashing [2]	Low	Low
DBH [14]	Low	Low
Mint [19]	Medium	Medium
Greedy [2]	High	High
HDRF [12]	High	High
CLUGP	Low	High

From Table I, it can be seen that heuristic-based methods achieve better partitioning quality than hashing-based methods, and perform better in bulk synchronous processing systems [20]. However, heuristic-based methods are time-consuming, because a global status table needs to be locked each time a partition decision of an edge is made. Hashing-based methods and Mint perform faster than heuristic-based methods but are inferior in partition quality.

To this end, we study the problem of vertex-cut partitioning for large-scale web graphs to propose a new versatile partitioning architecture. We tackle the performance and quality challenge by exploring graph modularity, which has intuitive and profound connections with graph clustering and partitioning [21]–[24]. Our vision is to explore modularity-based clustering for enhancing the partitioning quality, employ streaming techniques for improving the efficiency, and break the ties of global structures for boosting system performance.

Nevertheless, a series of technical challenges arise in confronting clustering-based vertex-cut partitioning. First, existing

streaming clustering techniques only work for edge-cut partitioning, so that a high-degree vertex can hardly be accurately identified with partial degree information. Once such vertices are falsely identified for cutting, many replicas would be generated deteriorating system balance and communication efficiency. More, it is infeasible for correcting the false cutting with low-cost subsequent compensation, since it takes much communication overhead for high-degree vertex retrieving and reshuffling. Second, existing partitioning methods (e.g., HDRF [12]) are highly dependent on the global structure of vertex degrees or partial degrees, hindering its extensibility to large-scale graph streaming scenarios. The corresponding maintenance overhead becomes no more negligible, and even dominates the total time of graph application (e.g., pagerank) running on large partitions.

In our work, we present a CLUstering-based restreaming Graph Partitioning (CLUGP) architecture for vertex-cut partitioning over large-scale web graphs. Our algorithm follows a novel three-pass restreaming framework, which is pipelined as three steps, streaming clustering, cluster partitioning, and partition transformation. The streaming clustering step exploits the connection between clustering and vertex-cut partitioning for generating fine-grained clusters and reducing vertex replicas. The cluster partitioning step applies game theories for mapping generated clusters into specific partitions and further refines clustered results. Then, the partition transformation step transforms the cluster-based partitioning results into vertex-cut partitioning results.

Our contributions can be listed as follows.

- We propose a novel streaming partitioning architecture, which outperforms state-of-the-art solutions in terms of quality and scalability, for big web graph analytics.
- We study a new streaming clustering algorithm optimized for vertex-cut partitioning, by extending previous edge-cut streaming clustering algorithms.
- We provide a new method for mapping generated clusters to vertex-cut partitions by modeling the process by game theories. We theoretically prove the existence of Nash equilibrium and quality guarantees.
- We set up the parallel mechanism for CLUGP, getting rid of the computation bottleneck caused by frequent global table accessing by heuristic-based streaming algorithms.
- We empirically evaluate CLUGP with real datasets and real distributed graph systems. The results over representative algorithms, such as pagerank and connected component, demonstrate the superiority of our proposals.

The rest of the paper is organized as follows. We first formalize the vertex-cut partitioning problem in Section II. Then, we propose the CLUGP framework in Section III, investigate technical details of streaming clustering in Section IV, and study the partitioning game in Section V. We conduct extensive experiments with real datasets and real systems in Section VI. We conclude the paper in Section VIII. Notations of this paper are summarized in Table II.

TABLE II
NOTATIONS

Symbol	Notation
$G = (V, E)$	Directed graph with set of vertices V and edges E.
P	The set of k partitions $P = \{p_1, \cdots, p_k\}$.
$P(v)$	The set of partitions that hold vertex v .
$\|p_i\|$	The number of edges within p_i.
G_S	Edge streaming of the graph G.
G_C	The cluster set of graph G, $G_C = \{c_1, \cdots, c_m\}$.
$\|c_i\|$	The number of intra-cluster edges of c_i, $\|c_i\| = \|e(c_i, c_i)\|$.
m	The number of clusters, i.e., $\|G_C\| = m$.
$\varphi(a_i)$	The individual cost function of c_i under strategy a_i.
Φ	The potential function of a strategic game.
λ	Normalization factor.
τ	The imbalance factor.
$e(c_i, c_j)$	The set of edges that across from cluster c_i to c_j.
$e(c_i, V \backslash c_i)$	The set of edges that across from cluster c_i to other clusters.

II. PRELIMINARIES

A. Vertex-Cut Streaming Partitioning

Given a directed graph $G = (V, E)$, where V is a finite set of vertices, and E is a set of edges.

*Definition 1 (***Edge Streaming Graph Model):** The edge streaming graph model $G_S = \{e_1, e_2, \cdots, e_{|E|}\}$ assumes edges of an input graph $G = (V, E)$ arrive sequentially[1], where each edge $e_i = (u, v)$ indicates a directed edge form vertex u to vertex v.

In vertex-cut streaming partitioning, partitioning algorithms perform single- or multi-pass over the graph stream and make partitioning decisions for computational load-balancing and communication minimization.

*Problem 1 (***Vertex-Cut Streaming Partitioning):** Given k partitions $\{p_i\}_{1 \le i \le k}$, the vertex-cut streaming partitioning algorithm assigns each edge $e_i \in G_S$ to a partition p_i, such that $\cup_{1 \le i \le k} p_i = E$ and $p_i \cap p_j = \emptyset$ $(i \neq j)$. Each partition corresponds to a distributed node, each distributed node uses the divided graph edges to perform distributed graph analytic tasks.

B. Partition Quality

The main goal of partitioning algorithm is to improve the performance of the upper-level distributed graph processing system, like PowerGraph [14]. Considering the GAS model of the vertex-centric graph processing system, the graph computing messages are aggregated at the vertices and spread along the outgoing edges. After each iteration step, the master vertex gathers the message sent by mirror vertices, and synchronizes it to mirror vertices. Therefore, the number of edges determines the number of messages, and the number of mirror vertices determines the number of synchronizations, within an iteration.

To accelerate distributed graph processing, one should, 1) balance the computing time of each distributed node (computing cost); 2) reduce the number of synchronizations (communication cost). For the load balance part, we use the relative

[1]Without losing generality, we assume the edge stream of G arrives in the breadth-first (BFS) order, following the setting of [19], [25], [26], since most real web graphs are formulated and crawled in BFS order.

load balance $\tau \geq \frac{k \cdot \max |p_i|}{|E|}$ to denote the imbalance among partitions, where $|p_i|$ denotes the number of edges in partition p_i. $\tau \geq 1$ is a threshold for imbalance. For the synchronizations part, we use the replication factor $\frac{1}{|V|}\sum_{v \in V}|P(v)|$ to denote the proportion of mirror vertices, where $P(v)$ is the set of partitions holding vertex v, and $|P(v)|$ refers to the number of partitions holding v.

The vertex-cut partitioning can thus be modelled as an optimization problem [2], [12], as follows.

$$minimize \frac{1}{|V|}\sum_{v \in V}|P(v)| \quad s.t. \frac{k \cdot \max |p_i|}{|E|} \leq \tau \tag{1}$$

By minimizing the replication factor, the communication cost during graph computation is also minimized. By balancing the workload balance, the computing task of each computing node can be balanced.

C. Web Graphs and Modularity

We discuss two properties of web graphs, *power-law distribution* of vertex degrees, and *modularity*. The former determines partitioning strategy, and the latter paves the road to quality partitioning.

Distribution. According to Kumaret et al. [27], [28] and Kleinberg et al. [29], the degree distribution of web graphs follows *power law approximately*. That is, given a specific degree x, the number of vertices follows power-law distribution, $f(x) \propto x^{-\alpha}$, where α is a constant and $\alpha > 0$. The fact that web graphs are featured with power-law distributions are commonly accepted [30]–[32].

Modularity. Modularity [33] [34] is a quality metric for graph clustering, which is defined as the fraction of the edges that fall within the given groups minus the expected fraction if edges were distributed at random. The value range of modularity is thus $[0, 1]$. Graphs or networks with a high modularity has tight connections for intra-cluster vertices, and loose connections for inter-cluster vertices. TABLE III reports the modularity values of a series of representative algorithms over different web graphs. It shows that all web graphs are with a high modularity value (close to 1).

Next, we discuss how the modularity property facilitates the web graph partitioning, by investigating the partitioning framework, CLUGP.

TABLE III
MODULARITY[2] OF WEB GRAPHS

Algorithms	Datasets	Modularity
Fast MBGC [35]	webbase-2001	0.98
	uk-2005	0.98
Improved MOH [36], [37]	arabic-2005	0.99
	uk-2002	0.99
	it-2004	0.97
	sk-2205	0.97
MMVNS [38]	eu-2005	0.94
	in-2004	0.97

III. ARCHITECTURE

The CLUGP architecture consists of three steps, which process streamed graph edges in three passes, as shown in Figure 1. First, we improve the method of vertex stream clustering proposed by Hollocou et al. [39] to produce fine-grained clusters (*streaming clustering step*, Section III-A). Second, we investigate game theories to assign clusters to a set of partitions, such that the number of edges across partitions is minimized and the storage of partitions is balanced (*cluster partitioning step*, Section III-B). Last, we propose a heuristic method to transform cluster partitions into edge partitions (*partitioning transformation step*, Section III-C).

A. First Pass: Streaming Clustering

Web graphs are featured with high modularity which supports high-quality graph clustering. The first step is to exploit the connections between clustering and partitioning, so that graph structural information can be leveraged to supervise partitioning, laying the foundation for subsequent steps.

*Problem 2 (**Streaming Clustering**):* Suppose a streaming graph $G_S = \{e_1, e_2, \cdots, e_{|E|}\}$ and the maximum cluster volume V_{max}. The problem is to assign each vertex v to one of the m clusters $\{c_i\}_{1 \leq i \leq m}$, such that the edge-cutting is minimized. Notice that conditions $\cup_{1 \leq i \leq m} c_i = V$ and $|c_i| \leq V_{max}$ should be met. The output is a table mapping a vertex to a cluster, i.e., $\{\langle v_i, c_j \rangle\}$.

