HDRF: Stream-Based Partitioning for Power-Law Graphs*

Fabio Petroni
Department of Computer Control and Management Engineering Antonio Ruberti
Sapienza University of Rome
petroni@dis.uniroma1.it

Leonardo Querzoni
Department of Computer Control and Management Engineering Antonio Ruberti
Sapienza University of Rome
querzoni@dis.uniroma1.it

Khuzaima Daudjee
David R. Cheriton School of Computer Science
University of Waterloo
kdaudjee@uwaterloo.ca

Shahin Kamali
David R. Cheriton School of Computer Science
University of Waterloo
s3kamali@uwaterloo.ca

Giorgio Iacoboni
Department of Computer Control and Management Engineering Antonio Ruberti
Sapienza University of Rome
g.iacoboni@gmail.com

ABSTRACT

Balanced graph partitioning is a fundamental problem that is receiving growing attention with the emergence of distributed graph-computing (DGC) frameworks. In these frameworks, the partitioning strategy plays an important role since it drives the communication cost and the workload balance among computing nodes, thereby affecting system performance. However, existing solutions only partially exploit a key characteristic of natural graphs commonly found in the real-world: their highly skewed power-law degree distributions. In this paper, we propose High-Degree (are) Replicated First (*HDRF*), a novel streaming vertex-cut graph partitioning algorithm that effectively exploits skewed degree distributions by explicitly taking into account vertex degree in the placement decision. We analytically and experimentally evaluate *HDRF* on both synthetic and real-world graphs and show that it outperforms all existing algorithms in partitioning quality.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Graphs and Networks

Keywords

Graph Partitioning; Streaming Algorithms; Distributed Graph-Computing Frameworks; Replication; Load Balancing.

*This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research.

CIKM'15, October 19-23, 2015, Melbourne, VIC, Australia.
 ISBN 978-1-4503-3794-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2806416.2806424.

1. INTRODUCTION

The last few years have witnessed a huge growth in information production. Some corporations like IBM estimate that "2.5 quintillion bytes of data are created every day", amounting to 90% of the data in the world today having been created in the last two years [10]. On the face of this growth, researchers from both academia and industry have focussed their efforts on the design of new, efficient, approaches for parallel data analysis able to withstand the deluge of data expected in forthcoming years.

Given the proliferation of data which can be represented as graphs of interconnected vertices, a graph-based computation paradigm provides a nice, suitable, abstraction to perform computation on it. Large amounts of data, particularly scale-free graphs or power-law graphs[1], fall within this paradigm. An example is recommendation systems where the input data is usually provided in the form of votes (edges) that users (vertices) express on products (vertices). Additionally, graph-based computation finds application in many diverse and important fields such as social networks, computational biology, chemistry, and computer security. A key problem in graph computation is that it is often difficult to scale with increasing input data sizes as graphs are not easily partitionable into independent subgraphs that can be computed in parallel.

To be able to work on large datasets, distributed graph-computing (DGC) frameworks (such as GraphLab [18] or Pregel [20]) forcibly partition the input graph by placing its constituting elements, be they either vertices or edges, in distinct partitions, one for each available computing resource. During the partitioning phase, data elements that share connections with other elements already placed in other partitions result in having remote connections amongst them. Since these partitions are usually placed on different machines, this can incur unnecessary or excessive network and computation costs. To address this issue, one frequently used technique is to create and locally place replicas of remotely connected data among these partitions. While this reduces the access cost, replicated data elements must be synchronized during computation so as to avoid replica states

[1]We use scale-free and power-law graphs synonymously.

from diverging and generating meaningless computation results. This synchronization can significantly hinder performance as it forces replicas to coordinate and exchange data several times during computation.

The way the input dataset is partitioned has a large impact on the performance of the graph computation. A naive partitioning strategy may end up replicating a large fraction of the input elements on several partitions, severely hampering performance by inducing a large replica synchronization overhead during the computation phase. Furthermore, the partitioning phase should produce evenly balanced partitions (i.e. partitions with similar sizes) to avoid possible load skews in a cluster of machines over which the data is partitioned. Several recent approaches have looked at this problem. Here we focus our attention on *stream-based* graph partitioning algorithms, i.e. algorithms that partition incoming elements one at a time on the basis of only the current element properties and on previous assignments to partitions (no global knowledge on the input graph). Furthermore, these algorithms are usually *one-pass*, i.e. they refrain from changing the assignment of a data element to a partition once this has been done. These algorithms are the ideal candidates in settings where input data size and constraints on available resources restrict the type of solutions that can be employed.

Other characteristics of input data also play an important role in partitioning. It has been shown that vertex-cut algorithms are the best approach to deal with input graphs characterized by power-law degree distributions [1, 12]. This previous work also clearly outlined the important role high-degree nodes play from a partitioning quality standpoint. Nevertheless, few algorithms take this aspect into account [27, 24]. Understandably, this is a challenging problem to solve for stream-based approaches due to their *one-pass* nature.

In this paper, we leverage the idea that a partitioning algorithm should do its best to cut, i.e., *replicate*, high-degree vertices. In particular, we introduce High Degree (are) Replicated First (*HDRF*), a stream-based graph partitioning algorithm based on a greedy vertex-cut approach that leverages information on vertex degrees.
HDRF is characterized by the following desirable properties: (i) it outputs partitions with the smallest average replication factor among all competing solutions when applied on power-law graphs (Figure 2) while (ii) providing close to optimal load balancing (Figure 3). The former is obtained by greedily replicating vertices with larger degrees, while the latter is provided by a parametrizable balancing term whose impact can be tuned to adapt the algorithm behavior to any data input order. On the one hand, lowering the average replication factor is important to reduce network bandwidth cost, memory usage and replica synchronization overhead at computation time. A fair distribution of load on partitions, on the other hand, allows a more efficient usage of available computing resources. HDRF takes into account both of these aspects in an integrated way, significantly reducing the time needed to perform computations on large-scale graphs.

Summing up, this paper provides the following contributions:

- a novel stream-based graph partitioning algorithm, namely *HDRF*, that performs better than any competing solution (i.e. processes less vertex-cuts while balancing the load) when applied on power-law graphs;
- a theoretical analysis of *HDRF* that provides an average-case upper bound for the vertex replication factor;
- a comprehensive experimental evaluation based both on simulations and on a working prototype integrated with GraphLab [19] that shows how a system using *HDRF* achieves up to 2× speedup than adopting a standard greedy placement, and close to 3× speedup than using a constrained solution.

The rest of this paper is organized as follows: we define the problem in Section 2; we briefly describe existing solutions in Section 3; we introduce *HDRF* in Section 4; we show theoretical bounds for *HDRF* in Section 5; we present the results of an extensive experimental evaluation in Section 6 and we conclude the paper in Section 7.

