A Study of Partitioning Policies for Graph Analytics on Large-scale Distributed Platforms

Gurbinder Gill, Roshan Dathathri, Loc Hoang, Keshav Pingali
Department of Computer Science, University of Texas at Austin
{gill,roshan,loc,pingali}@cs.utexas.edu

Abstract

Distributed-memory clusters are used for in-memory processing of very large graphs with billions of nodes and edges. This requires partitioning the graph among the machines in the cluster. When a graph is partitioned, a node in the graph may be replicated on several machines, and communication is required to keep these replicas synchronized. Good partitioning policies attempt to reduce this synchronization overhead while keeping the computational load balanced across machines. A number of recent studies have looked at ways to control replication of nodes, but these studies are not conclusive because they were performed on small clusters with eight to sixteen machines, did not consider work-efficient data-driven algorithms, or did not optimize communication for the partitioning strategies they studied.

This paper presents an experimental study of partitioning strategies for work-efficient graph analytics applications on large KNL and Skylake clusters with up to 256 machines using the Gluon communication runtime which implements partitioning-specific communication optimizations. Evaluation results show that although simple partitioning strategies like Edge-Cuts perform well on a small number of machines, an alternative partitioning strategy called Cartesian Vertex-Cut (CVC) performs better at scale even though paradoxically it has a higher replication factor and performs more communication than Edge-Cut partitioning does. Results from communication micro-benchmarks resolve this paradox by showing that communication overhead depends not only on communication volume but also on the communication pattern among the partitions.

These experiments suggest that high-performance graph analytics systems should support multiple partitioning strategies, like Gluon does, as no single graph partitioning strategy is best for all cluster sizes. For such systems, a decision tree for selecting a good partitioning strategy based on characteristics of the computation and the cluster is presented.

1. INTRODUCTION

Graph analytics systems must handle extremely large graphs with billions of nodes and trillions of edges [30]. Graphs of this size do not fit in the main memory of a single machine, so systems like Pregel [32], PowerGraph [18], PowerLyra [12], Gemini [51], and D-Galois [15] use distributed-memory clusters. Since each host in the cluster can address only its own memory, it is necessary to partition the graph among hosts and use a communication layer like MPI to perform data transfers between hosts [7, 8, 11, 12, 18, 23, 41, 42, 44].

Graph partitioning must balance two concerns.

The first concern is *computational load balance*. For the graph algorithms considered in this paper, computation is performed in rounds: in each round, *active* nodes in the graph are visited, and a computation is performed by reading and writing the immediate neighbors of the active node [30]. For simple *topology-driven* graph algorithms in which all nodes are active at the beginning of a round, the computational load of a host is proportional to the numbers of nodes and edges assigned to that host, so by dividing nodes and edges evenly among hosts, it is possible to achieve load balance [12]. However, work-efficient graph algorithms like the ones considered in this paper are *data-driven*: nodes become active in data-dependent, statically unpredictable ways, so a statically balanced partition of the graph does not necessarily result in computational load balance.

The second concern is *communication overhead*. We consider graph partitioning with replication: when a graph is partitioned, its edges are divided up among the hosts, and if edge $(n_1 \rightarrow n_2)$ is assigned to a host h, proxy nodes are created for n_1 and n_2 on host h and connected by an edge. A given node in the original graph may have proxies on several hosts in the partitioned graph, so updates to proxies must be synchronized during execution using inter-host communication. This communication entirely dominates the execution time of graph analytics applications on large scale clusters as shown by the results in Section 4, so optimizing communication is the key to high performance.

Substantial effort has gone into designing graph partitioning strategies that reduce communication overhead. Overlapping communication with computation (as done in HPC applications) would reduce its relative overhead, but there is relatively little computation in graph analytics applications. Therefore, reducing the volume of communication has been the focus of much effort in this area. Since communication is needed to synchronize proxies, reducing the average number of proxies per node (known in the literature as the *average replication factor*) while also ensuring computational load balance can reduce communication [8, 23, 41, 42, 44]. This is one of the driving principles behind the Vertex-Cut partitioning strategy (Vertex-Cuts and other partitioning strategies are described in detail in Section 2) used in PowerGraph [18] and PowerLyra [12]. Partitioning policies developed for efficient distribution of sparse matrix computation such as 2D block partitioning [7, 11] can also be used since sparse graphs are isomorphic to sparse matrices.

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/TBD

Several papers [3,7,12,45,51] have studied how the performance of graph analytics applications changes when different partitioning policies are used, and they have advocated particular partitioning policies based on their results. We believe these studies are not conclusive for the following reasons.

- Most of the evaluations were done for small graphs on small clusters, so it is not clear whether their conclusions extend to large graphs and large clusters.
- In some cases, only topology-driven algorithms were evaluated, so it is not clear whether their conclusions extend to work-efficient data-driven algorithms.
- The distributed graph analytics systems used in the studies optimize communication only for particular partitioning strategies, putting other partitioning strategies at a disadvantage.

This paper makes the following contributions:

1. We present the first detailed performance analysis of state-of-the-art, work-efficient graph analytics applications using different graph partitioning strategies including Edge-Cuts, 2D block partitioning strategies, and general Vertex-Cuts on large-scale clusters, including one with 256 machines and roughly 69K threads. These experiments use a system called D-Galois, a distributed-memory version of the Galois system [36] based on the Gluon communication runtime [15]. Gluon performs communication optimizations that are specific to each partitioning strategy. Our results show that although Edge-Cuts perform well on small-scale clusters, a 2D partitioning policy called Cartesian Vertex-Cut (CVC) [7] performs the best at scale *even though it results in higher replication factors and higher communication volumes than the other partitioning strategies*.
2. We present an analytical model for estimating communication volumes for different partitioning strategies, and an empirical study using micro-benchmarks to estimate communication times. These help estimate and understand the performance differences in communication required by the partitioning strategies. In particular, these models explain why at scale, CVC has lower communication overhead even though it performs more communication.
3. We give a simple decision tree that can be used by the user to select a partitioning strategy at runtime given an application and the number of distributed hosts. Although the chosen policy might not be the best in some cases, we show that the application's performance using the chosen policy is almost as good as using the best policy.

This paper's contributions include some important lessons for designers of high-performance graph analytics systems.

1. It is desirable to support optimized implementations of *multiple* partitioning policies including Edge-Cuts and Cartesian Vertex-Cuts, like D-Galois does. Existing systems either support general partitioning, using approaches like gather-apply-scatter (PowerGraph, PowerLyra), without optimizing communication for particular partitioning policies like Edge-Cuts, or support only Edge-Cuts (Gemini). Neither approach is flexible enough.
2. An important lesson for designers of efficient graph partitioning policies is that the communication overhead in graph analytics applications depends on not only the communication volume but also the communication pattern among the partitions as explained in Section 2. The replication factor and the number of edges/vertices split between partitions [12, 18, 41, 44] are not adequate proxies for communication overhead.