The modularity-based graph clustering can potentially be used for exploiting the structural information of web graphs. However, there is no existing solution for streaming edge clustering which can be directly used for vertex-cut partitioning. In our work, we improve the vertex streaming clustering algorithm [39] for adapting to vertex-cut graph partitioning. The challenge is that clustering and partitioning are with different optimization targets. The goal of vertex clustering is to minimize edge-cutting, while vertex-cut partitioning is on minimizing vertex replicas. We use an example to show the difference of the two optimization targets in Figure 1.

For vertex clustering, as shown in Figure 1 (c-1), vertices v_0 to v_6 are uniquely assigned to clusters c_0 to c_3, thus there exist cutting edges, but not vertex replicas. For example, $e(3, 5)$ is a cutting edge generated by the vertex clustering algorithm, while v_3 and v_5 belong to clusters c_3 and c_1, respectively.

For edge partitioning, as shown in Figure 1 (c-2), two replicas of vertex v_0 and one replica of v_3 are generated, as highlighted by dashed circle. The existence of vertex replicas eliminates edge-cutting for "real" edges. The dashed lines represent virtual edges, connecting master and mirror vertices. For example, v_0 is the master vertex, and v_0' and v_0'' are mirror vertices and replicas of v_0. Hence, $e(v_0, v_0')$ is a virtual edge[3].

[2]The modularity equation [33] [34] can be written as, $Q = \frac{1}{2|E|}\sum_{u,v \in V}[X_{uv} - \frac{d_u d_v}{2|E|}\delta(u, v)]$, where (u, v) is an arbitrary vertex pair in G, $|E|$ is the number of edges in G, X is the adjacent matrix of G, d_v is the degree of vertex v, and $\delta(u, v) = \sum_{c \in G_C} S_{uc}S_{vc}$. S_{uc} is 1, if vertex u belongs to cluster c; and is 0, otherwise.

[3]Without causing any ambiguities, we also call virtual edges as cutting edges for vertex-cut partitioning in the rest of the paper.

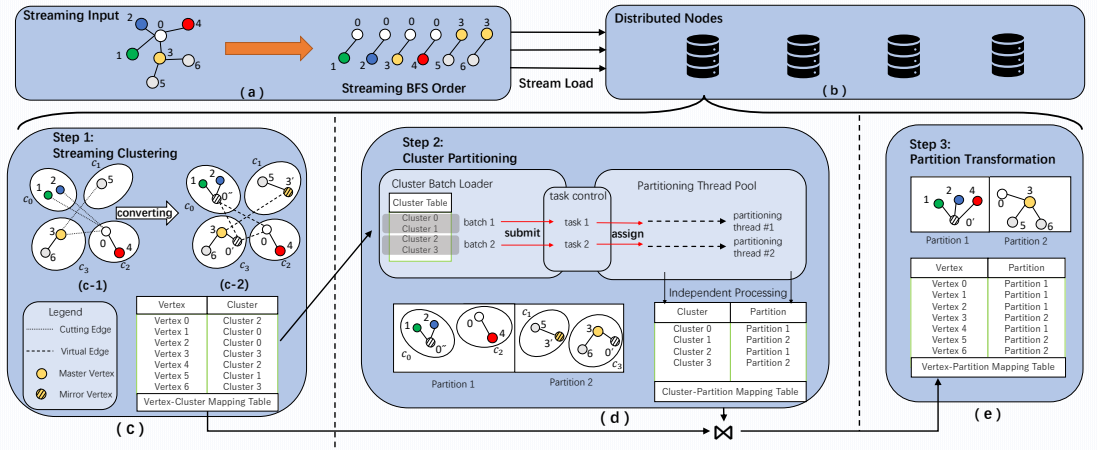


Fig. 1. CLUGP Architecture

We therefore propose a new vertex clustering framework, tailored for minimizing vertex replicas. It produces a coarse-grained vertex-cut partitioning result, in the form of vertex-cluster pairs. Details are shown in Section IV.

B. Second Pass: Cluster Partitioning

The second step is to assign the generated clusters to the given set of partitions, which can be formalized as follows.

*Problem 3 (**Cluster Partitioning**):* Suppose m clusters $G_C = \{c_i\}_{1\leq i\leq m}$ and k partitions $P = \{p_i\}_{1\leq i\leq k}$. The problem is to assign each cluster $c_i \in G_C$ to a partition $p_i \in P$, while minimizing edge-cutting and imbalance. The output is a table mapping a cluster to a partition, $\{\langle c_i, p_j\rangle\}$.

The optimization target of cluster partitioning problem has two parts, load balancing and edge-cutting minimization. For the load balancing part, without losing generality, we use $\lambda\frac{1}{k}\sum_{p_i\in P}|p_i|^2$ to denote imbalance cost of k partitions [40], [41]. Parameter λ is for normalization. It is obvious that the lowest imbalance is achieved, when partitions are of the same size. For the edge-cutting part, we can use the number of inter-partition (virtual) edges as the cost. By integrating the two costs, we can get the overall cluster partitioning cost function.

Definition 2 (Cluster Partitioning Cost): The overall cluster partitioning cost is defined as:

$$Cost = \underbrace{\lambda\frac{1}{k}\sum_{p_i\in P}|p_i|^2}_{load\ balancing} + \underbrace{\sum_{p_i\in P}|e(p_i, V\backslash p_i)|}_{edge-cutting} \tag{2}$$

where $\sum_{p_i\in P}|e(p_i, V\backslash p_i)|$ is the number of cutting edges.

It can be shown that finding the global optimal solution targeted on Equation 2 is NP-hard, by reducing it from the set cover problem. To get a sub-optimal solution, we treat each cluster as a player. Then, the cluster partitioning problem can be modelled as a strategic game. For a cluster, the selection of a partition can thus be regarded as a rational game strategy, where each cluster affects others' costs and meanwhile minimizes its own cost by strategically manipulating its partition choice. Thus, the optimization problem is transformed into finding the Nash equilibrium of the game, so that each player/cluster minimizes its own cost.

However, the retrieval of the Nash equilibrium is compute-bound, a.k.a., the overhead of computation dominates that of I/O. So, we design a parallel strategy to accelerate the cluster partitioning process. As shown in Figure 1(d), clusters generated are grouped into batches, where each batch is executed by an independent thread to find the Nash equilibrium.

More technical details and analysis of cluster partitioning problem are covered by Section V.

C. Third Pass: Partitioning Transformation

By joining the outputs of the first two steps, we can map a vertex to a partition. For mapping an edge to a partition, we utilize partitioning transformation as the third step of CLUGP. It accesses edge streams for further refining the cluster-based partitioning result of the second step.

*Problem 4 (**Partitioning Transformation**):* Given the mapping table from vertices to partitions, $\{\langle v_i, p_j\rangle\} = \{\langle v_i, c_j\rangle\} \bowtie \{\langle c_i, p_j\rangle\}$, the problem is to transform vertex mapping table $\{\langle v_i, p_j\rangle\}$, to edge mapping table $\{\langle e_i, p_j\rangle\}$, which serves as the partitioning result.

For each edge $e(u, v) \in G_S$, partitions $P(u)$ and $P(v)$ are accessed to determine which partition e is assigned to. The two partitions are retrieved based on the joining results of the first two steps. Notice that we do not explicitly maintain the joining results for reducing memory cost. Instead, one can quickly map a vertex to a partition by querying the two mapping tables sequentially. The determination of e assignment has been addressed in previous two steps, following the optimization target of edge-cutting and imbalance. The de facto assignment of edges is implemented in the third step, by traversing the streaming graph. The details are covered in Algorithm 1.

For each edge $e(u, v)$, if neither of $P(u)$ and $P(v)$ can accommodate e, then e will be assigned to an underflow partition, for workload balancing (lines 6-14). When u and v are in the same partition, e will be assigned to the partition (lines 15-16). If $u(v)$ has mirror vertices, which means the

Algorithm 1 Partition Transformation

Input Cluster Partition Strategy a^*, Cluster Set $clu[]$, Vertex Degree $deg[]$, Load Balance Factor τ
Output Partition Result

1: Let a_i be the partition choice of cluster c_i;
2: Initialize the array of load, $L_{max} = \tau \frac{|E|}{k}$;
3: **for** $e(u, v) \in G_S$ **do**
4: $c_u, c_v \leftarrow clu[u], clu[v]$;
5: $p_u, p_v \leftarrow a_u, a_v$;
6: **if** $|p_u| \geq L_{max}$ **or** $|p_v| \geq L_{max}$ **then**
7: **if** $|p_u| < L_{max}$ **then**
8: assign e to p_u;
9: **continue**;
10: **if** $|p_v| < L_{max}$ **then**
11: assign e to p_v;
12: **continue**;
13: **for** $p_i \in P$ **do**
14: assign e to p_i if $|p_i| < L_{max}$; ▷ Load Balance
15: **else if** p_u **equal** p_v **then**
16: assign e to p_u
17: **else**
18: **if** either u or v has mirror vertices **then**
19: assign e to p_v or p_u;
20: **else**
21: assign e to p_u if $deg[v] > deg[u]$;
22: assign e to p_v if $deg[u] > deg[v]$; ▷ Reduce Replicas

vertex $u(v)$ has been replicate during step 1 (Section IV), e will be assigned to the partitions where $u(v)'s$ mirror vertex belongs to (lines 18-19). Otherwise, the vertex with a higher degree will be cut (lines 21-22) to reduce vertex replicas, similar to [12], [18], [42].

During the transformation, there is a user-specified parameter, i.e., imbalance factor τ, on controlling the partition size. Compared to τ, V_{max} of the first step is merely the upper limit of cluster capacities. The purpose of τ is to further improve partition balancing from coarse-grained cluster-level to fine-grained partition-level. This way, edges that incur partitioning overflowing are moved to underflow partitions, strictly conforming to the system parameter τ.