2. PROBLEM DEFINITION

The problem of optimally partitioning a graph to minimize vertex-cuts while maintaining load balance is a fundamental problem in parallel and distributed applications as input placement significantly affects the efficiency of algorithm execution [25]. An edge-cut partitioning scheme results in partitions that are vertex disjoint while a vertex-cut approach results in partitions that are edge disjoint. Both variants are known to be NP-Hard [16, 11, 2] but have different characteristics and difficulties [16]; for instance, one fundamental difference between the two is that a vertex can be cut in multiple ways and span several partitions while an edge can only connect two partitions.

One characteristic observed in real-world graphs from social networks or the Web is their skewed power-law degree distribution: most vertices have relatively few connections while a few vertices have many. It has been shown that vertex-cut techniques perform better than edge-cut ones on such graphs (i.e., create less storage and network overhead) [12]. For this reason modern graph parallel processing frameworks, like GraphLab [19], adopt a vertex-cut approach to partition the input data over a cluster of computing nodes. The focus of this paper is on streaming vertex-cut partitioning schemes able to efficiently handle graphs with skewed power-law degree distribution.

Notation — Consider a graph $G = (V, E)$, where $V = (v_1, \cdots, v_n)$ is the set of vertices and $E = (e_1, \cdots, e_m)$ the set of edges. We define a partition of edges $P = (p_1, .., p_k)$ to be a family of pairwise disjoint sets of edges (i.e. $p_i, p_j \subseteq E$, $p_i \cap p_j = \emptyset$ for every $i \neq j$). Let $A(v) \subseteq P$ be the set of partitions each vertex $v \in V$ is replicated. The size $|p|$ of each partition $p \in P$ is defined as its edge cardinality, because computation steps are usually associated with edges. Since we consider G having a power-law degree distribution, the probability that a vertex has degree d is $P(d) \propto d^{-\alpha}$, where α is a positive constant that controls the "skewness" of the degree distribution, i.e. the smaller the value of α, the more skewed the distribution.

Balanced k-way vertex-cut problem — The problem consists in defining a partition of edges such that (i) the average number of vertex replicas (i.e. the number of partitions each vertex is associated to as a consequence of edge partitioning) is minimized and (ii) the partition load (i.e. the number of edges associated to a partition) is within a given bound from the theoretical optimum (i.e. $|E|/|P|$) [2].

More formally, the balanced $|P|$-way vertex-cut partitioning problem aims at solving the following optimization problem:

$$\min \frac{1}{|V|} \sum_{v \in V} |A(v)| \quad s.t. \quad \max_{p \in P} |p| < \sigma \frac{|E|}{|P|} \tag{1}$$

where $\sigma \geq 1$ is a small constant that defines the system tolerance to load imbalance. The objective function (Equation (1)) is called *replication factor* (*RF*), which is the average number of replicas per vertex.

Streaming setting — Without loss of generality, here we assume that the input data is a list of edges, each identified by the two connecting vertices and characterized by some application-related data. We consider algorithms that consume this list in a streaming fashion, requiring only a single pass. This is a common choice for several reasons: (i) it handles situations in which the input data is large enough that fitting it completely in the main memory of a single computing node is impractical; (ii) it can efficiently process dynamic graphs; (iii) it imposes the minimum overhead in time and (iv) it's scalable, providing for straightforward parallel and distributed implementations. A limitation of this approach is that the assignment decision taken on an input element (i.e., an edge) can be based only on previously analyzed data and cannot be later changed.

3. STREAMING ALGORITHMS

Balanced graph partitioning is a well known NP-hard problem with a wide range of applications in different domains. We do not discuss offline and edge-cut partitioning techniques since they are out of the scope of the paper. It is possible to divide existing streaming vertex-cut partitioning techniques in two main families: hashing and constrained partitioning algorithms and greedy partitioning algorithms.

Hashing and constrained partitioning algorithms — All of these algorithms ignore the history of the edge assignments and rely on the presence of a predefined hash function $h : \mathbb{N} \to \mathbb{N}$. The input of the hash function h can be either the unique identifier of a vertex or of an edge. All these algorithms can be applied in a streaming setting and achieve good load balance if h guarantees uniformity. Four well-known existing heuristics to solve the partitioning problem belong to this family: *hashing*, *DBH*, *grid* and *PDS*. The simplest solution is given by the *hashing* technique that (pseudo-)randomly assigns each edge to a partition: for each input edge $e \in E$, $A(e) = h(e) \bmod |P|$ is the identifier of the target partition. This heuristic results in a large number of vertex-cuts in general and performs poorly on power-law graphs [12]. A recent paper describes the *Degree-Based Hashing* (*DBH*) algorithm [27], a variation of the *hashing* heuristic that explicitly considers the degree of the vertices for the placement decision. *DBH* leverages some of the same intuition as *HDRF* by cutting vertices with higher degrees to obtain better performance. Concretely, when processing edge $e \in E$ connecting vertices $v_i, v_j \in V$ with degrees d_i and d_j, *DBH* defines the hash function $h(e)$ as follows:

$$h(e) = \begin{cases} h(v_i), & \text{if } d_i < d_j \\ h(v_j), & \text{otherwise} \end{cases}$$

Then, it operates as the *hashing* algorithm.

The *grid* and *PDS* techniques belong to the *constrained partitioning* family of algorithms [14]. The general idea of these solutions is to allow each vertex $v \in V$ to be replicated only in a small subset of partitions $S(v) \subset P$ that is called the *constrained set* of v. The constrained set must guarantees some properties; in particular, for each $v_i, v_j \in V$: (i) $S(v_i) \cap S(v_j) \neq \emptyset$; (ii) $S(v_i) \not\subseteq S(v_j)$ and $S(v_j) \not\subseteq S(v_i)$; (iii) $|S(v_i)| = |S(v_j)|$. It is easy to observe that this approach naturally imposes an upper bound on the replication factor. To position a new edge e connecting vertices v_i and v_j, it picks a partition from the intersection between $S(v_i)$ and $S(v_j)$ either randomly or by choosing the least loaded one. Different solutions differ in the composition of the vertex constrained sets. The *grid* solution arranges partitions in a $X \times Y$ matrix such that $|P| = XY$. It maps each vertex v to a matrix cell using a hash function h, then $S(v)$ is the set of all the partitions in the corresponding row and column. It this way each constrained sets pair has at least two partitions in their intersection. *PDS* generates constrained sets using *Perfect Difference Sets* [13]. This ensure that each pair of constrained sets has exactly one partition in the intersection. *PDS* can be applied only if $|P| = x^2 + x + 1$, where x is a prime number.

Greedy partitioning algorithms — This family of methods uses the entire history of the edge assignments to make the next decision. The standard *greedy* approach [12] breaks the randomness of the hashing and constrained solutions by maintaining some global status information. In particular, the system stores the set of partitions $A(v)$ to which each already observed vertex v has been assigned and the current partition sizes. Concretely, when processing edge $e \in E$ connecting vertices $v_i, v_j \in V$, the *greedy* technique follows this simple set of rules:

Case 1: If neither v_i nor v_j have been assigned to a partition, then e is placed in the partition with the smallest size in P.