The rest of this paper is organized as follows. Section 2 describes the graph partitioning strategies used in this study, including Edge-Cuts, 2D block partitioning strategies, and general Vertex-Cuts. Section 3 presents an analytical model for estimating the communication volumes required by different partitioning strategies and an empirical study using micro-benchmarks to model communication time. Section 4 presents detailed experimental results using state-of-the-art graph analytics benchmarks including data-driven algorithms for betweenness centrality, breadth-first search, connected components, page-rank, and single-source shortest-path. Section 5 presents a decision tree for choosing a partitioning strategy. Section 6 puts the contributions of this paper in the perspective of related work. Section 7 concludes the paper.

2. PARTITIONING POLICIES

The graph partitioning policies considered in this work divide a graph's edges among hosts and creating proxy nodes on each host for the endpoints of the edges assigned to that host. If edge $e : (n_1 \rightarrow n_2)$ is assigned to a host h, h creates proxy nodes on itself for nodes n_1 and n_2, and adds an edge between them. For each node in the graph, one proxy is made the *master*, and the other proxies are made *mirrors*. Intuitively, the master holds the canonical value of the node during the computation, and it communicates that value to the mirrors as needed. Each partitioning strategy represents choices made along two dimensions: (i) how edges are partitioned among hosts and (ii) how the master is chosen from the proxies of a given node. To understand these choices, it is useful to consider both the graph-theoretic (i.e., nodes and edges) and the adjacency matrix representation of a graph.

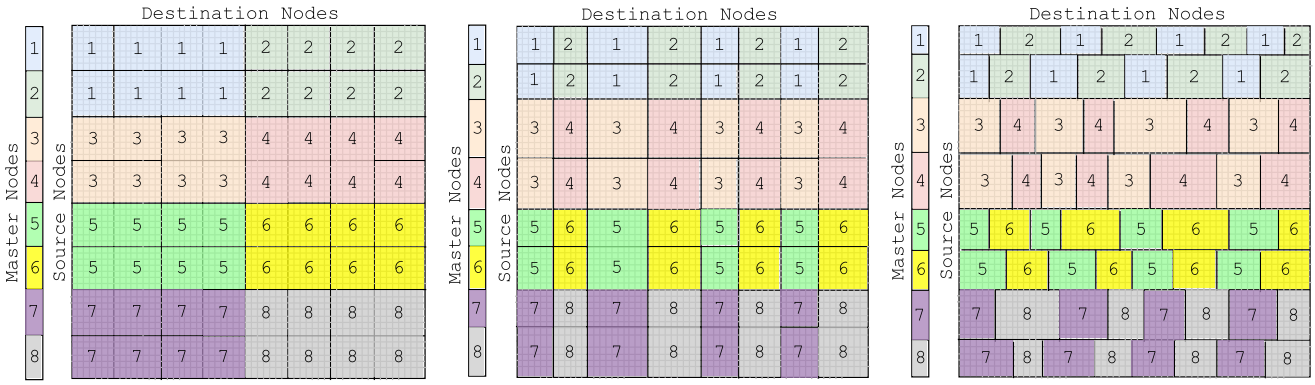
2.1 1D Partitions

In 1D partitioning, nodes are partitioned among hosts. If a host owns node n, all outgoing (or incoming) edges connected to n are assigned to that host, and the corresponding proxy for node n is made the master. In the graph analytical literature, this partitioning strategy is called outgoing (or incoming) Edge-Cut [23, 41, 42, 51]. In matrix-theoretic terms, this corresponds to assigning rows (or columns) to hosts. 1D partitioning is commonly used in stencil codes in computational science applications; the mirror nodes are often referred to as *halo nodes* in that context. Stencil codes use topology-driven algorithms on meshes, which are uniform-degree graphs, and computational load balance can be accomplished by assigning roughly equal numbers of nodes to all hosts. For power-law graphs, the adjacency matrix is irregular, and ensuring load balance for data-driven graph algorithms through static partitioning is difficult.

A number of policies are used in practice.

1. *Balanced nodes*: Assign roughly equal numbers of nodes to all hosts.
2. *Balanced edges*: Assign nodes such that all hosts have roughly the same number of edges.
3. *Balanced nodes and edges* [51]: Assign nodes such that a given linear combination of the number of nodes and edges on a host has roughly the same value on all hosts.

The first policy is the simplest and does not require any analysis of the graph. However, it may result in substantial load imbalance



(a) CheckerBoard Vertex-Cut (BVC) (b) Cartesian Vertex-Cut (CVC) (c) Jagged Vertex-Cut (JVC)

Figure 1: 2D partitioning policies.

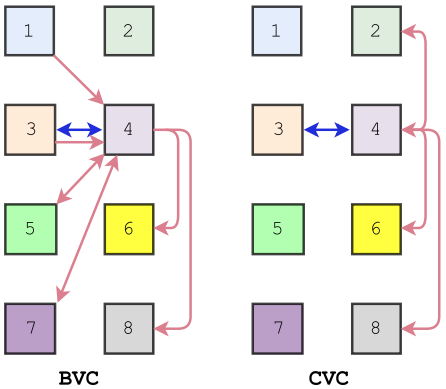


Figure 2: Communication patterns in BVC and CVC policies: red arrows are reductions and blue arrows are broadcasts for host 4.

for power-law graphs since there is usually a large variation in node degrees. The other two policies require computing the degree of each vertex and the prefix-sums of these degrees to determine how to partition the set of nodes.

2.2 2D Block Partitions

In 2D block partitioning, the adjacency matrix is blocked along both dimensions, and each host is assigned some of the blocks. Unlike in 1D partitioning, both outgoing and incoming edges of a given node may be distributed among different hosts. In the graph analytical literature, such partitioning strategies are called *Vertex-Cuts*. 2D block partitioning can be viewed as a restricted form of Vertex-Cuts in which the adjacency matrix is blocked.

This paper explores three alternative implementations of 2D block partitioning, illustrated in Figure 1 using a cluster of eight hosts in a 4×2 grid. The descriptions below assume that hosts are organized in a grid of size $p_r \times p_c$.

1. *CheckerBoard Vertex-Cuts (BVC)* [11, 27]: The nodes of the graph are partitioned into equal size blocks and assigned to the hosts. Masters are created on each host for its block of nodes. The matrix is partitioned into contiguous blocks of size $N/p_r \times N/p_c$, and each host is assigned one block of edges as shown in Figure 1(a). This approach is used by CombBLAS [9] for sparse matrix-vector multiplication (SpMV) on graphs. There are several variations to this approach; the one used in this study is shown in Figure 1(a).

2. *Cartesian Vertex-Cuts (CVC)* [7]: Nodes are partitioned among hosts using any 1D block partitioning policy, and masters are created on hosts for the nodes assigned to it. Unlike BVC, these node partitions need not be of the same size, as shown in Figure 1(b). This can happen, for example, if the 1D block partitioning assigns nodes to hosts such that the number of edges is balanced among hosts.

   The columns are then partitioned into same sized blocks as the rows. Therefore, blocks along the diagonal will be square, but other blocks may be rectangular unlike in BVC. These blocks of edges can be distributed among hosts in different ways. This study uses a block distribution along rows and a cyclic distribution along columns, as shown in Figure 1(b).