For the third step, CLUGP traverses the edge stream to perform partition transformation that merely takes $O(1)$ space cost, since we only need a k elements array to store the partition size. To perform transformation, the query over vertex-to-partition mapping tables only takes $O(1)$ time for each edge. The total time complexity is $O(|E|)$.

This way, our architecture can be well parallelized. Of the system, each distributed node accesses partial streaming edges and performs the three steps, clustering, game processing, and transformation, locally. Further, game processing of a distributed node can be parallelized by multi-threading. After the three steps, the final graph partitioning result is obtained by combining the partial partitioning results of distributed nodes.

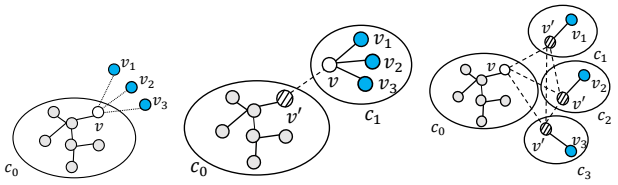
IV. Streaming Clustering

In this section, we investigate a new streaming clustering algorithm. In particular, we propose the allocation-splitting-migration framework in Section IV-A, and conduct theoretical analysis in Section IV-B.

A. Allocation-splitting-migration Framework

Modularity-based strategies [21], [35], [39], [43] are widely used for web graph clustering. The first and only streaming version of modularity-based graph clustering algorithm, *Holl*, is proposed by Hollocou et al. [39].

Holl presented a modularity-based *allocation-migration* framework for streaming clustering. Holl has proved that the heuristic strategies of the framework can help improve the modularity effectively. However, Holl cannot be directly applied for graph partitioning, because the *allocation-migration* framework of Holl incurs high replication factors. CLUGP improves Holl by adding a splitting operation, and thus construct a new *allocation-splitting-migration*. We will prove that the splitting operation can improve the modularity.



(a) before splitting (b) CLUGP with splitting (c) Holl without splitting

Fig. 2. An Example of Cluster Splitting for Streaming Clustering (Vertex IDs follow BFS order, e.g., $G_S = \{e(v, v_1), e(v, v_2), e(v, v_3), \cdots\}$

Consider the example in Figure 2. Suppose cluster c_0 reaches the maximum cluster volume V_{max}, in Figure 2(a). In Holl, to handle incoming edges $e(v, v_1)$, $e(v, v_2)$, and $e(v, v_3)$, cluster c_0 remains as it is, while new clusters, c_1, c_2, and c_3, are generated to accommodate successive streaming edges, as shown in Figure 2(c). According to the *allocation-migration* mechanism of Holl, cluster c_0 never splits, so that the master vertex of v is always subordinated to c_0, and the mirror vertex v' exists in c_1 to c_3. After clustering, the number of master vertices is 10, and the number of mirror vertices is 3. Based on Equation 1, the replication factor of Holl is $\frac{10+3}{10} = \frac{13}{10}$.

CLUGP adds a splitting operation, as highlighted in Algorithm 2. The splitting operation can effectively chop high-degree vertices to reduce replicas in the streaming clustering process, since high-degree vertices tend to form new clusters with subsequent neighboring vertices. In Figure 2 (b), with the splitting operation, c_0 is split into two clusters, c_0 and c_1, and the master vertex of v is assigned to new cluster c_1 meanwhile generating a mirror vertex v' in c_0. In this case, the replication factor of CLUGP is $\frac{11}{10}$, which is smaller than that of Holl.

The details about the improved streaming clustering algorithm of CLUGP are covered in Algorithm 2. The clustering process builds clusters in a bottom-up manner, where each cluster initially has one vertex. For an incoming edge $e(u, v)$ of streaming G_S, the two incident vertices of e are assigned to two clusters c_u and c_v (lines 3-5). The volume of a cluster is defined as the sum of the degrees of master vertices in the cluster. A cluster overflows, if the volume of a cluster exceeds its maximum capacity (V_{max}). Holl handles cluster overflowing, by assigning incoming edges to a new cluster. CLUGP handles cluster overflowing, by splitting the original

Algorithm 2 Streaming Graph Clustering of CLUGP

Input Edge Stream G_S, Maximum Cluster Volume $V_{max} = \frac{|E|}{k}$
Output Cluster Set G_C, Cluster ID $clu[]$, Vertex Degree $deg[]$

1: Initialize the array of degree, cluster, and volume;
2: $vol(c_i)$ returns the volume of cluster c_i
3: **for** $e(u, v) \in G_S$ **do**
4: **if** $clu[u]$ is NULL **or** $clu[v]$ is NULL **then** } Allocation
5: Assign a new cluster ID for u or v;
6: $c_u \leftarrow clu[u]$, $c_v \leftarrow clu[v]$;
7: $deg[u] \leftarrow deg[u] + 1$, $deg[v] \leftarrow deg[v] + 1$;
8: $vol(c_u) \leftarrow vol(c_u) + 1$, $vol(c_v) \leftarrow vol(c_v) + 1$;
9: **if** $vol(c_u) \geq V_{max}$ **then**
10: Assign a new cluster ID for u;
11: $c'_u \leftarrow clu[u]$, mark u as divided vertex;
12: $vol(c_u) \leftarrow vol(c_u) - deg[u]$;
13: $vol(c'_u) \leftarrow vol(c'_u) + deg[u]$;
14: **if** $vol(c_v) \geq V_{max}$ **then** } Splitting
15: Assign a new cluster ID for v;
16: $c'_v \leftarrow clu[v]$, mark v as divided vertex;
17: $vol(c_v) \leftarrow vol(c_v) - deg[v]$;
18: $vol(c'_v) \leftarrow vol(c'_v) + deg[u]$;
19: $c_u \leftarrow clu[u]$, $c_v \leftarrow clu[v]$;
20: **if** $vol(c_u) < V_{max}$ **and** $vol(c_v) < V_{max}$ **then**
21: **if** $vol(c_u) \leq vol(c_v)$ **then**
22: Migrate u from c_u to c_v;
23: Update $vol(c_u)$ and $vol(c_v)$;
24: **else** } Migration
25: Migrate v from c_v to c_u;
26: Update $vol(c_u)$ and $vol(c_v)$;
return G_C

cluster into two smaller clusters for generating fewer replicas (lines 9-19). At last, the algorithm migrates an incident vertex of edge e from a smaller cluster to a bigger cluster (lines 20-26). The process repeats until all incoming edges of G_S are processed, so that the cluster set G_C is generated as the output.

In the streaming clustering step, we use the vertex-cluster mapping table to store the cluster that a vertex belongs to. To get the degree of vertices, we also need an array to record the degree. So, the space cost of this step is $O(|V|)$. The time cost of modifying and querying the mapping table or degree array is $O(1)$. The process traverses all incoming edges, so the time cost of Algorithm 2 is $O(|E|)$.

During the splitting operation, we mark the vertex that causes cluster splitting as *divided vertex* (lines 11, 16). Then, we can quickly find which vertex has been replicated and which cluster its mirror vertices belongs to. In Figure 2 (b), during the clustering step, vertex v is marked as a divided vertex. So, when processing edge $e(v, v_1)$, we can quickly find that $e(v, v_1)$ should be assigned to c_1 and generate a mirror vertex v' in c_0, thus there exists a cutting edge between c_0 and c_1 (denoted as dashed lines). By the way, when both vertices of an edge are marked as divided vertices, we split the vertex with a higher degree vertex and assign the edge to the cluster where lower degree vertex belongs to, which is shown to be effective in reducing replication factor for power-law graphs [12], [18], [42].

We can get two facts from Algorithm 2: a) allocation and splitting operations increase at most one vertex replica at an iteration; b) migration operation reduces at most one vertex replica at an iteration. But, a seemingly plausible observation, that the splitting operation triggers more vertex replicas, is not correct, because the splitting operation of CLUGP can reduce total number of replicas. This is guaranteed by Algorithm 2. First, if the splitting operation is not triggered, CLUGP is degenerated into Holl, so that the two have the same replication factor. Second, if the splitting operation is triggered, CLUGP can derive a smaller replication factor than Holl. In summary, CLUGP derives a smaller replication factor than Holl.

B. Analysis

We prove that CLUGP can effectively reduce the replicate factor, based on the properties for power-law graphs. According to [44], for a power-law graph, if we remove $\frac{1}{\theta}$ of vertices with the highest degrees, then the maximum degree $\tilde{M}$ of the remnant subgraph can be approximated by $\tilde{M} = \gamma\theta^{1/(1-\alpha)}$, where γ is the global minimum degree, and α is the exponent of the power-law graph. Based on this property, we can get that, given a specific degree d, the fraction θ of vertices satisfying $\{v|degree(v) \geq d\}$, is:

$$\theta = \left(\frac{\gamma}{d-1}\right)^{\alpha-1} \tag{3}$$

Equation 3 can be used for describing the worst case of CLUGP, a.k.a., the highest replication factor of the splitting operation. The details are covered in Theorem 1.

Theorem 1: The upper bound of replication factor of CLUGP is always no larger than that of Holl.

Proof 1: To prove the theorem, we only need to show that, the upper bound of replicate factor of CLUGP is no larger than the replication factor of Holl, $RF_{clugp} \leq RF_{holl}$.

Given the number of replicas r of vertex v, we have $degree(v) \geq d_{min}^{clugp}(r)$, where $d_{min}^{clugp}(r)$ denotes the minimum degree of the vertex v, if v has been replicated r times. Based on Equation 3, we can get the maximum number of vertices with r replicas equals to $|V|\left(\frac{\gamma}{d_{min}^{clugp}(r)-1}\right)^{\alpha-1}$, by multiplying $|V|$ with θ.