Case 2: If only one of the two vertices has been already assigned (without loss of generality assume that v_i is the assigned vertex) then e is placed in the partition with the smallest size in $A(v_i)$.

Case 3: If $A(v_i) \cap A(v_j) \neq \emptyset$, then edge e is placed in the partition with the smallest size in $A(v_i) \cap A(v_j)$.

Case 4: If $A(v_i) \neq \emptyset$, $A(v_j) \neq \emptyset$ and $A(v_i) \cap A(v_j) = \emptyset$, then e is placed in the partition with the smallest size in $A(v_i) \cup A(v_j)$ and a new vertex replica is created accordingly.

Symmetry is broken with random choices. An equivalent formulation consists of computing a score $C^{\text{greedy}}(v_i, v_j, p)$ for all partitions $p \in P$, and then assigning e to the partition p^* that maximizes C^{greedy}. The score consists of two elements: (i) a replication term $C^{\text{greedy}}_{\text{REP}}(v_i, v_j, p)$ and (ii) a balance term $C^{\text{greedy}}_{\text{BAL}}(p)$. It is defined as follows:

$$C^{\text{greedy}}(v_i, v_j, p) = C^{\text{greedy}}_{\text{REP}}(v_i, v_j, p) + C^{\text{greedy}}_{\text{BAL}}(p) \tag{2}$$

$$C^{\text{greedy}}_{\text{REP}}(v_i, v_j, p) = f(v_i, p) + f(v_j, p) \tag{3}$$

$$f(v, p) = \begin{cases} 1, & \text{if } p \in A(v) \\ 0, & \text{otherwise} \end{cases}$$

$$C^{\text{greedy}}_{\text{BAL}}(p) = \frac{\text{maxsize} - |p|}{\epsilon + \text{maxsize} - \text{minsize}} \tag{4}$$

where *maxsize* is the maximum partition size, *minsize* is the minimum partition size, and ϵ is a small constant value.

A recent paper [24] proposes an hybrid solution that tries to combine both edge-cut and vertex-cut approaches together. The resulting heuristic, called *Ginger*, aims at optimizing the partitioning in a DGC framework. However, *Ginger* is not a streaming solution, since it needs extra reassignment phases after the original streaming graph partitioning.

We remark there are other facets of graph partitioning that may affect performance of a DGC framework and have been addressed in other works. For example, some applications are based on dynamic graphs and provided a hashing-based partitioning solution to manage such type of input [21]. Another aspect is the use of other metrics for optimization. For example [28] proposes a solution aimed at aggressively replicating vertices to improve the performance of queries on the graph and to keep them local to each single partition as much as possible. Further contributions along these lines are orthogonal and out of the scope of this paper.

4. THE HDRF ALGORITHM

In this section, we present *HDRF*, a greedy algorithm tailored for skewed power-law graphs.

In the context of robustness to network failure, Cohen et al. [7, 8] and Callaway et al [6] have analytically shown that if only a few high-degree vertices (hubs) are removed from a power-law graph then it is turned into a set of isolated clusters. Moreover, in power-law graphs, the clustering coefficient distribution decreases with increase in the vertex degree [9]. This implies that low-degree vertices often belong to very dense sub-graphs and those sub-graphs are connected to each other through high-degree vertices.

Our partitioning scheme leverages these properties by focusing on the locality of low-degree vertices. In particular, it tries to place each strongly connected component with low-degree vertices into a single partition by cutting high-degree vertices and replicating them on a large number of partitions. As the number of high-degree vertices in power-law graphs is very low, encouraging replication for only these vertices leads to an overall reduction of the replication factor.

Concretely, when *HDRF* creates a replica, it does so for the vertex with the highest degree. However, obtaining degrees of vertices for a graph that is consumed in a streaming fashion is not trivial. To avoid the overhead of a preprocessing step (where the input graph should be fully scanned to calculate the vertex exact degrees), a table with partial degrees of the vertices can be maintained that is continuously updated while input is analyzed. As each new edge is considered in the input, the degree values for the corresponding vertices are updated in the table. The partial degree values collected at runtime are usually a good indicator for the actual degree of a vertex since it is more likely that an observed edge belongs to a high-degree vertex rather than to a low-degree one.[2]

More formally, when processing edge $e \in E$ connecting vertices v_i and v_j, the *HDRF* algorithm retrieves their partial degrees and increments them by one. Let $\delta(v_i)$ be the partial degree of v_i and $\delta(v_j)$ be the partial degree of v_j. The degree values are then normalized such that they sum up to one:

$$\theta(v_i) = \frac{\delta(v_i)}{\delta(v_i) + \delta(v_j)} = 1 - \theta(v_j) \tag{5}$$

As for the greedy heuristic, the *HDRF* algorithm computes a score $C^{\text{HDRF}}(v_i, v_j, p)$ for all partitions $p \in P$, and then assigns e to the partition p^* that maximizes C^{HDRF}. The score for each partition $p \in P$ is defined as follows:

$$C^{\text{HDRF}}(v_i, v_j, p) = C^{\text{HDRF}}_{\text{REP}}(v_i, v_j, p) + C^{\text{HDRF}}_{\text{BAL}}(p) \tag{6}$$

$$C^{\text{HDRF}}_{\text{REP}}(v_i, v_j, p) = g(v_i, p) + g(v_j, p) \tag{7}$$

$$g(v, p) = \begin{cases} 1 + (1 - \theta(v)), & \text{if } p \in A(v) \\ 0, & \text{otherwise} \end{cases}$$

$$C^{\text{HDRF}}_{\text{BAL}}(p) = \lambda \cdot C^{\text{greedy}}_{\text{BAL}}(p) = \lambda \cdot \frac{\text{maxsize} - |p|}{\epsilon + \text{maxsize} - \text{minsize}} \tag{8}$$

The λ parameter allows control of the extent of partition size imbalance in the score computation. We introduced this parameter because the standard *greedy* heuristic may result in highly imbalanced partition sizes, especially when the input is ordered somehow. To see this problem note that $C^{\text{greedy}}_{\text{BAL}}(p)$ (Equation 4) is always smaller than one, while $C^{\text{greedy}}_{\text{REP}}$ and $C^{\text{HDRF}}_{\text{REP}}$ are either zero or greater than one. For this reason, the balance term C_{BAL} in the *greedy* algorithm or when $0 < \lambda \leq 1$ is used only to choose among partitions that exhibit the same value for the replication term C_{REP}, thereby breaking symmetry.