3. *Jagged Vertex-Cuts (JVC)* [11]: The first stage of JVC is similar to CVC. However, instead of partitioning columns into same sized blocks as the rows, each block of rows can be partitioned independently into blocks for a more balanced number of edges (non-zeros) in edge blocks. Therefore, blocks will not be aligned among the column dimension. These edge blocks can be assigned to hosts in any fashion, but in this study, the host assignment is kept the same as CVC as shown in Figure 1(c).

Although these 2D block partitioning strategies seem similar, they have different communication requirements.

Consider a push-style graph algorithm in which active nodes perform computation on their own labels and push values to their immediate outgoing neighbors, where they are reduced to compute labels for the next round. In a distributed-memory setting, a proxy with outgoing edges of a node n on host h push values to its proxy neighbors also present on h. These proxies may be masters or mirrors, and it is useful to distinguish two classes of mirrors for a node n: *in-mirrors*, mirrors that have incoming edges, and *out-mirrors*, mirrors that have outgoing edges. For a push-style algorithm in the distributed setting, the computation at an active node in the original graph is performed partially at each in-mirror, the intermediate values are reduced to the master, and the final value is broadcast to the out-mirrors. Therefore, the communication pattern can be described as broadcast along the row dimension and reduce along the column dimension of the adjacency matrix. The hosts that participate in broadcast and/or reduce depends on the 2D partitioning policy and the assignment of edge blocks to the hosts.

To illustrate this, consider Figure 2 which shows the 4×2 grid of hosts and the reduce and broadcast partners for host 4 in BVC and CVC. For BVC, by walking down the fourth block column of the matrix in Figure 1(a), we see that incoming edges to the nodes owned by host 4 may be mapped to hosts {1,3,5,7}, so labels

of mirrors on these hosts must be communicated to host 4 during the reduce phase. Similarly, the labels of mirrors on host 4 must be communicated to masters on hosts {5,6,7,8}. Once the values have been reduced at masters on host 4, they must be sent to hosts that own out-mirrors for nodes owned by host 4. Walking over the fourth block row of the matrix, we see that only host 3 is involved in this communication. Similarly, the labels of masters on the host {3} must be sent to host 4.

The same analysis can be done on CVC and JVC partitionings. For JVC, a given host may need to involve all other hosts in reductions and broadcasts, leading to larger communication requirements than CVC.

2.3 Unrestricted Vertex-Cut Partitions

Unrestricted or *general* Vertex-Cuts are partitioning strategies that assign edges to hosts without restriction, and they do not correspond to 1D or 2D blocked partitions. The partitioning strategies used in the PowerGraph [18] and PowerLyra systems [12] are examples of this strategy. For example, in PowerLyra's *Hybrid Vertex-Cut* (HVC), nodes are assigned to hosts in a manner similar to an Edge-Cut. However, high-degree nodes are treated differently from low-degree nodes to avoid assigning all edges connected to a high-degree node to the same host, which creates load imbalance. If $(n_1 \rightarrow n_2)$ is an edge and n_2 has low in-degree (based on some threshold), the edge is assigned to the host that owns n_2; otherwise, it is assigned to the node that owns n_1.

2.4 Discussion

A small detail in 2D block partitioning is that when the number of processors is not a perfect square, we factorize the number into a product $p_x \times p_y$ and assign the larger factor to the row dimension rather than the column dimension. This heuristic reduces the number of broadcast partners because the reduce phase communicates only updated values from proxies to the master and the number of these updates decrease over iterations, whereas the broadcast phase in each round sends the updated canonical value from the master to all its proxies.

In our studies, we observed that for Edge-Cuts and HVC, almost all pairs of hosts have proxies in common for some number of nodes. Therefore, for these partitioning policies, Edge-Cut partitioning performs all-to-all communication while HVC performs two rounds of all-to-all communication (from mirrors to masters followed by masters to mirrors). On the other hand, CVC has p_c number of parallel all-to-all communications among $p_r \approx \sqrt{P}$ hosts followed by p_r number of parallel all-to-all communications among $p_c \approx \sqrt{P}$ hosts. Therefore, each host in CVC sends fewer messages and has fewer communication partners than EC and HVC, which can help both at the application level (a host need not wait for another host in another row) and at the hardware level (less contention for network resources). The next section explores these differences among partitioning policies quantitatively.

3. PERFORMANCE MODEL FOR COMMUNICATION

This section presents performance models to estimate communication volume and communication time for three partitioning policies - Edge-Cut, Hybrid Vertex-Cut, and Cartesian Vertex-Cut. We formally define replication factor (Section 3.1), describe a simple analytical model for estimating communication volume using the replication factor (Section 3.2), and use micro-benchmarks to estimate the communication time from the communication volume (Section 3.3).

3.1 Replication Factor

Given a graph $G=(V, E)$, and a strategy S for partitioning the graph between P hosts, let v be a node in V and let $r_{P,S}(v)$ denote the number of proxies of v created when the graph is partitioned. $r_{P,S}(v)$ is referred to as the *replication factor*, and it is an integer between *1* and P. The average replication factor across all nodes is a number between 1 (no node is replicated) and P (all nodes are replicated between all hosts), and it is given by the following equation:

$$\overline{r}_{P,S} = \frac{\sum_{v \in V} r_{P,S}(v)}{|V|} \quad (1)$$

3.2 Estimating Communication Volume

For simplicity in the communication volume model, assume that the data flow in the algorithm is from the source to the destination of an edge. A similar analysis can be performed for other cases. Assume that u nodes ($0 \leq u \leq |V|$) are updated in a given round and that the size of the node data (update) is b bytes.

3.2.1 Edge-Cut (EC)

Consider an incoming Edge-Cut (IEC) strategy (1D column partitioning), and let $r_{P,E}(v)$ be the replication factor for a node v. In IEC, the destination of an edge is always a master, so only the master node is updated during computation. At the end of a round, the master node updates its mirrors on other hosts. Since an update requires b bytes, the master node for v in the graph must send $(r_{P,E}(v) - 1) * b$ bytes. If u nodes are updated in the round, the volume of communication is the following:

$$C_{IEC}(G, P) \approx (\overline{r}_{P,E} - 1) * u * b \quad (2)$$

For an outgoing Edge-Cut (OEC) strategy (1D row partitioning), the source of an edge is always a master, so the mirrors must send the updates to their master, and only the master needs the updated value. The communication volume remains the same as IEC's only if all the mirrors of u nodes are updated in the round. In practice, only some mirrors of the nodes in u are updated. Let f_E denote the average fraction of mirrors of nodes in u that are updated. The communication volume in the round is the following:

$$C_{OEC}(G, P) \approx (f_E * \overline{r}_{P,E} - 1) * u * b \quad (3)$$

3.2.2 Hybrid Vertex-Cut (HVC)

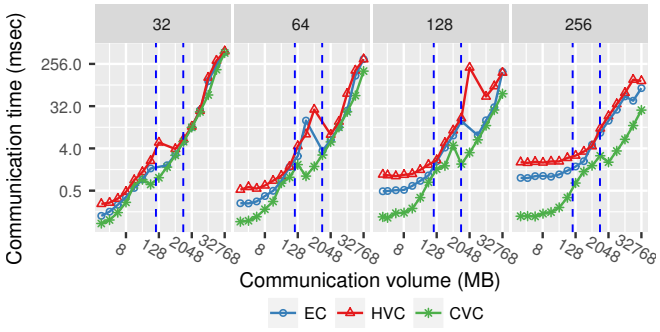
In a general Vertex-Cut strategy like Hybrid Vertex-Cut, the mirrors must communicate with their master, and the masters must communicate with their mirrors. If $\overline{r}_{P,H}$ is the average replication factor, the volume of communication in a round in which an average fraction f_H of mirrors are updated is

$$C_{HVC}(G, P) \approx ((f_H * \overline{r}_{P,H} - 1) + (\overline{r}_{P,H} - 1)) * u * b \quad (4)$$