Considering the worst case, let the number of clusters be m, for any vertex v with degree $degree(v)$, there can be most $max(degree(v)-1, m-1)$ replicas for v. If the $degree(v)$ less than m, the worst case can be happened when all $degree(v)$ edges of the vertex are assigned to different clusters. Otherwise, each cluster $\{c_i\}_{1\leq i\leq m}$ has an edge (mirror vertex) of v. So, a sequence of maximum replicas can be generated by vertices is $\{m-1, m-2, \cdots, \gamma-1\}$[4], each replica corresponds to a fraction of vertices $\theta_r^{clugp} = \left(\frac{\gamma}{d_{min}^{clugp}(r)-1}\right)^{\alpha-1}$, where $\gamma - 1 \leq r \leq m - 1$. Thus, we can get the upper bound of replicate factor of CLUGP as follows.

$$\begin{aligned} RF_{clugp} \leq & (m-1)\cdot\theta_{m-1}^{clugp} + (m-2)\cdot(\theta_{m-2}^{clugp} - \theta_{m-1}^{clugp}) \\ & + \cdots + (m-\gamma)(\theta_{\gamma-1}^{clugp} - \theta_{\gamma}^{clugp}) \\ = & \theta_{m-1}^{clugp} + \cdots + \theta_{\gamma}^{clugp} + (m-\gamma)\cdot\theta_{\gamma-1}^{clugp} \end{aligned} \tag{4}$$

[4]For power-law graphs, γ is much smaller than m, $\gamma << m$. Usually, γ equals 1.

Similarly, for Hollocou's algorithm, we can have that:

$$RF_{holl} \leq \theta_{m-1}^{holl} + \cdots + \theta_{\gamma}^{holl} + (m-\gamma) \cdot \theta_{\gamma-1}^{holl} \tag{5}$$

Based on Theorem 2, we know that $d_{min}^{clugp}(r \geq 2) > d_{min}^{holl}(r \geq 2)$. Substituting it into Equation 3, we get $\theta_r^{clugp} \leq \theta_r^{holl}$, then combining it into Equations 4, 5, we can get $RF_{clugp} \leq RF_{holl}$. So, the theorem is proved.

Theorem 2: Suppose two vertices $v_c, v_h \in V$, where v_c and v_h are processed by CLUGP and Holl, respectively. If v_c and v_h are both with r replicas, the minimum degree of v_c must be no less than that of v_h. Formally, $d_{min}^{clugp}(r) \geq d_{min}^{holl}(r)$.

Proof 2: Let $R(v)$ denotes the number of replicas of vertex v. For CLUGP, it can be obviously seen from Figure 2(b) that, if $R(v) = 0$, it means the vertex v does not need any replicate, so we have $d_{min}^{clugp}(r) = 1$, if $R(v) = 1$, it means the vertex v has at least one splitting operation, so we have $d_{min}^{clugp}(r) = 2$. Similarly, when $R(v) = 2$, it means we must fill up the cluster c_1 and split the vertex v out of c_1. To better prove the theorem, we let the degree of v's neighbors equal to the global maximum degree d_{max}, which is the worst case of CLUGP, since the splitting operation can be triggered intensively. So on when $R(v) = r \geq 2$ we have the following equation sets:

$$\begin{cases} 1 + |Ne_1| + |Ne_1| \cdot d_{max} & = V_{max} \\ 1 + |Ne_1| + |Ne_2| + |Ne_2| \cdot d_{max} & = V_{max} \\ 1 + |Ne_1| + |Ne_2| + |Ne_3| + |Ne_3| \cdot d_{max} & = V_{max} \\ \vdots & \\ 1 + |Ne_1| + \cdots + |Ne_{r-2}| + |Ne_{r-1}| \cdot d_{max} & = V_{max} \\ |Ne_r| & = 1 \end{cases} \tag{6}$$

, where $|Ne_i|$ denotes the number of neighbor vertices that vertex v needed to fill up the cluster c_i. And next, we can get the solution as follows:

$$\begin{cases} |Ne_i| = \frac{V_{max}-1}{d_{max}} \cdot (\frac{d_{max}}{1+d_{max}})^i & , 1 \leq i \leq r-1 \\ |Ne_i| = 1 & , i = r \end{cases} \tag{7}$$

After summing the number of edges needed for v, we have:

$$\begin{aligned} d_{min}^{clugp}(r \geq 2) &= 1 + |Ne_1| + \cdots + |Ne_{r-1}| + |Ne_r| \\ &= (V_{max} - 1) \left[1 - (1 - \frac{1}{1+d_{max}})^{r-1} \right] + 2 \end{aligned} \tag{8}$$

That is, if a vertex v has been replicated $r \geq 2$ times, the degree of v must satisfy $degree(v) \geq d_{min}^{clugp}(r)$. For Holl, since it does not have splitting operation to migrate vertex v out of c_0, each neighbour of vertex v will be allocated a independent cluster, thus, we can easily get that $d_{min}^{holl}(r \geq 2) = r - 1$. Additionally, $d_{min}^{holl}(r) = d_{min}^{clugp}(r)$ when $r \leq 1$, so we only consider the situation that $r \geq 2$. Since for power-law graph $d_{max} \gg 1$, $r - 1 > 0$, thus $\frac{1}{1+d_{max}} \to 0$ and we can get $(1 - \frac{1}{1+d_{max}})^{r-1} \sim 1 - \frac{r-1}{1+d_{max}}$. Therefore, we can get that:

$$\begin{aligned} d_{min}^{clugp}(r \geq 2) &= (V_{max} - 1) \left[1 - (1 - \frac{1}{1+d_{max}})^{r-1} \right] + 2 \\ &= (V_{max} - 1) \cdot (\frac{r-1}{1+d_{max}}) + 2 \\ &> (1 + d_{max}) \cdot (\frac{r-1}{1+d_{max}}) + 2 \\ &= r + 1 > r - 1 = d_{min}^{holl}(r \geq 2) \end{aligned} \tag{9}$$

where $V_{max} = \frac{|E|}{k} > d_{max}$. Hence, the theorem is proved.

We next show the relationship between splitting operation and modularity.

Theorem 3: The splitting operation can help in increasing the modularity.

Proof 3: Assume the modularity of t streaming edges $\{e_1, \cdots, e_t\}$ be represented by Q_t. For a new edge $e_{t+1}(u, v)$, the modularity Q_{t+1} based on e_{t+1} can be defined as [39]:

$$Q_{t+1} = Q_t + 2[\delta(u,v) - \frac{vol_t(clu(u)) + vol_t(clu(v)) + 1 + \delta(u,v)}{2|E|}]$$

where $clu(u)$ denotes the cluster of vertex u, $vol_t(clu(u)$ denotes the volume of cluster $clu(u)$ after e_t arrived. $\delta(u, v) = 1$ if u and v belongs to the same cluster, and 0 otherwise.

It can be observed that, if e_{t+1} triggers the splitting operation. According to the setting of splitting operation, vertices u and v will be assigned to the same cluster, thus $\delta(u, v) = 1$. Otherwise, vertices u and v will be separately stored in different clusters, thus $\delta(u, v) = 0$. Therefore, splitting operation helps in increasing the value of modularity.

V. Game Theory-based Cluster Partitioning

In this section, we study a suboptimal solution for the problem of cluster partitioning. We formalize the problem of cluster partitioning and prove the existence of Nash equilibrium in Section V-A. We theoretically prove the quality guarantee for the game in Section V-C.

A. Modeling of Cluster Partitioning Problem

In strategic games, a player aims to choose the strategy that minimizes his/her own individual cost. The game continues until a steady state is achieved, in which no player can benefit by unilaterally changing its strategy.

In our work, clusters $\{c_i\}_{i \leq m}$ can be considered as independent and competing players in a strategy game. For each cluster c_i, there can be k choices for choosing a partition. Let strategy a_i be the partition choice of cluster c_i. Then, the strategy profile $\Lambda = \{a_i\}_{i \leq m}$ consists of the strategies for all clusters. For a fixed strategy profile Λ, each strategy $a_i \in \Lambda$ refers to the partition that c_i belongs to. We use $|a_i|$ to represent the number of edges of the partition that c_i belongs to. For example, if a_i refers to p_j, $|a_i|$ equals to the size of p_j, i.e., $|e(p_j, p_j)|$.

Given a strategy profile Λ, the global deployment cost is denoted as $\varphi(\Lambda)$, and the deployment cost of each cluster c_i is denoted as $\varphi(a_i)$. Intuitively, a lower deployment cost corresponds to a higher partition quality. Based on the cluster partitioning optimization target (Equation 2), the global deployment target $\varphi(\Lambda)$ can be defined as:

$$\varphi(\Lambda) = \underbrace{\lambda \frac{1}{k} \sum_{i=1}^{k} |p_i|^2}_{load\ balancing} + \underbrace{\sum_{i=1}^{k} |e(p_i, V \backslash p_i)|}_{edge-cutting} \tag{10}$$

The game-based solution ensures that the global partitioning optimization target $\varphi(\Lambda)$ (Equation 10) can be achieved, if each cluster c_i's locally minimized partitioning cost $\varphi(a_i)$ is

achieved. We first explain the local optimization target for each cluster/player. Then, we prove that the local optimization targets of clusters can be integrated as the global target.

The local cost of a cluster has two parts, *load balancing* and *edge-cutting* (Equation 11), which is consistent with the form of the global cost function $\varphi(\Lambda)$ (Equation 10).