However, this may not be enough to ensure load balance. For instance, if the stream of edges is ordered according to some visit order on the graph (e.g., breadth first search or depth first search), when processing edge $e \in E$ connecting vertices v_i and v_j there is always a single partition p^* with $C^{\text{greedy}}_{\text{REP}}(v_i, v_j, p^*) \geq 1$ (resp. $C^{\text{HDRF}}_{\text{REP}}(v_i, v_j, p^*) > 1$) and all the other partitions $p \in P$ s.t. $p \neq p^*$ have $C^{\text{greedy}}_{\text{REP}}(v_i, v_j, p) = 0$ (resp. $C^{\text{HDRF}}_{\text{REP}}(v_i, v_j, p) = 0$). In this case, the balance term is useless as there is no symmetry to break, and the heuristic ends up placing all edges in a single partition p^*. This problem can be solved by setting a value for $\lambda > 1$. In our evaluation (Section 6), we empirically studied the trend of the replication factor and the load balance by varying λ (Figure 6). Moreover, note that when $\lambda \to \infty$ the algorithm resembles a random heuristic, where past observations are ignored and it only matters to have partitions with equal size. The following summarizes the behavior of the *HDRF* algorithm with respect to the λ parameter:

$$\begin{cases} \lambda = 0, & \text{agnostic of the load balance} \\ 0 < \lambda \leq 1, & \text{balance used to break the symmetry} \\ \lambda > 1, & \text{balance importance proportional to } \lambda \\ \lambda \to \infty, & \text{random edge assignment} \end{cases}$$

When $\lambda = 1$ the *HDRF* algorithm can be represented by a set of simple rules, exactly as in *greedy*, with the exception of *Case 4* that is modified as follows:

Case 4 If $A(v_i) \neq \emptyset$, $A(v_j) \neq \emptyset$ and $A(v_i) \cap A(v_j) = \emptyset$, then
- if $\delta(v_i) < \delta(v_j)$, e is assigned to the partition with the smallest size $p^* \in A(v_i)$ and a new replica of v_j is created in p^*;

[2]During experiments, we noticed no significant improvements in the algorithm performance when using exact degrees instead of their approximate values.

- if $\delta(v_j) < \delta(v_i)$, e is assigned to the partition with the smallest size $p^* \in A(v_j)$ and a new replica of v_i is created in p^*.

HDRF can be run as a single process or in parallel instances to speed up the partitioning phase. As with *greedy*, *HDRF* also needs some state to be shared among parallel instances during partitioning. In particular, we noticed that sharing the values of $A(v)$, $\forall v \in V$ is sufficient to let *HDRF* perform at its best. Note that optimizing the execution time of *HDRF* was a goal beyond the scope of this work; we will consider it as part of our future work.

5. THEORETICAL ANALYSIS

In this section we characterize the *HDRF* algorithm behavior from a theoretical perspective, focussing on the vertex replication factor. In particular we are interested in an average-case analysis of *HDRF*. A worst-case analysis would provide poor performance, as expected for any similar greedy algorithm, while failing to capture the typical behavior of *HDRF* in real cases. In the rest of this section we assume $\lambda = 1$ for the sake of simplicity.

Cohen et al. [8] considered the problem of a scale-free network (characterized as a power-law graph) attacked by an adversary able to remove a fraction c of vertices with the largest degrees. In particular they characterized the approximate maximum degree $\tilde{M}$ observable in the graph's largest component after the attack. If $|V| \gg 1/c$ this value can be approximated by the following equation:

$$\tilde{M} = mc^{1/(1-\alpha)} \tag{9}$$

where m is the (global) minimum vertex degree and α is the parameter characterizing the initial vertex degree distribution.

Let us now consider the algorithm *aHDRF* as an approximation of *HDRF*: *aHDRF* performs exactly as *HDRF*, but for the fact that we assume it knows the exact degree of each input vertex (and not the observed degree as for *HDRF*).

THEOREM 1. *Algorithm* aHDRF *achieves a replication factor, when applied to partition a graph with $|V|$ vertices on $|P|$ partitions, that can be bounded by:*

$$RF \leq \tau|P| + \frac{1}{|V|(1-\tau)} \sum_{i=0}^{|V|(1-\tau)-1} \left[1 + m\left(\tau + \frac{i}{|V|}\right)^{\frac{1}{1-\alpha}}\right]$$

$$\tau = \left(\frac{|P|-1}{m}\right)^{1-\alpha}$$

PROOF. The replication factor bound is the sum of two distinct parts. The first part considers the fraction τ of vertices with the largest degrees in the graph, referred to as *hubs*. The worst case for hubs is to be replicated in all the partitions, with a corresponding replication factor of $\tau|P|$. τ represents the fraction of vertices that must be removed from the graph such that the maximum vertex degree in the remaining graph is $|P| - 1$; this value is obtainable through Equation (9) by imposing $\tilde{M} = |P| - 1$.

The second part of the equation consider the contribution to the replication factor from non-hub vertices, i.e. all vertices whose degree is expected to be smaller than $|P| - 1$ after the τ vertices with the largest degrees have been removed from the graph (together with their edges). When *aHDRF* processes an edge connecting a hub vertex with a non-hub vertices, it always favors the replication the hub vertex (that has a larger degree) and replicates the non-hub vertex only if executes *Case 1* or *Case 2*, that is only if it is the first time it observes that vertex. Since the degree of non-hub vertices, ignoring the connections with hub vertices, is bounded by $m\tau^{1/(1-\alpha)}$, and since the connections with hub vertices can produce at most one replica, the worst case replication factor for non-hub vertices is bounded by:

$$\frac{1}{|V|(1-\tau)}\left(1 + m\tau^{\frac{1}{1-\alpha}}\right)$$

This bound can be further improved by considering what happens to the graph once the non-hub vertex v_0 with the largest degree is removed. The previous bound is valid for v_0. However, the removal of v_0 from the graph will change the degree distribution, thus also reducing the bound for the next non-hub vertex with the largest degree. Using this consideration, it is possible to iteratively bound the degree of each non-hub vertex v_i with $m(\tau + i/|V|)^{1/(1-\alpha)}$ where $0 \leq i \leq |V|(1-\tau)-1$. Hence, the total worst case replication factor for non-hub vertices, is bounded by:

$$\frac{1}{|V|(1-\tau)} \sum_{i=0}^{|V|(1-\tau)-1} \left[1 + m\left(\tau + \frac{i}{|V|}\right)^{\frac{1}{1-\alpha}}\right]$$

□

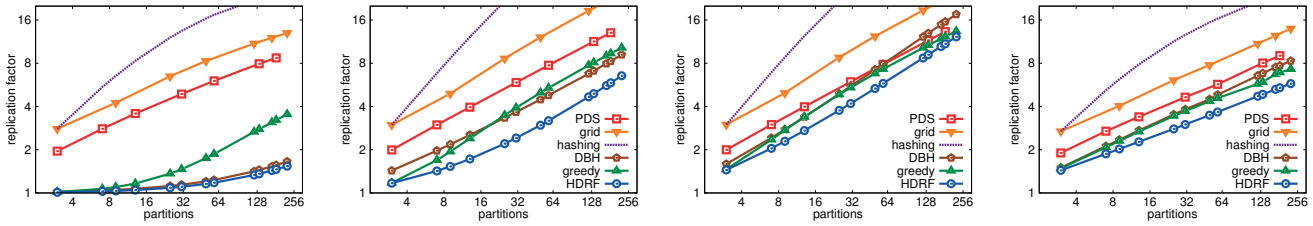
If edges arrive in random order, *aHDRF* gives an approximation of *HDRF*. In this case, the observed values for vertex degrees are a good estimate for the actual degrees. We can conclude that, assuming random order arrival for edges, *HDRF* is expected to achieve a replication factor, when applied to partition a graph with $|V|$ vertices on $|P|$ partitions, of at most *RF* of Theorem 1.