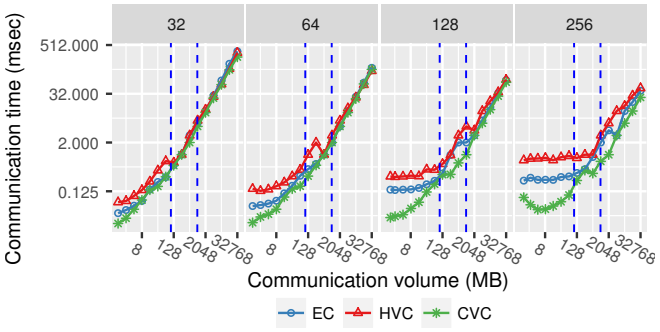
Comparing this with (2) and (3), we see that the volume of communication for OEC and IEC can be much lower than HVC if they have the same replication factor, and it will be greater than HVC only if their replication factor is almost twice that of HVC.

3.2.3 Cartesian Vertex-Cut (CVC)

Consider a Cartesian Vertex-Cut with $P = p_r * p_c$. Unlike HVC, at most p_r mirrors must communicate with their master, and the masters must communicate with at most p_c of their mirrors. Let $\overline{r}_{P,C}$ be the average replication factor, and let f_C and f'_C be the



(a) KNL hosts



(b) Skylake hosts

Figure 3: Performance of different communication patterns on different number of hosts on Stampede for different CPU architectures.

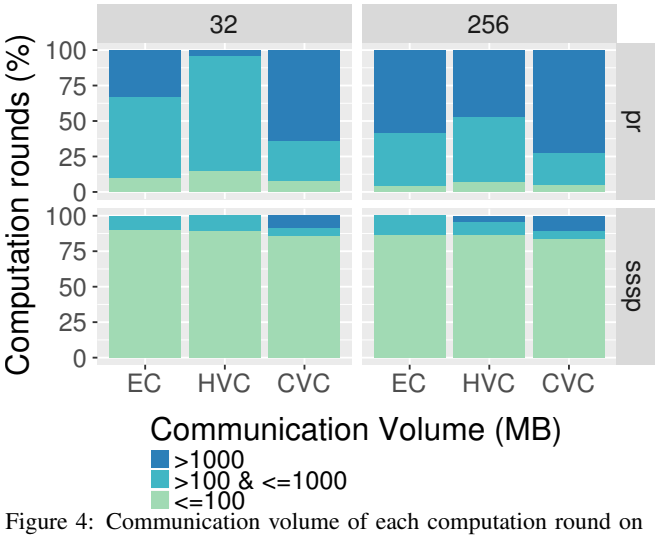


Figure 4: Communication volume of each computation round on clueweb12.

fractions of mirrors that are active along rows and columns respectively. Then, the volume of communication in a round is

$$C_{CVC}(G,P) \approx ((f_C * \overline{r}_{P,C} - 1) + (f'_C * \overline{r}_{P,C} - 1)) * u * b \quad (5)$$

Comparing this with (4), we see that the communication volume for CVC can be less than the volume for HVC even with the same replication factor. This illustrates that the replication factor is not the sole determinant for communication volume.

3.3 Micro-benchmarking to Estimate Communication Time

To relate communication volumes under different communication patterns to communication time, we adapted the MVAPICH2 all-to-allv micro-benchmark. For a given total communication volume across all hosts, we simulate the communication patterns of the three strategies (the message sizes differ for different strategies): (1) IEC/OEC: one round of all-to-all communication among all hosts, (2) HVC: two rounds of all-to-all communication among all hosts, (3) CVC: a round of all-to-all communication among all row hosts followed by a round of all-to-all communication among all column hosts. In a given strategy, the same message size is used between every pair of hosts. In practice, the message sizes usually differ and point-to-point communication (instead of a collective) is typically used to dynamically manage buffers and parallelize serialization and deserialization of data.

Figures 3a and 3b show the communication time of the different strategies on different number of KNL hosts and Skylake hosts, respectively, on the Stampede cluster [43] (described in Section 4) as the total communication volume increases. As expected, the *communication time is dependent not only on the communication volume but also on the communication pattern*. The performance difference in the communication patterns grows as the number of hosts increases. While EC, HVC, and CVC perform similarly on 32 KNL hosts, CVC gives a geomean speedup of $4.6\times$ and $5.6\times$ over EC and HVC respectively to communicate the same volume on 256 KNL hosts; the speedup is higher for low volume ($\leq$ 100MB) than for medium volume ($>$ 100MB and $\leq$ 1000MB). The communication volume in work-efficient graph analytical applications changes over rounds and several rounds have low communication volume in practice. For example, Figure 4 shows the percentage of computation rounds of pagerank and sssp with communication volume in the low, medium, and high ranges for different partitioning policies on 32 and 256 KNL hosts of Stampede. In Figure 3, CVC communicates more data in the same amount of time on 256 KNL hosts; e.g., CVC can synchronize 128 to 256 MB of data in the same time that EC and HVC can synchronize 1 MB of data. Similar behavior is also observed on Skylake hosts. Thus, CVC can reduce communication time over EC and HVC for the same communication volume or can increase the volume without an increase in time.

3.4 Discussion

The analytical model and micro-benchmark results described in this section show that although communication time depends on the communication volume (which in turn depends on the average replication factor), it is incorrect to conclude from this alone that partitioning strategies with larger average replication factors or communication volume will perform worse than those with smaller replication factors or communication volumes. In particular, CVC might be expected to perform better at scale because it requires fewer messages and fewer pairs of processors to communicate than other partitioning strategies like EC and HVC do. While the micro-benchmarks used MPI collectives with the same message sizes, most distributed graph analytics systems use MPI or similar interfaces to perform point-to-point communication with variable message sizes. For example, the D-Galois [15] system uses LCI [14][1] instead of MPI for sending and receiving point-to-point messages between hosts. Even in such systems, CVC can be expected to perform better at scale than EC and HVC because it needs fewer messages and fewer pairs of processors to communicate. We study this using the D-Galois system quantitatively in the next section.

[1]LCI [14] is an alternative to MPI that has been shown to perform better than MPI for graph analytical applications.