We use variable $|c_i|$ to denote the number of edges of cluster c_i, formally, $|c_i| = |e(c_i, c_i)|$. To ensure the load balance, we should assign the large-scale clusters to the partitions with small size. So, for each cluster c_i and its partition a_i, the cost of imbalance can be defined as $\varphi^{load}(a_i) = \frac{1}{k}|c_i||a_i|$. To reduce the number of cut edges, the cluster c_i should be placed in the partition that has the least number of cut edges from other partitions. Therefore, the cost of edge-cut can be defined as $\varphi^{cut}(a_i) = \frac{1}{2}(|e(c_i, V \backslash a_i)| + |e(V \backslash a_i, c_i)|)$. In conclusion, we can get a cluster c_i's cost under the partition a_i.

$$\begin{aligned}\varphi(a_i) =& \lambda\varphi^{load}(a_i) + \varphi^{cut}(a_i) \\ =& \underbrace{\frac{\lambda}{k}|c_i| \cdot |a_i|}_{load\ balancing} + \underbrace{\frac{1}{2}(|e(c_i, V \backslash a_i)| + |e(V \backslash a_i, c_i)|)}_{edge-cutting}\end{aligned} \tag{11}$$

We next show how the local cost function (Equation 11) can be derived from the global cost function (Equation 10).

$$\begin{aligned}\varphi(\Lambda) =& \lambda\frac{1}{k}\sum_{i=1}^{k}\sum_{c_j \in p_i}|c_j||p_i| + \frac{1}{2}\sum_{i=1}^{k}(|e(p_i, V \backslash p_i)| + |e(V \backslash p_i, p_i)|) \\ =& \lambda\frac{1}{k}\sum_{i=1}^{m}|c_i||a_i| + \frac{1}{2}\sum_{i=1}^{k}\sum_{c_j \in p_i}(|e(c_j, V \backslash a_j)| + |e(V \backslash a_j, c_j)|) \\ =& \lambda\frac{1}{k}\sum_{i=1}^{m}|c_i||a_i| + \frac{1}{2}\sum_{i=1}^{m}(|e(c_i, V \backslash a_i)| + |e(V \backslash a_i, c_i)|) \\ =& \lambda\sum_{i=1}^{m}\varphi^{load}(a_i) + \sum_{i=1}^{m}\varphi^{cut}(a_i) = \sum_{i=1}^{m}\varphi(a_i)\end{aligned} \tag{12}$$

Consequently, minimizing the global deployment cost is equivalent to minimizing the set of individual deployment costs. Then, we can define the Nash equilibrium of the cluster partitioning game as follows.

Definition 3 (Nash equilibrium): A strategy decision profile $\Lambda^* = \{a_1^*, a_2^*, \cdots, a_m^*\}$ of all clusters is a Nash equilibrium [45], if all clusters achieve their locally optimization targets. This way, no cluster has an incentive for unilaterally deviating the strategy for a lower cost.

Algorithm 3 shows the process of finding Nash equilibrium. Initially, clusters of G_C randomly choose partitions with equal probabilities. For each cluster, the partition set P is traversed to get its best strategy (partition choice) that incurs the minimum cost (lines 6-10), and the current strategy is updated if necessary (line 12). The iterative subroutine (lines 4-13) continues until no cluster updates its partition strategy.

Theorem 4: The time complexity of each round is $\Theta(m)$, the space complexity of game is $O(m)$.

Proof 4: It is clear that the number of clusters in $N[]$ is $\sum_{c_i \in G_C}|N(c_i)|$, where $|N(c_i)|$ is the number of neighbors

Algorithm 3 Nash equilibrium

Input Cluster Set G_C, Partition Set P, Cluster Neighbors $N[]$
Output Nash equilibrium

```
1:  Initial individual cost for each cluster.
2:  Assign each cluster c_i ∈ G_C to a random partition.
3:  repeat
4:      for c_i ∈ G_C do
5:          minCost ← ∞, partition ← ∅;
6:          for p_i ∈ P do
7:              put cluster c_i into p_i;
8:              for c' ∈ N[c_i] do
9:                  update individual cost of c_i based on c';
10:             update minCost, partition;
11: until Nash equilibrium
```

of cluster c_i. In Algorithm 3, each cluster needs to traverse in total $|N(c_i)|$ clusters to compute the individual cost (lines 8-9). Since all clusters perform this step in a round-robin fashion, the time complexity of each round for all players is $O(\sum_{c_i \in G_C}|N(c_i)|)$. Considering the average case of $\sum_{c_i \in G_C}|N(c_i)|$, the traverse time complexity is $\Theta(m)$. In the game process, we need a table mapping from cluster to partitions, so the space complexity is $O(m)$.

B. Existence of Nash Equilibrium

The existence of Nash equilibrium supports the deployment of efficient cluster partitioning, as it enables an alternative solution for retrieving a set of local optimizations instead of a global optimization. Hence, we proceed to prove the existence of a Nash equilibrium for the cluster partitioning problem. We start by showing an exact potential function $\Phi(\Lambda)$ (Definition 4), which is defined based on the global cost function $\varphi(\Lambda)$ (Equation 10). It is important to note that, if a game has an exact potential function, the game must have Nash equilibrium.

Definition 4 (Exact Potential Function): According to Equations 11 and 10, the exact potential function of the game can be defined as:

$$\Phi(\Lambda) = \lambda\frac{1}{2k}\sum_{i=1}^{k}|p_i|^2 + \frac{1}{2}\sum_{i=1}^{k}|e(p_i, V \backslash p_i)| \tag{13}$$

The potential function is useful in the analysis of game equilibrium, since the "incentives" of all players, e.g., load balancing and egde-cutting, are reflected in the function, so that the pure Nash equilibrium can be found by locating the local optima of the potential function. Next, we show the cluster partitioning game is an exact potential game, so that it always converges to a Nash equilibrium.

Theorem 5: The cluster partitioning game is an exact potential game.

Proof 5: To prove a game is an exact potential game, we need to show that when the partition choice of cluster c_i is changed, the value change of potential function (Equation 13) is the same as the value change of c_i's individual cost (Equation 11), according to Lemma 19.7 of [46].

We define a_{-i} as the strategies of all clusters except cluster c_i, under the strategy profile Λ, formally $a_{-i} = \Lambda - \{a_i\}$.

Let $\Phi(a'_i, a_{-i})$ be the potential value, when cluster c_i changes its partition choice from a_i to a'_i, and other clusters remain unchanged. Thus, to prove the theorem, it is equivalent to show $\Phi(a'_i, a_{-i}) - \Phi(a_i, a_{-i}) \equiv \varphi(a'_i, a_{-i}) - \varphi(a_i, a_{-i})$. According to Equation 11, we have $\Delta\varphi = \varphi(a'_i, a_{-i}) - \varphi(a_i, a_{-i}) = \lambda\frac{1}{k}|c_i|(|a'_i| + |c_i| - |a_i|) + \frac{1}{2}(\Delta E)$ and $\Delta E = |e(c_i, a_i)| + |e(a_i, c_i)| - |e(c_i, a'_i)| - |e(a'_i, c_i)|$.

Similarly, according to Equation 13, we can obtain the potential function difference as: $\Delta\Phi = \Phi(a'_i, a_{-i}) - \Phi(a_i, a_{-i}) = \Delta\Phi_{load} + \Delta\Phi_{cut}$. Hence, for the first part we have $\Delta\Phi_{load} = \lambda\frac{1}{2k}(2|a'_i||c_i| + |c_i|^2 - 2|a_i||c_i| + |c_i|^2) = \lambda\frac{1}{k}|c_i|(|a'_i| + |c_i| - |a_i|)$. And the second part can be computed as:

$$\begin{aligned}\Delta\Phi_{cut} &= \frac{1}{2}[|e(a'_i, V\backslash a'_i)| - |e(a'_i, c_i)| + |e(c_i, V\backslash a'_i)| + |e(a_i, V\backslash a_i)| \\ &\quad + |e(a_i, c_i)| - |e(c_i, V\backslash a_i)| - |e(a'_i, V\backslash a'_i)| - |e(a_i, V\backslash a_i)|] \\ &= \frac{1}{2}[|e(c_i, V\backslash a'_i)| - |e(a'_i, c_i)| + |e(a_i, c_i)| - |e(c_i, V\backslash a_i)|] \\ &= \frac{1}{2}[|e(c_i, a_i)| + |e(a_i, c_i)| - |e(c_i, a'_i)| - |e(a'_i, c_i)|] = \frac{1}{2}(\Delta E)\end{aligned} \tag{14}$$

Therefore, it can be concluded that $\Phi(a'_i, a_{-i}) - \Phi(a_i, a_{-i}) \equiv \varphi(a'_i, a_{-i}) - \varphi(a_i, a_{-i})$. Thus, Theorem 5 is proved.

Next, we answer 3 remained questions, a) how to choose the normalization factor λ; b) how fast the Nash equilibrium can be found; and c) how good is the derived solution.

C. Game Analysis

Normalization. We show how to choose the normalization factor λ. In the game process, the factors of load balancing and edge-cutting have confounded effects over the partitioning optimization. However, the cost of load balancing is counted in millions, and the cost of edge-cutting is counted in thousands, which may not faithfully reflect the weights of the two factors in the total cost. We thus show a strategy for determining λ.

Without losing generality, we assume the two factors in Equation 10 are of equal importance [19], [47], so that $\lambda\frac{1}{k}\sum_{i=1}^{k}|p_i|^2 = \sum_{i=1}^{k}|e(p_i, V\backslash p_i)|$. Therefore, we can have:

$$\lambda = \frac{k\sum_{i=1}^{k}|e(p_i, V\backslash p_i)|}{\sum_{i=1}^{k}|p_i|^2} \tag{15}$$

Then, we show the value range of λ, which is important in analyzing the quality of partitioning.

Theorem 6: The value range of λ is $[0, \frac{k^2\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|}{(\sum_{i=1}^{m}|c_i|)^2}]$.

Proof 6: When all clusters are assigned to the same partition, the load balancing factor reaches the maximum value $(\sum_{i=1}^{m}|c_i|)^2$, while edge-cutting factor gets the minimum value 0. Conversely, when all clusters are evenly separated to different partitions, the load balancing factor reaches the minimum value $\frac{(\sum_{i=1}^{m}|c_i|)^2}{k}$, while the edge-cutting factor gets the maximum value $\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|$. According to the valid variation range of the two factors, we can get $0 \le \lambda \le \frac{k^2\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|}{(\sum_{i=1}^{m}|c_i|)^2}$. Therefore, the theorem is proved.

We next analyze the round complexity and the quality of the game process. The round complexity of a game refers to the number of iterations, i.e., rounds of Algorithm 3, taken in finding the Nash equilibrium. A smaller number of rounds corresponds to a faster convergence to the equilibrium, and better efficiency. For the partitioning quality, we use PoA and PoS to measure the quantification of the suboptimality of finding an equilibrium in approaching the optimal solution. PoA (PoS) denotes the ratio of the worst (best) local optimal solution find by Algorithm 3 to the global optimal solution of Equation 10. It indicates the upper and lower bound of the Nash equilibrium, reflecting the stability of the game process.