For example, consider a graph with $\alpha = 2.2$, $|P| = 128$, $m = 1$ and 1M vertices. The average-case upper bound for the replication factor of *HDRF* is ≈ 5.12 while the actual result it achieves is ≈ 1.37. The bounds for *DBH* and *hashing* [12, 27] with this configuration are respectively ≈ 5.54 and ≈ 5.88, while the actual results they achieve are ≈ 1.89 and ≈ 2.52.

The upper bound given by Theorem 1 cannot be extended to other algorithms (e.g., *greedy*). Informally, *HDRF breaks* network at hubs by replicating a small fraction of vertices with large degrees. In contrast, *greedy* and other algorithms are agnostic to the degree of vertices when replicating them. Intuitively, these algorithms try to break network by removing *random* vertices. Unfortunately, power-law graphs are resilient against removing random vertices (see [7] for details). This implies that, in order to fragment a scale-free network, a very large number of random vertices should be removed. In other words, *greedy* and other algorithms tend to replicate a large number of vertices in different partitions. This intuition is verified in our experiments (see Section 6).

6. EVALUATION

This section presents experimental results for the *HDRF* partitioning algorithm. The evaluation was performed on real-world graphs by running the proposed algorithm both in a stand-alone partitioner (useful for scaling up to large partition numbers) and running an implementation of *HDRF*



(a) Tencent Weibo (b) Netflix (c) MovieLens 10M (d) twitter-2010

Figure 1: Replication factor varying the number of target partitions (log-log scale).

Dataset	$\|V\|$	$\|E\|$
Tencent Weibo	1.4M	140M
Netflix	497.9K	100.4M
MovieLens 10M	80.6K	10M
twitter-2010	41.7M	1.47B

Table 1: Statistics for real-world graphs.

integrated into GraphLab[3]. The evaluation also reports experiments on synthetic graphs generated randomly with increasingly skewed distributions to study the extent to which *HDRF* performance is sensitive to workload characteristics.

6.1 Experimental Settings and Test Datasets

Evaluation Metrics — We evaluate the performance of *HDRF* by measuring the following metrics:

Replication factor: is the average number of replicas per vertex. This metric is a good measure of the synchronization overhead and should be minimized.

Load relative standard deviation: is the relative standard deviation of the number of edges hosted in target partitions. An optimal partitioning strategy should have a value for this metric close to 0.

Max partition size: is the number of either vertices or edges hosted in the largest partition. We consider this metric with respect to both vertices and edges as each conveys different information. Edges are the main input for the computation phase, thus more edges in a partition mean more computation for the computing node hosting it; conversely, the number of vertices in the system, and, therefore, in the largest partition, also depends on the number of replicas generated by the partitioning algorithm.

Execution time: is the number of seconds needed by the DGC framework to perform the indicated computation on the whole input graph. Better partitioning, by reducing the number of replicas, is expected to reduce the synchronization overhead at runtime and thus reduce the execution time as well.

Datasets — In our evaluation, we used as datasets both synthetic power-law graphs and real-word graphs. The former were used to study how *HDRF* performance vary when the degree distribution skewness of the input graph gradually increases. In particular, each synthetic graph was generated with 1M vertices, minimum degree of 5 and edges using a power law distribution with α ranging from 1.8 to 4.0. Therefore, the number of edges in the graphs ranges from $\sim$ 60M ($\alpha = 1.8$) to $\sim$ 3M ($\alpha = 4$). Graphs were generated with *gengraph* [26]. We also tested the performance of *HDRF* on real-world graphs: *twitter-2010* from LAW (Laboratory for Web Algorithmics) [5, 4], *Tencent Weibo* from the KDD-Cup 2012 [22], *Netflix* from the Netflix Prize [3] and *MovieLens 10M* from the GroupLens research lab (`http://grouplens.org`). Table 1 reports some statistics for these 4 datasets.

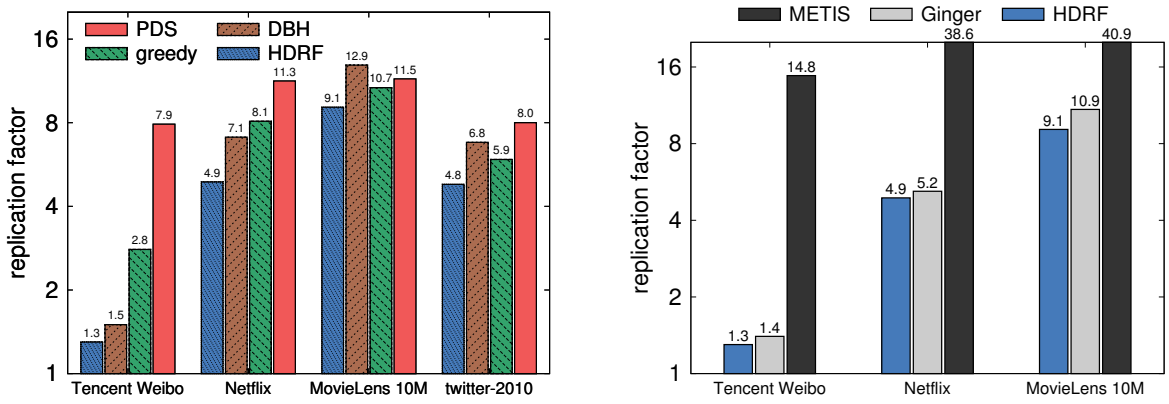
System Setup — We implemented a stand-alone version of a graph partitioner that captures the behavior of a DGC framework during the graph loading and partitioning phase. Within our partitioner, we implemented the five different algorithms described so far: *hashing*, *DBH*, *grid*, *PDS*, *greedy* and *HDRF*. Furthermore, we compared our solution against two offline methods: *Ginger* [24] and METIS [15], a well-known edge-cut partitioning algorithm. To compute the replication factor delivered by METIS, we used the same strategy of [12]: every edge-cut forces the two spanned partitions in maintaining a replica of both vertices and a copy of the edge data. To run realistic tests needed to measure execution time, we implemented and integrated *HDRF* into GraphLab v2.2. Experiments with GraphLab where conducted on a cluster consisting of 8 machines with dual 16-core *Intel Xeon* CPUs and 128*GB* of memory each. We experimented with 32, 64 and 128 partitions by running multiple instances on a single machine.