Table 1: Inputs and their properties.

	kron30	clueweb12	wdc12
$\|V\|$	1073M	978M	3,563M
$\|E\|$	10,791M	42,574M	128,736M
$\|E\|/\|V\|$	16	44	36
Max D_{out}	3.2M	7,447	55,931
Max D_{in}	3.2M	75M	95M
Size on disk	136GB	325GB	986GB

Table 2: Execution time of Gemini and D-Galois with EC.

	32 hosts	Gemini (sec)	D-Galois (sec)
kron30	**bfs**	7.8	3.0
	cc	16.0	4.8
	pr	213.2	211.6
	sssp	17.5	6.1
clueweb12	**bfs**	72.9	8.9
	cc	38.0	16.9
	pr	231.9	219.6
	sssp	115.8	13.1

Table 3: Different initial Edge-Cut policies used for different benchmarks, inputs, and partitioning policies: IE: Incoming Edge-Cut, OE: Outgoing Edge-Cut, UE: Undirected Edge-Cut.

		bc/bfs/sssp	cc	pr
kron30	**XEC**	OE	UE	IE
	EC	IE	UE	IE
	HVC	IE	UE	IE
	BVC	OE	UE	IE
	JVC	OE	UE	IE
	CVC	OE	UE	IE
clueweb12	**XEC**	OE	UE	IE
	EC	OE	UE	OE
	HVC	OE	UE	OE
	BVC	OE	UE	IE
	JVC	OE	UE	IE
	CVC	OE	UE	IE
wdc12	**XEC**	OE	UE	IE
	EC	OE	UE	OE
	HVC	OE	UE	OE
	BVC	OE	UE	IE
	JVC	OE	UE	IE
	CVC	OE	UE	IE

4. EXPERIMENTAL RESULTS

All experiments were conducted on the Texas Advanced Computing Center's Stampede Cluster (Stampede2) [2, 43]. Stampede2 has 2 distributed clusters: one with Intel Knights Landing (KNL) nodes and another with Intel Skylake nodes. Each KNL cluster host has 68 1.4 GHz cores with 4 hardware threads per core, 96 GB RAM, and 16 GB MC-DRAM, which serves as a direct-mapped L3 cache. Each Skylake cluster host has 48 2.1 GHz cores on 2 sockets (24 cores per socket) with 2 hardware threads per core and 192 GB RAM. Both clusters use 100Gb/sec Intel Omni-Path (OPA) network. We limit our experiments to 256 hosts on both the clusters and we use 272 threads per host (69632 threads in total) on the KNL cluster and 48 threads per host (12288 threads in total) on the Skylake cluster. All code was compiled using g++ 7.1.0. Unless otherwise stated, all results are presented using the KNL cluster.

We used three graphs in our evaluation: synthetically generated randomized power-law graph `kron30` (using kron [31] generator with weights of 0.57, 0.19, 0.19, and 0.05, as suggested by graph500 [1]) and the largest publicly available web-crawl graphs, `clueweb12` [5, 6, 38] and `wdc12` (Web Data Commons) [33, 34]; we present results for wdc12 only at scale (256 hosts). Properties of these graphs are listed in Table 1.

To study the impact of graph partitioning on application execution time, we use five graph analytics applications: betweenness centrality (`bc`), breadth-first search (`bfs`), connected components (`cc`), pagerank (`pr`), and single-source shortest path (`sssp`). We implement a topology-driven algorithm for pagerank and data-driven algorithms for the rest. We run bc with only one source. The source nodes for bc, bfs, and sssp are the maximum out-degree node. The directed graph is given as input to bc, bfs, pagerank, and sssp (bc and sssp use a weighted graph), while an undirected graph (we make the directed graph symmetric by adding reverse edges) is given as input to cc. We present the mean execution time of 3 runs (each being maximum across all hosts) excluding graph partitioning time. *All algorithms are run until convergence except for pagerank, which is run for up to 100 rounds or iterations.*

4.1 Implementation

Existing graph analytics systems implement a single partitioning strategy, so they are not suitable for comparative studies of graph partitioning. Moreover, these frameworks do not optimize communication patterns to exploit structure in the partitioning policy. Therefore, we used a system called D-Galois: it uses the shared-memory Galois system [30, 36] for computing on each host and the Gluon communication runtime [15] for inter-host communication and synchronization. Gluon is an efficient bulk-synchronous communication substrate, which enables existing shared-memory CPU and GPU graph analytics frameworks to run on distributed heterogeneous clusters. Gluon is partition-aware and optimizes communication for particular partitioning strategies by exploiting their *structural invariants*. Instead of naively reducing from mirrors to masters and broadcasting from masters to mirrors during synchronization (as done in gather-apply-scatter model), it avoids redundant reductions or broadcasts by exploiting structural invariants in the partitioning strategy, as described in Section 3. Gluon also exploits the fact that the partitioning of the graph does not change during computation and it uses this *temporal invariance* to reduce the overhead and volume of communication by optimizing metadata like node IDs that needs to be communicated along with the node values or updates.

To ensure that D-Galois is a suitable platform for this study, we compared it with Gemini [51], a state-of-the-art system that uses only Edge-Cuts. Table 2 shows the execution times for Gemini versions of our benchmarks on 32 KNL hosts; wdc12 is omitted because Gemini runs out of memory while partitioning it (even on 256 hosts). Since Gemini supports only Edge-Cuts, we compare it with D-Galois using Edge-Cut (EC). We see that D-Galois outperforms Gemini. Gemini also does not scale beyond 32 hosts [15]. These results support the claim that D-Galois is a reasonable state-of-the-art platform for performing studies of graph partitioning.

For our study, graphs are stored on disk in CSR and CSC formats (size for directed, weighted graphs in Table 1). The synthetic graphs (kron30) are stored after randomizing the vertices (randomized order) while the web-crawl graphs (clueweb12 and wdc12) are stored in their natural (crawled) vertex order. D-Galois's distributed graph partitioner reads the input graph such that each host only reads a distinct portion of the file on disk once.

D-Galois also supports loading partitions directly from disk, and we use this to evaluate XtraPulp [41], the state-of-the-art graph partitioner for large scale-free graphs. XtraPulp generates (using command line parameters "`-e 1.1 -v 10.0 -c -d -s 0`" and 272 OpenMP threads) partitions for kron30 and clueweb12,

Table 4: Graph partitioning time (includes time to load and construct graph) and static load balance of edges assigned to hosts on 256 KNL hosts.

		Partitioning time (sec)			Max-by-mean edges		
		bc bfs sssp	cc	pr	bc bfs sssp	cc	pr
kron30	**XEC**	304	448	312	1.05	1.08	1.06
	EC	51	76	51	1.01	1.01	1.01
	HVC	102	130	101	1.02	1.02	1.02
	BVC	345	379	365	1.02	1.02	1.02
	JVC	1006	1006	1016	1.00	1.00	1.00
	CVC	261	288	241	1.00	1.00	1.00
clueweb12	**XEC**	381	647	373	3.18	8.93	14.69
	EC	27	152	38	1.00	1.11	1.00
	HVC	308	374	308	3.39	1.64	3.39
	BVC	1179	12907	12843	20.24	20.24	20.24
	JVC	1904	1924	1960	1.82	1.53	1.01
	CVC	573	1239	1119	9.16	2.03	3.26
wdc12	**XEC**	OOM	OOM	OOM	OOM	OOM	OOM
	EC	109	251	236	1.00	1.03	1.00
	HVC	3080	2952	3068	1.18	1.13	1.18
	BVC	8039	OOM	OOM	15.44	OOM	OOM
	JVC	5263	6570	8890	1.09	1.05	1.01
	CVC	2487	4276	3221	1.79	1.17	1.27

and it runs out of memory for wdc12 (the authors have been informed about this). We write these partitions to disk and load it in D-Galois. We term this the XtraPulp Edge-Cut (XEC) policy.