Round Complexity. We next study the round complexity of Algorithm 3.

Theorem 7: The number of rounds of cluster partitioning game is bounded by $\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|$.

Proof 7: We next study how to prove the Theorem 7. Based on the Equations 13, 15 and the Theorem 6, we can have $0 \le \Phi(a)^{load} = \frac{1}{2}\sum_{i=1}^{k}|e(p_i, V\backslash p_i)| \le \frac{1}{2}\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|$ and $0 \le \Phi(a)^{cut} \le \frac{1}{2}\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|$. Hence, we can get the range of potential function $0 \le \Phi(a) = \Phi(a)^{load} + \Phi(a)^{cut} <= \sum_{i=1}^{m}|e(c_i, V\backslash c_i)|$.

Since $\Phi(a)^{cut} = \frac{1}{2}\sum_{i=1}^{k}|e(p_i, V\backslash p_i)|$ is in the integer domain. It implies that if a cluster changes its current strategy in the game, the reduction of $\Phi(a)$ should be at least 1. So, the number of rounds will be bounded by $\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|$.

Partitioning Game Quality. To theoretically bound the partitioning quality, we analyze the worst-case and best-case partitioning quality at the equilibrium, relative to the optimal performance, respectively. In algorithmic game theories, the two are called *Price of Anarchy* (PoA) and *Price of Stability* (PoS), which are counterparts to the concept of approximation ratio in algorithm designs.

Definition 5: **(Price of Anarchy)** PoA is the lowest ratio of Nash equilibrium achieved over the optimum value of the overall cost function. Generally, $PoA = \frac{\varphi(\Lambda)_{max}}{\varphi(\Lambda^{opt})}$, where Λ^{opt} denotes the global optimal strategy minimizing function $\varphi(\Lambda)$.

Theorem 8: The PoA of the game is bounded by $k + 1$.

Proof 8: If the load balancing and edge-cutting factors get their maximum values, simultaneously, the upper bound of $\varphi(\Lambda)$ can be computed as:

$$\begin{aligned}\varphi(\Lambda) &\le \frac{k\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|}{(\sum_{i=1}^{m}|c_i|)^2} \cdot \sum_{i=1}^{k}|p_i|^2 + \sum_{i=1}^{k}|e(p_i, V\backslash p_i)| \\ &\le (k+1)\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|\end{aligned} \tag{16}$$

Similarly, if the two factors get their minimum values, simultaneously, the lower bound of $\varphi(\Lambda^{opt})$ can be computed as:

$$\varphi(\Lambda^{opt}) \ge \frac{k\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|}{(\sum_{i=1}^{m}|c_i|)^2} \cdot \frac{(\sum_{i=1}^{m}|c_i|)^2}{k} = \sum_{i=1}^{m}|e(c_i, V\backslash c_i)| \tag{17}$$

PoA must be no larger than the quotient of upper bound of φ (Equation 16) and lower bound of $\varphi(\Lambda^{opt})$ (Equation 17). Thus, we have $PoA = \frac{\varphi(\Lambda)_{max}}{\varphi(\Lambda^{opt})} \le \frac{(k+1)\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|}{\sum_{i=1}^{m}|e(c_i, V\backslash c_i)|} = k + 1$. Therefore, the theorem is proved.

Definition 6: **(Price of Stability)** Price of Stability(PoS) is the highest ratio of Nash equilibrium over the optimum value of overall cost function. Generally, $PoS = \frac{\varphi(\Lambda')}{\varphi(\Lambda^{opt})}$, where

Λ' is the best Nash equilibrium strategy that minimize the potential function $\Phi(\Lambda)$.

Theorem 9: The PoA of the game is bounded by 2.

Proof 9: Based on Equations 11 and 13, for any partition strategy a, we can get that $\Phi(\Lambda) \leq \varphi(\Lambda) \leq 2\Phi(\Lambda)$. Since $\Phi(\Lambda') \leq \Phi(\Lambda^{opt})$, we can have $\varphi(\Lambda') \leq 2\Phi(\Lambda') \leq 2\Phi(\Lambda^{opt}) \leq 2\varphi(\Lambda^{opt})$. Thus, we can conclude that $PoS = \frac{\varphi(\Lambda')}{\varphi(\Lambda^{opt})} \leq 2$ Therefore, the theorem is proved.

So far, we analyze the feasibility of game-based cluster partitioning. Next, we proceed to discuss how to parallelize the cluster partitioning process.

D. Parallelization

The parallelization is enabled by clustering, which preserves the graph locality, so that two clusters tend to be adjacent in the graph structure, if their cluster IDs are close. For example, as shown in Figure 2 (b), if neighbors of vertex v arrive in the BFS order, vertex v_1 stands out to form a new cluster c_1 with its neighbors, so that c_1 and c_0 are structurally adjacent. Based on the observation, we divide clusters within a distributed node into batches according to cluster IDs, for being further parallelized by multi-threading. We recommend setting the batch size as a constant integer multiple of k, for dividing clusters equally into partitions. Otherwise, the solution space of the cluster partitioning problem can be enlarged, because one has to consider more possibilities for balancing, which increases the overhead of finding the Nash equilibrium.

Without parallelization, according to Algorithm 3, the time cost for each round of finding Nash equilibrium is $\Theta(m)$, and the round complexity is far less than $|E|$ (Theorem 7), so the average time complexity of cluster partitioning step is $\Theta(|E|)$. Additionally, the space cost of this step is $O(m)$. With parallelization, the average time complexity can be approximated as $\Theta(|E_b^{avg}| * \frac{m}{batchsize \times threads\_num})$, where $|E_b^{avg}|$ is the average number of the intra-cluster edges in each batch. The space cost of this step is $O(bathsize \times threads\_num)$. Each thread holds clusters of the batch size, which is far less than $O(|V|)$.

VI. Experiments

A. Experiment Setup

Datasets. We used four real web graphs, UK, Arabic, WebBase, and IT, as listed in Table IV. Although the scope of the manuscript is on web graphs, we also test the partitioning quality on real social graph Twitter.

TABLE IV
Details of real-world Web graphs

Alias	Source	$\|V\|$	$\|E\|$	Size
UK	uk-2002 [25]	19M	0.3B	4.7GB
Arabic	arabic-2005 [25]	22M	0.6B	11GB
WebBase	webbase-2001 [48], [49]	118M	1.0B	17.2GB
IT	it-2004 [48], [49]	41M	1.5B	18.8GB
Twitter	twitter [48], [49]	41M	1.4B	18.3GB

Competitors. We consider 5 competitors for evaluating the performance of vertex-cut partitioning, as shown in Table I, where HDRF is considered as the state-of-the-art vertex-cut streaming partitioning algorithm[5]. For a fair comparison, we choose default settings and best streaming orders for each of the competitors, a.k.a., random orders for HDRF, Greedy, Hash, and DBH, and BFS orders for Mint and CLUGP. The default parameters of CLUGP are set as follows. The maximum cluster volume V_{max} is set as $\frac{|E|}{k}$ according to the suggestion of [39], the imbalance factor $\tau = 1.0$, batch size is set as 6400 and the number of partitioning threads are set to 32. For cluster partitioning game, the normalization factor λ is set to its maximum value.

Metrics. We use the replication factor and relative load balance, which are commonly accepted to measure the partitioning quality. Details are shown in Section II-B.

Environment. All algorithms are implemented in Java and run a PC with 20 x Intel(R) Xeon(R) CPU $E5\text{-}2698v4$ @ 2.20GHz 40 cores and 256GB main memory. To test the partitioning quality on real distributed environment, we use docker to simulate 32 computing nodes equipped with PowerGraph [2], and allocate one CPU for each computing node.

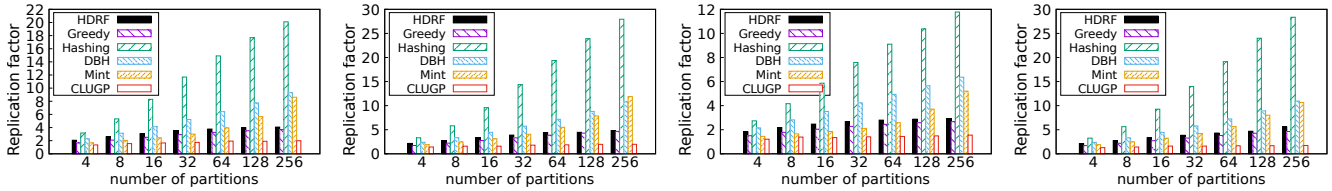
B. Results

Replication factor. We show the results of quality on 4 real datasets in Figure 3. In all testings, CLUGP outperforms its competitors and the trend of CLUGP is relatively stable. For example, in Figure 3 (b), by increasing the number of partitions from 4 to 256, the replication factor of CLUGP increases only about 1.5 times, while Hashing increases about 10 times. When the number of partitions equals 256, the replication factor of CLUGP is only 1/2 of that of HDRF, the best partitioning baseline. More, the second and third best competitors, Greedy and Mint, fall far behind CLUGP in terms of scalability, as it will be shown later.

The lowest replication factors of CLUGP shows the effectiveness of our proposal. The reasons are threefold. 1) For the stream clustering step, the splitting operation helps in reducing the replication factor. 2) For the cluster partitioning step, the cost function is designed to minimize edge-cutting and control the replication factor. 3) For the partition transformation step, the transformation fine-tunes the cluster partitioning result.