Data input order — Since the input dataset is consumed as a stream of edges, the input order can affect the performance of the partitioning algorithm. We considered three different stream orders as in [25]: *random*, where edges arrive according to a random permutation; *BFS*, where the edge order is generated by selecting a vertex uniformly at random and performing a breadth first search visit starting from that vertex; *DFS*, that works as *BFS* except for the visit algorithm that is depth-first search. All reported results are based on a *random* input order unless otherwise mentioned in the text.

6.2 Performance Evaluation

The experimental results reported in this section are organized as follows: we first report on experiments that show the ability of *HDRF* to deliver the best overall performance in terms of execution time with the smallest overhead (replication factor) and close to optimal load balance when executed on real-world graphs. We then study how *HDRF* performance is affected by changes in the characteristics of the input dataset and changes in the target number of partitions. Finally, the last set of results analyze the sensitivity of *HDRF* to input stream ordering.

[3]The stand-alone software and the GraphLab patch are available at `https://github.com/fabiopetroni/VGP`.



(a) Streaming algorithms

(b) Offline algorithms

Figure 2: Replication factor (log scale) with $|P| = 133$. *HDRF* is compared against streaming (a) and offline (b) solutions.

6.2.1 *Runtime comparison*

We first measured *HDRF* performance against other streaming partitioning algorithms on our set of real-world graphs. These experiments were run by partitioning the input graphs on a set of target partitions in the range $[3, 256]$ with our stand-alone partitioner. Figure 1 reports the replication factor that the considered partitioning algorithms achieve on different input graphs[4]. Moreover, Figure 2a provides a snapshot of the evaluation, by setting the number of target partition to 133, a number compliant with PDS constraints. It can be observed that *HDRF* is the algorithm that provides the smallest replication factor for all the considered datasets.

In particular, for the Weibo dataset, characterized by large edge count differences among high-degree and low-degree vertices, it is possible to observe how *HDRF* and *DBH* are the best performers as they both exploit vertex degrees. In all the other datasets *HDRF* is always the best performer, albeit with larger absolute RF values. Summarizing, on the considered datasets *HDRF* achieves on average a replication factor about 40% smaller than DBH, more than 50% smaller than *greedy*, almost 3× smaller than *PDS*, more than 4× smaller than *grid* and almost 14× smaller than *hashing*. We experimented with other datasets as well (i.e. *arabic-2005*, *uk-2002*, *indochina-2004* from LAW, and *Yahoo! Music* from the KDD-Cup 2011). In all our test *HDRF* outperforms competing solutions, simultaneously guaranteeing close to perfect load balance (results omitted due to space constraints).

Next, we compared *HDRF* against two offline partitioning algorithms: *Ginger* and *METIS*. Note that these offline solutions have full knowledge of the input graph that can be exploited to drive their partitioning choices. Figure 2b compares the replication factor achieved by these two solutions and *HDRF* (we maintain $|P| = 133$ to be coherent with Figure 2a). We do not report the results for the *twitter-2010* dataset since *METIS* produced greatly unbalanced partitions[5], making a comparison on this dataset unfair. The poor performance of *METIS* was an expected result since it has been proved that edge-cut approaches perform worse than vertex-cut ones on power-law graphs [12]. However, *HDRF* outperforms *Ginger* as well, by reducing its replication factor by 10% on average. In addition, *HDRF* has the clear advantage of consuming the graph in a one-pass fashion while *Ginger* needs several passes over the input data to converge. These results show that *HDRF* is always able to provide a smaller average replication factor with respect to all other algorithms, both streaming and offline, when used to partition graphs with power-law degree distributions.

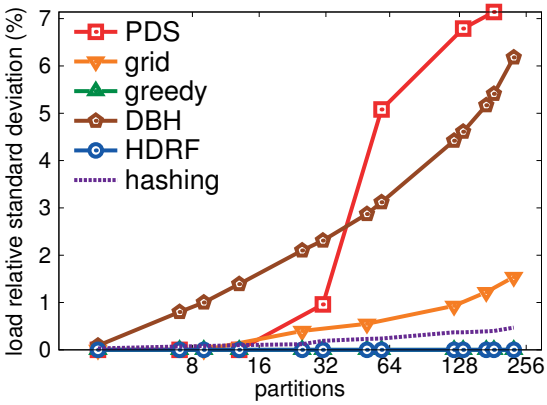


Figure 3: Load relative standard deviation produced by different partitioning algorithms on the MovieLens 10M dataset.

Figure 3 reports the load relative standard deviation produced by the tested streaming algorithms when run on the *MovieLens 10M* dataset with a variable number of target partitions (results for other datasets showed similar behavior so we omit them). The curves show that *HDRF* and *greedy* provide the best performance as the number of target partitions grows. As expected, *hashing* provides well bal-

[4]Due to specific constraints imposed by the *PDS* and *grid* algorithms on the total number of partitions, their data points are not aligned with those of the other algorithms.

[5]Note that the scope of Metis is to balance vertex load among partitions.

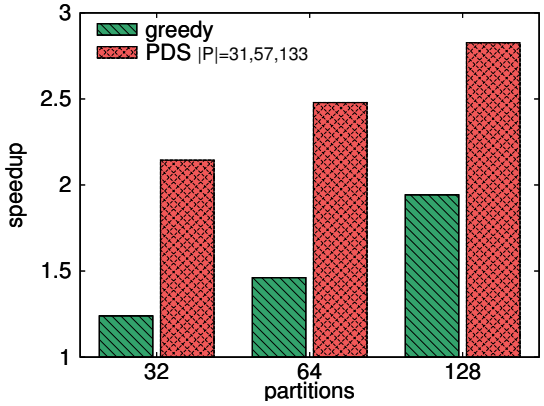


Figure 4: Speedup in the execution time for the SGD algorithm on the *Tencent Weibo* dataset by applying *HDRF* with respect to *greedy* and *PDS*, with 32, 64 and 128 partitions.

anced partitions, but it still performs worse that the other algorithms as its expected behavior with respect to load balancing is only probabilistic. *Grid* performs similarly, even if its more complex constraints induce some skew in load. *DBH* and *PDS* are the worse performers, with load skew growing at a fast pace as the number of target partitions grows. Note that replication factor reflects communication cost, and edge-imbalance reflects workload-imbalance; providing good performance with these two metrics means that *HDRF* can provide partitioning results that make the execution of application algorithms more efficient.