We implement five partitioning policies in D-Galois: (1) Edge-Cut (EC), (2) Hybrid Vertex-Cut (HVC), (3) CheckerBoard Vertex-Cut (BVC), (4) Jagged Vertex-Cut (JVC), and (5) Cartesian Vertex-Cut (CVC). Direction of edges traversed during computation is application dependent. bc, bfs, and sssp traverse outgoing edges of a directed graph, pr traverses incoming edges of a directed graph, and cc traverses outgoing edges of a symmetric, directed graph (Gluon handles undirected graphs in this way). Due to this, each application might prefer a different EC policy:

- Outgoing Edge-Cut (OE): Nodes of a directed graph are partitioned into contiguous blocks while trying to balance outgoing edges and all outgoing edges of a node are assigned to the same block as the node, like in Gemini [51].
- Incoming Edge-Cut (IE): Nodes of a directed graph are partitioned into contiguous blocks while balancing incoming edges and all incoming edges are assigned to the same block.
- Undirected Edge-Cut (UE): Nodes of a symmetric, directed graph are partitioned into contiguous blocks while trying to balance outgoing edges and all outgoing edges are assigned to the same block.

EC involves no communication during graph partitioning since each process reads its portion of the graph directly from the file. *All Vertex-Cut policies use some EC*[2] for further partitioning the edges of some vertices, which are communicated to the respective hosts. Table 3 shows the initial Edge-Cut policy used by the different partitioning policies for each benchmark and input. HVC as described in Section 2.3 is used for graphs with skewed in-degree (e.g., wdc12 and clueweb12). For graphs with skewed out-degree (e.g., kron30), if $(n_1 \rightarrow n_2)$ is an edge and n_1 has low out-degree (based on some threshold), the edge is assigned to the host that owns n_1; otherwise, it is assigned to the node that owns n_2. BVC, JVC, and CVC are as described in Section 2.2.

[2]The Vertex-Cut policies could also have used XEC instead of EC to further partition the edges of some vertices, but we chose EC to show that CVC can do well at scale even with a simple Edge-Cut.

Table 5: Dynamic load balance: maximum-by-mean computation time on 256 KNL hosts.

		bc	bfs	cc	pr	sssp
kron30	**XEC**	3.4	1.47	3.49	1.75	1.66
	EC	1.06	1.64	3.60	1.71	1.63
	HVC	1.09	1.70	1.37	1.61	1.57
	BVC	1.16	1.54	1.55	1.19	1.48
	JVC	1.17	1.50	1.55	1.17	1.43
	CVC	1.19	1.45	1.50	1.16	1.36
clueweb12	**XEC**	2.96	2.09	11.81	27.80	2.14
	EC	33.19	2.17	28.76	4.12	2.08
	HVC	8.86	2.12	3.03	7.93	2.26
	BVC	5.04	2.39	19.50	32.46	4.09
	JVC	21.17	2.13	5.69	3.23	2.08
	CVC	7.71	2.20	5.42	6.73	2.65
wdc12	**XEC**	OOM	OOM	OOM	OOM	OOM
	EC	—	2.20	6.31	13.30	2.19
	HVC	—	2.06	2.31	6.32	2.05
	BVC	OOM	1.94	OOM	OOM	OOM
	JVC	—	1.59	1.78	1.81	1.59
	CVC	—	1.67	2.74	6.53	1.75

4.2 Partitioning Time

Although the time to partition graphs is an overhead in distributed-memory graph analytics systems, shared-memory systems must also take time to load the graph from disk and construct it in memory. For example, Galois [36] and other shared-memory systems like Ligra [40] take roughly 2 minutes to load and construct rmat28 (35GB), which is relatively small compared to the graphs used in this study. This time should be used as a point of reference when considering the graph partitioning times shown in Table 4 for different partitioning policies and inputs on 256 KNL hosts. XEC excludes time to write partitions from XtraPulp to disk, load it in D-Galois, and construct the graph. All other policies include graph loading and construction time. Note that some partitioning policies run out-of-memory (OOM). EC is a lower bound for partitioning as each host loads its partition of the graph directly from disk in parallel. Vertex-Cut policies are slower than Edge-Cut policies because they involve communication and more analysis on top of EC.

Comparing partitioning time for different partitioning strategies is not the focus of this work. In particular, graphs can be partitioned once, and different applications can run using these partitions. The main takeaway for partitioning time is that it can finish within a few minutes. CVC partitioning time is better than or similar to that of XEC. CVC takes around 5 minutes, 10 minutes, and 40 minutes for directed kron30, clueweb12, and wdc12, respectively.

4.3 Load Balance

We first compare the quality of the resulting partitions. The number of edges assigned to a host represents the static load of that host. Table 4 shows the maximum to mean ratio of edges assigned to hosts. We see that static load balance not only varies with different policies but also with the input graphs. For both directed and undirected graphs, kron30 is very well balanced for all policies. For clueweb12 and wdc12, EC is well balanced while the rest are not.

Dynamic load may be quite different from static load, especially for data-driven algorithms in which computation changes over rounds. To measure the dynamic load balance, we measure the computation time of each round on each host and calculate the maximum and mean across hosts. Summing up over rounds, we get the maximum and mean computation time respectively. Table 5 shows the

Table 6: Execution time (sec) for different partitioning policies, benchmarks, and inputs on KNL hosts.

		32 hosts						256 hosts					
		XEC	EC	HVC	BVC	JVC	CVC	XEC	EC	HVC	BVC	JVC	CVC
kron30	bc	32.7	40.9	51.7	75.5	26.0	**22.6**	23.6	39.1	51.2	20.4	13.6	**10.0**
	bfs	4.6	3.0	4.4	5.1	**2.6**	2.9	3.8	2.4	4.0	1.3	1.3	**1.0**
	cc	10.1	4.8	10.7	13.9	5.6	**4.2**	4.3	4.2	3.9	2.4	2.2	**1.9**
	pr	230.1	211.6	293.0	287.8	**191.9**	339.9	**57.6**	64.2	84.1	65.7	78.7	72.9
	sssp	9.7	6.1	8.3	10.4	5.6	**4.8**	5.4	3.8	5.9	2.0	2.0	**1.8**
clueweb12	bc	**136.2**	439.1	627.9	961.9	660.6	539.1	413.7	420.0	1012.1	404.6	627.6	**266.9**
	bfs	10.4	**8.9**	17.4	41.1	38.7	19.2	36.4	27.1	46.5	25.1	36.1	**14.6**
	cc	OOM	16.9	**7.5**	OOM	25.9	19.6	43.0	84.7	8.4	21.2	12.5	**7.3**
	pr	272.6	219.6	**193.5**	OOM	354.6	217.9	286.7	82.3	97.5	267.4	**58.6**	60.8
	sssp	16.5	**13.1**	26.6	63.7	63.2	31.7	43.0	32.5	54.7	44.5	44.7	**21.8**

Table 7: Execution statistics of wdc12 on 256 KNL hosts.