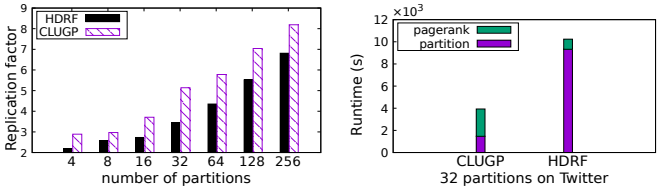
We also present the results on social graphs, e.g., Twitter, in Figure 4. It shows that the replication factor of CLUGP is slightly higher than that of HDRF. But the total task runtime cost, including graph partitioning time and distributed algorithm (e.g, pagerank) execution time, of CLUGP is much lower, because of the partitioning efficiency of CLUGP dominates that of HDRF. We would like to point out that our framework is modularity-based and is targeted on web graphs, instead of social graphs. The modularity of social graphs is often much lower than web graphs, as shown in Table V, making the quality of streaming clustering not as good as web graphs.

[5]The source codes of Hashing, DBH, HDRF, Greedy are provided by the first author of HDRF paper (https://github.com/fabiopetroni/VGP). The source code of Mint is obtained by the first author of Mint upon personal request.



(a) RF vs. #. of Partitions (UK-2002) (b) RF vs. #. of Partitions (Arabic-2005) (c) RF vs. #. of Partitions (WebBase-2001) (d) RF vs. #. of Partitions (IT-2004)

Fig. 3. Results on Quality (Replication Factor)



(a) RF vs. #. of Partitions (b) Runtime Cost

Fig. 4. Results on Twitter

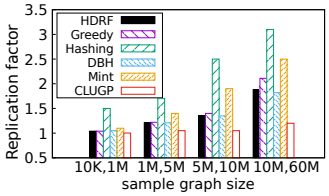


Fig. 5. Results on Sample Graph

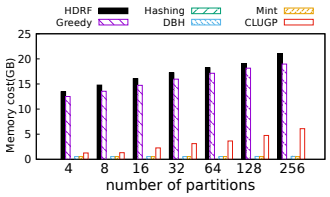


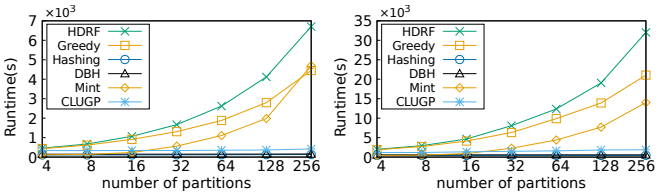
Fig. 6. Space vs. #. of Partitions (IT-2004)

(R1.O2) We test the performance of CLUGP w.r.t. varied graph sizes. We randomly sample UK-2002 to create a series of graph datasets. Figure 5 shows that CLUGP has the best partitioning quality. By varying the graph size from $10K$ to $60M$, the replication factor of CLUGP increases only 20%, while HDRF increases about 80%.

TABLE V
MODULARITY OF SOCIAL GRAPHS

Algorithms	Datasets	Modularity
Improved MOH [36], [37]	twitter-2010	0.47
GPDGP [50]	soc-Pokec	0.63
	soc-LiveJournal1	0.72
	wiki-Talk	0.56

Load balance. As for the relative load balance, all algorithms achieve 1.0. We also analyze the influence of relative load balance on replication factor in Figure 11 (a). It shows the replication factor slightly decreases as the increase of relative load balance. In all testings, the performance of CLUGP is stable.



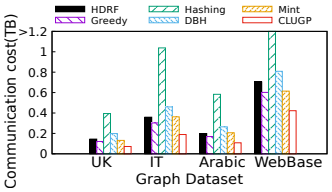
(a) Time vs. #. of Partitions (UK-2002) (b) Time vs. #. of Partitions (IT-2004)

Fig. 7. Scalability in Terms of Time

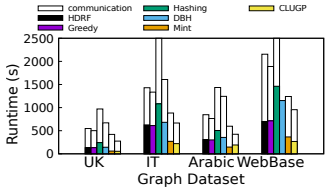
Space Overhead. We measure CLUGP's scalability in terms of space cost against other methods, as shown in Figure 6. Heuristic-based methods occupy the biggest amount of space, which is about 8 to 10 times higher than CLUGP. Because heuristic-based methods need to maintain the information from all partitions for the optimization purpose. By contrast, hashing-based methods take the minimum amount of space. Especially, Hashing takes 0 space cost, because it merely needs a hash function for making the partition decision. CLUGP takes larger space cost than Mint, since the space complexity of Mint is $O(batch\_size * number\_of\_threads)$, and CLUGP is $O(2|V|)$, as is mentioned in Section III, Section IV and Section V. Although CLUGP takes bigger space than hashing-based methods and Mint, we argue that the space of several gigabytes are totally affordable for a cluster of hundreds of computing nodes. It is also worthy of the cost for the gaining in replication factor (Figure 3), partitioning efficiency (Figure 7), and the computing efficiency (Figure 8).

Runtime Scalability. We compare the scalability of all methods in terms of runtime cost, in Figure 7 (a-b). The time costs of heuristic-based methods and Mint increase significantly as the increase of the number of partitions. In particular, when the number of partitions equals 256 (Figure 7 (b)), HDRF takes about $35,000$ seconds for fulfilling the task of graph partitioning. By contrast, CLUGP and hashing-based methods are not sensitive to the number of partitions. For example, when the number of partitions increase from 4 to 256, the runtime cost increases only from $1,162$ to $1,869$ seconds. In all testing, the runtime cost of CLUGP is about 2 to 3 times of that of hashing-based methods. We argue that the cost of CLUGP is worthwhile, since the small amount of runtime cost brings in the great benefits of partitioning quality (up to $10\times$ decrease of replication factor in Figure 3 (d)).

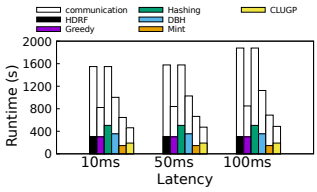
Performance on Real Systems. We examine the performance of partitioning algorithms on real distributed graph systems, PowerGraph. We report the computation and communication cost on pagerank, in Figure 8. In all testings, CLUGP has the lowest computing time and communication time. The excellent performance is due to the high partitioning quality of CLUGP, including load balancing and low replication factor. In general, hashing-based method perform the worst, and the performance gap is increasing w.r.t. the data volumes.



(a) Communication cost vs. Dataset



(b) Runtime Cost vs. Dataset



(c) Runtime Cost vs. Latency

Fig. 8. Results on PageRank (PowerGraph)

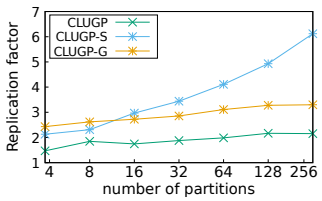
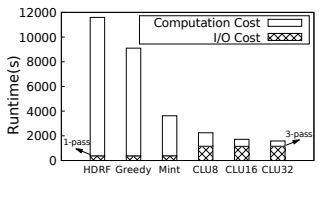
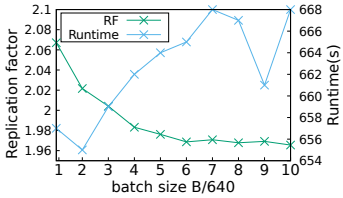


Fig. 9. Ablation Study

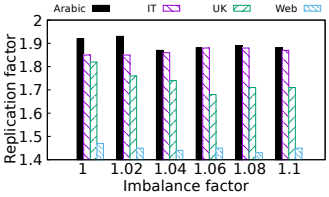


(a) Runtime Cost vs. Algorithms

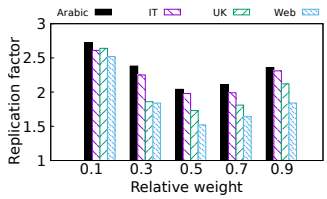


(b) Effect of Batch Size

Fig. 10. Parallelization



(a) Effect of imbalance factor



(b) Effect of relative weight

Fig. 11. Analysis

Heuristic-based methods and Mint are close, but still about 50% to 100% higher than CLUGP. In particular, on IT, CLUGP takes about 40% of communication cost (Figure 8 (a)), and about 50% computation cost (Figure 8 (b)), of the second best method, Greedy. To simulate real networking latency, we use PUMBA[6] to vary the RTT from 10ms to 100ms. The running time of pagerank under different network latency is shown in Figure 8 (c). In all testings, CLUGP is the most efficient and the stablest method.

Ablation Study. Splitting operation is the core part of stream clustering. Therefore, we compare the results with and without splitting operation, denoted as CLUGP and CLUGP-S, respectively. The experiment is done on IT by varying the number of partitions from 4 to 256. As shown in Figure 9, RF of CLUGP is lower than CLUGP-S, in all cases. The trend of RF of CLUGP is relatively stable, while the RF of CLUGP-S increases sharply. So, we conclude that the splitting operation significantly improves the partitioning quality. Also, we compare the results with and without game theory-based cluster partitioning, denoted as CLUGP and CLUGP-G, respectively, to show its effectiveness. CLUGP-G greedily assigns a big-sized cluster to a small-sized partition. The result is depicted in Figure 9. The replication factor of CLUGP is about 60-70% lower than CLUGP-S, demonstrating the effectiveness of game theory-based cluster partitioning.

Parallelization. We evaluate the parallelization performance by varying the number of threads, in Figure 10 (a). Notice that the total runtime cost consists of I/O cost and computation cost, whereas the latter dominates the total cost. Furthermore, since streaming cluster (step 1) and partitioning transformation (step 3) are all constant time complexity, the cluster partitioning game (step 2) almost occupies all the computation time. Compared to one-pass streaming partitioning algorithms, e.g., HDRF, Greedy, and Mint (with 32 threads), the total runtime cost of CLUGP is much less. In particular, the runtime cost of CLUGP is about 60% less than that of the second best competitor, Mint, although the I/O cost of our three-pass streaming partitioning algorithm is three times of that of one-pass competitors. Also, it shows that, when the number of threads is increased from 8 to 32, the computation cost of CLUGP decreases from 1091 to 429 seconds, demonstrating good acceleration ratio of our parallelization mechanism. In particular, the runtime cost of CLUGP (with only 8 threads) is about 45% lower than that of Mint (with 32 threads). Besides, we test the impact of batch size. In Figure 10 (b), it shows that the runtime cost is insensitive to the batch size. With the increasing of batch size, the running time of CLUGP increases slightly. When increasing the batch size, although time cost of cluster partitioning game within a batch will be increased, the number of partitioning tasks decreased.