To this end, we studied how much all of this translates to a performance improvement with respect to the execution time. Since a DGC framework has to periodically synchronize all the vertex replicas, having fewer replicas in the system is expected to provide an advantage and to speed up the execution time. To investigate the impact of the different partitioning techniques we ran the Stochastic Gradient Descent (SGD) algorithm for matrix completion [17, 23] on GraphLab, using the *Tencent Weibo* dataset and 100 latent factors, with 32, 64 and 128 partitions respectively. Figure 4 reports the measured speed-up, obtained by using *HDRF* to partition the input over *greedy* and *PDS*[6]. The SGD algorithm runs up to 2× faster using *HDRF* as input partitioner with respect to *greedy*, and close to 3× faster than *PDS*. The actual improvement is larger as the number of target partitions grows. Moreover, the speedup is proportional to the gain in RF (see Figure 1a) and, as already shown in [12], halving the replication factor approximately halves runtime. Furthermore, having partitions with fewer replicas also help SGD to converge faster [23].

We tested the speedup for other datasets and algorithms as well, namely Single Source Shortest Path (SSSP), Weakly Connected Components (WCC), Page Rank (PR) and Alternating Least Squares (ALS). The results (not reported here due to space constraints) confirmed our intuitions: the speedup is proportional to both the advantage in replication factor and the actual network usage of the algorithm. The speedup it is larger for IO-intensive algorithms (e.g. SGD, ALS and PR) and smaller for algorithm with less network IO (SSSP and WCC). None of the tests we conducted with *HDRF* showed a slowdown with respect to other solutions.

Our results show that *HDRF* is the best solution to partition input graphs characterized by skewed power-law degree distributions. *HDRF* achieves the smallest replication factor with close to optimal load balance. These two characteristics combined make application algorithms execute more efficiently in the DGC framework.

6.2.2 Performance sensitivity to input shape

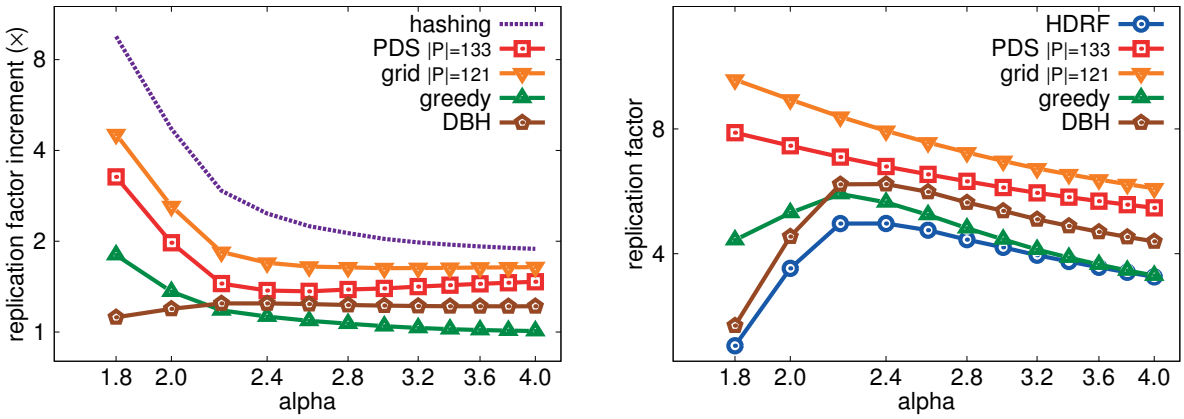
We next analyze how the input graph degree distribution affects *HDRF* performance. To this end, we used *HDRF* to partition a set of synthetic power-law graphs. In doing so, we experimentally characterize the sensitivity of the average replication factor on the power-law shape parameter α, and on the number of partitions. Figure 5a reports the replication factor improvement for *HDRF* with respect to other algorithms, expressed as a multiplicative factor, by varying α in the range $[1.8, 4.0]$ with $|P| = 128$ target partitions ($|P| = 133$ and $|P| = 121$ for *PDS* and *grid* respectively). The curves show two important aspects of *HDRF* behavior: (1) with highly skewed degree distributions (i.e. small values of α), its performance is significantly better than *greedy* and other algorithms (with the exception of DBH); (2) with less skewed degree distributions, the performance of *HDRF* approaches that provided by *greedy*, while all the other solutions (including *DBH*) perform worse. These results show how *HDRF* behavior approximates *greedy*'s behavior as the number of high degree vertices in the input graph grows as in this case making a different partitioning choice on high-degree vertices is less useful (as there are a lot of them). Note that Gonzalez et al. [12] showed that the effective gain of a vertex-cut approach relative to an edge-cut approach actually increases with smaller α. Our solution boosts this gain, not only with respect to constrained techniques but also over the *greedy* algorithm. Figure 5b reports the replication factor, and clearly shows that *HDRF* is able to outperform all competing algorithms for all values of α. At the extremes, *HDRF* is better than *DBH* when α is very small and performs slightly better than *greedy* when α is very large.

6.2.3 Performance sensitivity to input order

A shortcoming of the standard *greedy* algorithm is its inability to effectively handle streams of input edges when they are ordered. If the input stream is ordered such that two subsequent edges always share a vertex, *greedy* always places all the edges and their adjacent vertices in a single partition, whatever the target partition number is. The final result is clearly far from being desirable as all of the computation load will be incurred by a single node in the computing cluster.

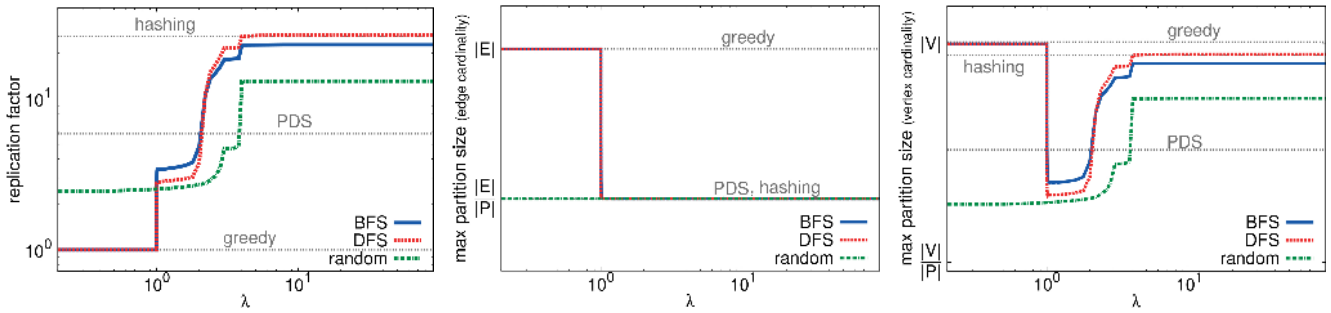
To overcome this limitation, we explicitly introduced the parameter λ in the computation of the score for each partition in *HDRF* (Equations (6) and (8)), that defines the importance of the load balance in the edge placement decision (Section 4). Figure 6 shows the result of an experiment run on the Netflix dataset, where the input stream of edges is ordered according to either a *depth-first-search* (DFS) or a *breadth-first-search* (BFS) visit on the graph. The figure shows the average replication factor (Figure 6a), the size of the largest partition expressed as number of contained edges (Figure 6b) and the size of the largest partition ex-

[6]We needed to use respectively 31, 57 and 133 partitions, to fit *PDS* constraints.