		EC	HVC	JVC	CVC
Execution Time (sec)	bfs	422.8	832.9	974.6	**373.4**
	cc	118.8	135.8	178.4	**74.2**
	pr	230.9	193.1	173.3	**138.3**
	sssp	633.1	1238.3	1395.6	**567.9**
Replication Factor	bfs	1.4	2	4.6	3.4
	cc	4.4	2.3	7.2	5.3
	pr	1.4	2.0	5	3.1
	sssp	1.4	2	4.6	3.4
Total Communication Volume (GB)	bfs	15	27	101	54
	cc	100	36	278	147
	pr	604	628	3019	1394
	sssp	66	159	697	352

Table 8: Execution time (sec) on Skylake hosts for EC and CVC.

		8 hosts		256 hosts	
		EC	CVC	EC	CVC
kron30	bc	45.9	**35.9**	21.1	**5.5**
	bfs	**3.3**	3.4	4.0	**0.7**
	cc	**8.2**	9.4	7.1	**1.0**
	pr	**230.3**	259.2	38.1	**28.4**
	sssp	**6.3**	7.1	6.0	**1.2**
clueweb12	bc	**339.1**	669.2	197.7	**152.4**
	bfs	**6.0**	16.1	6.1	**6.0**
	cc	10.9	**9.5**	64.9	**3.6**
	pr	**121.5**	130.1	20.6	**17.9**
	sssp	**16.9**	33.2	**8.6**	9.7

maximum-by-mean computation time for all policies, benchmarks, and inputs on 256 hosts. Note that bc, bfs, and sssp use the same partitions. Firstly, it is clear that although kron30 is statically well balanced for all policies, it is not dynamically load balanced. Moreover, the load balance depends on the dynamic nature of the algorithm. Even though bc and bfs use the same graph, their dynamic load balance is different. CVC on clueweb12 is well-balanced for bfs, but highly imbalanced for bc. The policy significantly impacts dynamic load balance, but this need not directly correlate with their static load balance. For example, for bfs and sssp, although CVC on clueweb12 is statically severely imbalanced while EC is not, both EC and CVC are fairly well balanced at runtime. This demonstrates that dynamic load balance is difficult to achieve since it depends on the interplay between policy, algorithm, and graph.

4.4 Execution Time

Table 6 shows the execution time of D-Galois with all policies on 32 and 256 KNL hosts for kron30 and clueweb12. Table 7 shows the execution time for wdc12 on 256 KNL hosts (all policies run out of memory on 32 hosts; XEC and BVC run out of memory on 256 hosts too). These tables also highlight which policy performs best for a given application, input, and number of hosts (scale). It is clear that the best performing policy is dependent on the application, the input, and the number of hosts (scale). Although there is no clear winner for all cases, CVC performs the best on 256 hosts for almost all applications and inputs.

Table 8 shows the execution time of D-Galois with EC and CVC on 8 and 256 Skylake hosts for kron30 and clueweb12. Even on these Skylake hosts, the best performing policy depends on the application, the input, and the number of hosts (scale), without any clear winner. Nonetheless, similar to KNL hosts, CVC performs the best on 256 hosts for almost all applications and inputs, whereas EC performs the best on 8 hosts for almost all applications and inputs. This suggests that the relative merits of the partitioning policies are not specific to the CPU architecture.

4.5 Strong Scaling

We measure the time for computation of each round or iteration on each host and take the maximum across hosts. Summing up over iterations, we get the computation time. Figure 5 shows the strong scaling of execution time (left) and computation time (right) for kron30 and clueweb12 (most partitioning policies run out of memory for wdc12 on less than 256 hosts). Some general trends are apparent. At small scale on 32 hosts, EC performs fairly well for almost all the benchmarks and is comparable to the best, but as we go to higher number of hosts, EC, XEC, and HVC do not scale. On the other hand, all 2D partitioning policies scale relatively better for almost all benchmarks and inputs. Among them, JVC scales better than BVC, and CVC scales better than JVC. In most cases, CVC has the best performance at scale and scales best.

CVC does not scale well for bfs and sssp on clueweb12 due to computation time not scaling well. In all other cases, both computation time and execution time scale. For bfs and sssp, CVC is still better than the others in execution time since computation times of all policies do not scale. This is likely due to the computation being too small for it to scale (both execute more than 180 iterations, so each iteration has little to compute on each host at scale).

Computation time of all policies scales similarly in most cases. However, their execution time scaling differs. This is most evident