Relative Weight. Similar to previous works, we treat the two partitioning metrics in Section II-B as equally important and set the relative weight of Equation 2 to 0.5. We next study the influence of relative weight on partitioning quality, the result is shown in Figure 11 (b). We can have two observations: 1) in all testings, the replication factor of CLUGP is lower than its competitors; 2) the curve of replication factor of CLUGP is U-shaped with a wide and smooth valley. The replication factor is high for two extremes. When the relative weight equals 0.1, the optimization target is mostly on the replication factor, so that clusters are mostly put to very few partitions, which is almost equivalent to skipping the game process, resulting in a high replication factor. When the relative weight equals 0.9, the optimization target is mostly on the load balance, so clusters tend to be sent evenly to the set of partitions. For other valued weights (i.e., the relative weight is in $[0.3, 0.7]$), the variation of replication factor is mostly within 10%. We can conclude that the relative weights do not have significant effect on partitioning quality, except extreme cases.

[6]https://github.com/alexei-led/pumba

VII. Related Work

There exists many algorithms for edge-cut and vertex-cut partitioning. Edge-cut partitioning aims to assign vertices into different partitions, while minimizing edge-cutting. METIS [9] is an offline algorithm that adopts multi-level heuristics achieving high partitioning quality for edge-cut partitioning. However, efficiency of offline partitioning is low. Streaming partitioning is considered to be practical for large-scale graph processing [2], [6], [51]. LDG [51] tends to assigning neighboring vertices into the same partition. FENNEL [6] is an edge-cut partitioning algorithm which places a new vertex to the partition holding the most neighboring vertices or holding the least non-neighboring vertices. The vertex-cut streaming partitioning is first proposed in [2], and has been proved to be effective on power-law graphs. Greedy [2] is a heuristic-based partitioning strategy which aims to minimize vertex-cuts. HDRF [12] makes use of the skewed distribution of degrees, and cuts the high-degree vertices first to reduce replicas. Similarly, DBH [14] prioritizes the cutting of high-degree vertices, with hashing-based methods. Both Greedy and HDRF need to record the previous partitioning results, which are hard to be parallelized. Mint [19] is a parallel algorithm that achieves a good trade-off between scalability and partitioning quality. Different from previous works, we explore modularity-based clustering for enhancing the partitioning quality, employ streaming techniques for improving the efficiency, and break the ties of global structures for boosting system performance.

VIII. Conclusion

In this paper, we study the problem of edge-centric partitioning for web graphs by proposing a novel restreaming architecture, called CLUGP. Of the architecture, our techniques can be pipelined as three steps, streaming clustering, cluster partitioning, and transformation. Compared with state-of-the-art algorithms, CLUGP achieves the best partitioning quality. Also, we investigate parallelization mechanism to enhance the partitioning scalability. The results on real datasets and real distributed graph systems show that the scalability of CLUGP is significantly better than that of one-pass streaming partitioning methods.

References

[1] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference 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 *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, 2012, pp. 17–30.

[3] 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 *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, 2014, pp. 599–613.

[4] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning in the cloud," *arXiv preprint arXiv:1204.6078*, 2012.

[5] R. Chen, J. Shi, Y. Chen, B. Zang, H. Guan, and H. Chen, "Powerlyra: Differentiated graph computation and partitioning on skewed graphs," *ACM Transactions on Parallel Computing (TOPC)*, vol. 5, no. 3, pp. 1–39, 2019.

[6] 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*, 2014, pp. 333–342.

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

[8] R. Krauthgamer, J. Naor, and R. Schwartz, "Partitioning graphs into balanced components," in *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2009, pp. 942–949.

[9] G. Karypis and V. Kumar, "Parallel multilevel graph partitioning," in *Proceedings of international conference on parallel processing*. IEEE, 1996, pp. 314–319.

[10] G. M. Slota, S. Rajamanickam, and K. Madduri, "Pulp/xtrapulp: Partitioning tools for extreme-scale graphs," Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2017.

[11] J. U. Joel Nishimura, "Restreaming graph partitioning: simple versatile algorithms for advanced balancing," in *KDD*, 2013, pp. 1106–1114.

[12] F. Petroni, L. Querzoni, K. Daudjee, S. Kamali, and G. Iacoboni, "Hdrf: Stream-based partitioning for power-law graphs," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015, pp. 243–252.

[13] C. Zhang, F. Wei, Q. Liu, Z. G. Tang, and Z. Li, "Graph edge partitioning via neighborhood heuristic," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 605–614.

[14] C. Xie, L. Yan, W.-J. Li, and Z. Zhang, "Distributed power-law graph computing: Theoretical and empirical analysis," *Advances in neural information processing systems*, vol. 27, pp. 1673–1681, 2014.

[15] D. Margo and M. Seltzer, "A scalable distributed graph partitioner," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1478–1489, 2015.

[16] R. Albert, H. Jeong, and A.-L. Barabási, "Error and attack tolerance of complex networks," *Nature*, vol. 406, no. 6794, pp. 378–382, 2000.

[17] D. Donato, L. Laura, S. Leonardi, and S. Millozzi, "Large scale properties of the webgraph," *The European Physical Journal B*, vol. 38, no. 2, pp. 239–243, 2004.

[18] M. A. K. Patwary, S. Garg, and B. Kang, "Window-based streaming graph partitioning algorithm," in *Proceedings of the Australasian Computer Science Week Multiconference*, 2019, pp. 1–10.

[19] Q.-S. Hua, Y. Li, D. Yu, and H. Jin, "Quasi-streaming graph partitioning: A game theoretical approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1643–1656, 2019.

[20] Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov, "Streaming graph partitioning: an experimental study," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1590–1603, 2018.

[21] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[22] J. Reichardt and S. Bornholdt, "Partitioning and modularity of graphs with arbitrary degree distribution," *Physical Review E*, vol. 76, no. 1, p. 015102, 2007.

[23] G. Agarwal and D. Kempe, "Modularity-maximizing graph communities via mathematical programming," *The European Physical Journal B*, vol. 66, no. 3, pp. 409–418, 2008.

[24] W. Yang, G. Wang, M. Z. A. Bhuiyan, and K.-K. R. Choo, "Hypergraph partitioning for social networks based on information entropy modularity," *Journal of Network and Computer Applications*, vol. 86, pp. 59–71, 2017.

[25] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed web crawler," *Software: Practice & Experience*, vol. 34, no. 8, pp. 711–726, 2004.

[26] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 301–316.

[27] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the web for emerging cyber-communities," *Computer networks*, vol. 31, no. 11-16, pp. 1481–1493, 1999.

[28] ——, "Extracting large-scale knowledge bases from the web," in *VLDB*, vol. 99. Citeseer, 1999, pp. 639–650.

[29] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins, "The web as a graph: Measurements, models, and methods," in *International Computing and Combinatorics Conference*. Springer, 1999, pp. 1–17.

[30] R. Albert, H. Jeong, and A.-L. Barabási, "Diameter of the world-wide web," *nature*, vol. 401, no. 6749, pp. 130–131, 1999.

[31] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[32] A.-L. Barabási, R. Albert, and H. Jeong, "Scale-free characteristics of random networks: the topology of the world-wide web," *Physica A: statistical mechanics and its applications*, vol. 281, no. 1-4, pp. 69–77, 2000.

[33] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[34] L. O. Prokhorenkova, P. Prałat, and A. Raigorodskii, "Modularity of complex networks models," in *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 2016, pp. 115–126.

[35] H. Shiokawa, Y. Fujiwara, and M. Onizuka, "Fast algorithm for modularity-based graph clustering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 27, no. 1, 2013.

[36] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[37] A. Panyala, O. Subasi, M. Halappanavar, A. Kalyanaraman, D. Chavarria-Miranda, and S. Krishnamoorthy, "Approximate computing techniques for iterative graph algorithms," in *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*. IEEE, 2017, pp. 23–32.

[38] D. Aloise, G. Caporossi, P. Hansen, L. Liberti, S. Perron, and M. Ruiz, "Modularity maximization in networks by variable neighborhood search." *Graph Partitioning and Graph Clustering*, vol. 588, p. 113, 2012.

[39] A. Hollocou, J. Maudet, T. Bonald, and M. Lelarge, "A streaming algorithm for graph clustering," *arXiv preprint arXiv:1712.04337*, 2017.

[40] F. Moons, "Game theory: Distributed selfish load balancing on networks," 2013.

[41] B. Vöcking, "Selfish load balancing," *Algorithmic game theory*, vol. 20, pp. 517–542, 2007.

[42] C. Mayer, R. Mayer, M. A. Tariq, H. Geppert, L. Laich, L. Rieger, and K. Rothermel, "Adwise: Adaptive window-based streaming edge partitioning for high-speed graph processing," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 685–695.

[43] A. Noack and R. Rotta, "Multi-level algorithms for modularity clustering," in *International symposium on experimental algorithms*. Springer, 2009, pp. 257–268.

[44] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin, "Breakdown of the internet under intentional attack," *Physical review letters*, vol. 86, no. 16, p. 3682, 2001.

[45] J. F. Nash *et al.*, "Equilibrium points in n-person games," *Proceedings of the national academy of sciences*, vol. 36, no. 1, pp. 48–49, 1950.

[46] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[47] N. Armenatzoglou, H. Pham, V. Ntranos, D. Papadias, and C. Shahabi, "Real-time multi-criteria social graph partitioning: A game theoretic approach," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 1617–1628.

[48] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*. Manhattan, USA: ACM Press, 2004, pp. 595–601.

[49] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proceedings of the 20th international conference on World Wide Web*, S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, Eds. ACM Press, 2011, pp. 587–596.

[50] M. Onizuka, T. Fujimori, and H. Shiokawa, "Graph partitioning for distributed graph processing," *Data Science and Engineering*, vol. 2, no. 1, pp. 94–105, 2017.

[51] I. Stanton and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 1222–1230.