(a) RF increment varying α ($HDRF$= 1) (b) RF varying α

Figure 5: Replication factor improvement for *HDRF* in synthetically generated graphs (log-log scale). Figure (a) reports the replication factor increment and Figure (b) the actual replication factor when α grows ($|P| = 128$ except for *PDS*, where $|P| = 133$, and *grid*, where $|P| = 121$). For Figure (a) *HDRF* represents the baseline.



(a) replication factor (b) max edge cardinality (c) max vertex cardinality

Figure 6: *HDRF* behavior on the Netflix dataset varying λ, with input edge stream either random or ordered (DFS or BFS graph visits). Reference grey lines represent *greedy*, *hashing* and *PDS* performance.

pressed as number of contained vertices (Figure 6c) all while varying the value of λ in the range $[0.1, 100]$ (log-log axes). All three figures report the performance obtained with the *greedy*, *hashing* and *PDS* algorithms as horizontal grey lines for reference.

With $\lambda \leq 1$ *HDRF* behaves exactly as *greedy* (curves BFS and DFS): all edges are placed in a single partition and no vertex is replicated. This behavior is confirmed by the size of the largest partition that in this case contains exactly $|E|$ edges and $|V|$ vertices (Figures 6b and 6c). For $\lambda > 1$ the C_{BAL} factor starts to play a fundamental role in balancing the load among the available partitions: the average replication factor for *HDRF* with both DFS and BFS inputs is just slightly larger than what is achievable with a random input[7] and still substantially lower than what is achievable with *PDS* or *hashing* (Figure 6a). At the same time, the size of the largest partition drops to its minimum (Figures 6b and 6c) indicating that the algorithm immediately delivers close to perfect load balancing (i.e. $|E|/|P|$ edges per partition), while the number of vertices hosted in the largest partition reaches its minimum. By further increasing λ toward larger values, the effect of C_{BAL} dominates the *HDRF* score computation and the algorithms behavior quickly approaches the behavior typical of *hashing*: large average replication factor, with close to perfect load balancing.

These results show that i) the C_{BAL} term in *HDRF* score computation plays an effective role in providing close-to-perfect load balancing among partitions while keeping a low average replication factor, and ii) by setting the value of λ slightly larger than 1, it is possible to let *HDRF* work at a "sweet spot" where it can deliver the best performance, even when working on an ordered stream of edges. This last point makes *HDRF* particularly suitable for application settings where it is not possible to randomize the input stream before feeding it to the graph partitioning algorithm.

[7]The difference is due to *HDRF*'s usage of partial information on vertex degrees. Such values are not a good proxy of real vertex degrees if the input stream is not random.

7. CONCLUSION

Distributed graph-computing frameworks provide programmers with convenient abstractions to enable computation on large datasets. In these frameworks, system performance is often determined by the graph data partitioning strategy, which impacts the communication cost and the workload balance among compute resources. In this paper, we pro-

posed *HDRF*, a novel stream-based graph partitioning algorithm for distributed graph-computing frameworks. *HDRF* is based on a greedy vertex-cut approach that leverages information on vertex degrees. Through a theoretical analysis and an extensive experimental evaluation on real-world as well as synthetic graphs using both a stand-alone partitioner and implementation of *HDRF* in GraphLab, we showed that HDRF is overall the best performing partitioning algorithm for graphs characterized by power-law degree distributions. In particular, *HDRF* provides the smallest average replication factor with close to optimal load balance. These two characteristics put together allow *HDRF* to significantly reduce the time needed to perform computation on graphs and makes it the best choice for partitioning graph data.

8. REFERENCES

[1] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000.

[2] K. Andreev and H. Räcke. Balanced graph partitioning. In *Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2004.

[3] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, 2007.

[4] 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*, 2011.

[5] P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, 2004.

[6] D. S. Callaway, M. E. Newman, S. H. Strogatz, and D. J. Watts. Network robustness and fragility: Percolation on random graphs. *Physical review letters*, 85(25):5468, 2000.

[7] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin. Resilience of the internet to random breakdowns. *Physical review letters*, 85(21):4626, 2000.

[8] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin. Breakdown of the internet under intentional attack. *Physical review letters*, 86(16):3682, 2001.

[9] S. N. Dorogovtsev and J. F. Mendes. Evolution of networks. *Advances in physics*, 51(4):1079–1187, 2002.

[10] C. Eaton, D. Deroos, T. Deutsch, G. Lapis, and P. Zikopoulos. *Understanding Big Data*. Mc Graw Hill, 2012.

[11] U. Feige, M. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.

[12] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*, 2012.

[13] H. Halberstam and R. Laxton. Perfect difference sets. In *Proceedings of the Glasgow Mathematical Association*, 1964.

[14] N. Jain, G. Liao, and T. L. Willke. Graphbuilder: Scalable graph etl framework. In *1st International Workshop on Graph Data Management Experiences and Systems*, 2013.

[15] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[16] M. Kim and K. S. Candan. SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices. *Data & Knowledge Engineering*, 72:285–303, 2012.

[17] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[18] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[19] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new framework for parallel machine learning. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, 2010.

[20] 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.

[21] J. Mondal and A. Deshpande. Managing large dynamic graphs efficiently. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012.

[22] Y. Niu, Y. Wang, G. Sun, A. Yue, B. Dalessandro, C. Perlich, and B. Hamner. The tencent dataset and kdd-cup'12. In *KDD-Cup Workshop*, 2012.

[23] F. Petroni and L. Querzoni. Gasgd: stochastic gradient descent for distributed asynchronous matrix completion via graph partitioning. In *Proceedings of the 8th ACM Conference on Recommender systems*, 2014.

[24] Y. C. R. Chen, J. Shi and H. Chen. Powerlyra: Differentiated graph computation and partitioning on skewed graphs. In *Proceedings of the 10th ACM SIGOPS European Conference on Computer Systems*, 2015.

[25] 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.

[26] F. Viger and M. Latapy. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. In *Computing and Combinatorics*. Springer, 2005.

[27] C. Xie, L. Yan, W.-J. Li, and Z. Zhang. Distributed power-law graph computing: Theoretical and empirical analysis. In *Advances in Neural Information Processing Systems*, 2014.

[28] S. Yang, X. Yan, B. Zong, and A. Khan. Towards effective partition management for large graphs. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, 2012.