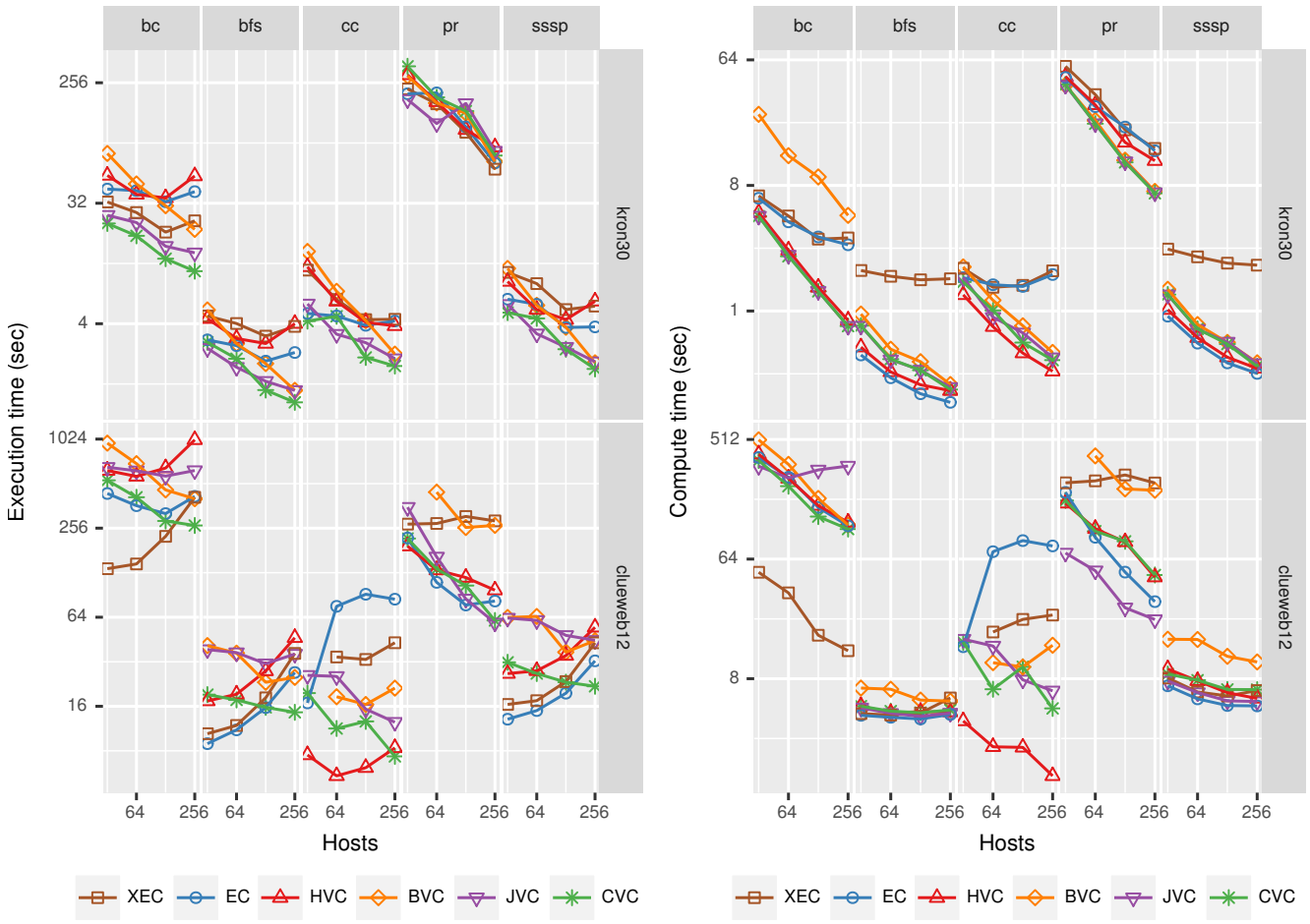


Figure 5: Execution time (left) and compute time (right) of D-Galois with different partitioning policies on KNL hosts.

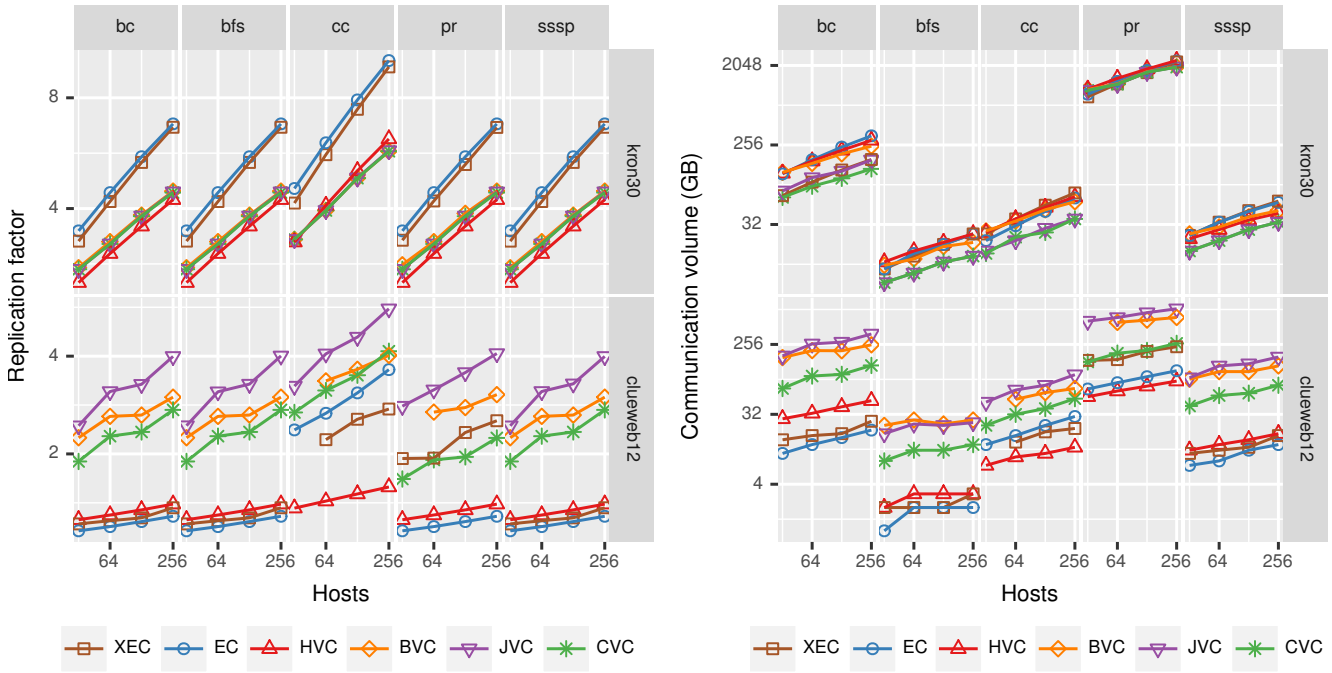


Figure 6: Replication factor (left) and communication volume (right) of D-Galois with different partitioning policies on KNL hosts.

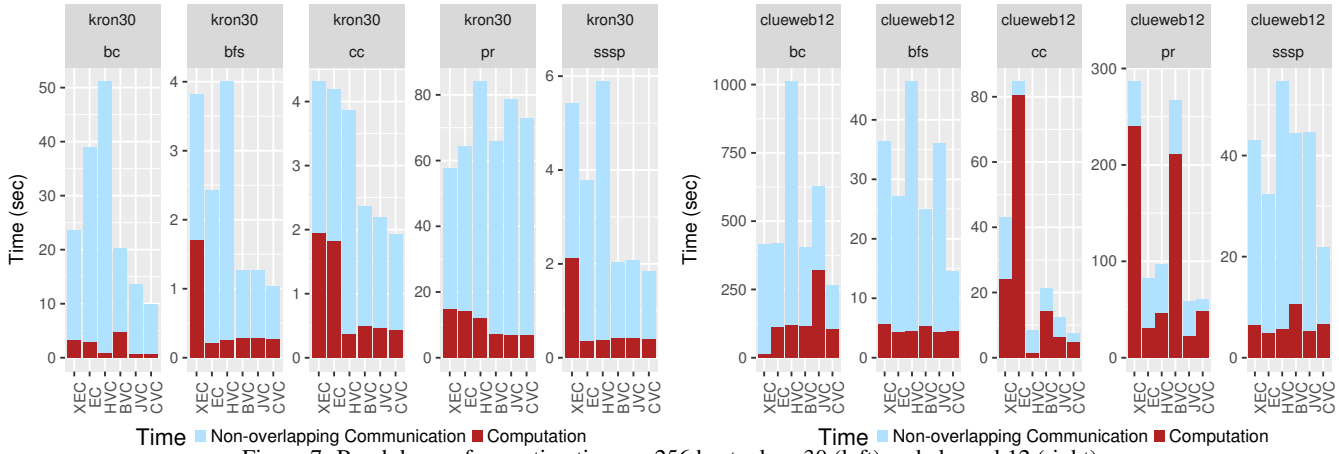


Figure 7: Breakdown of execution time on 256 hosts: kron30 (left) and clueweb12 (right).

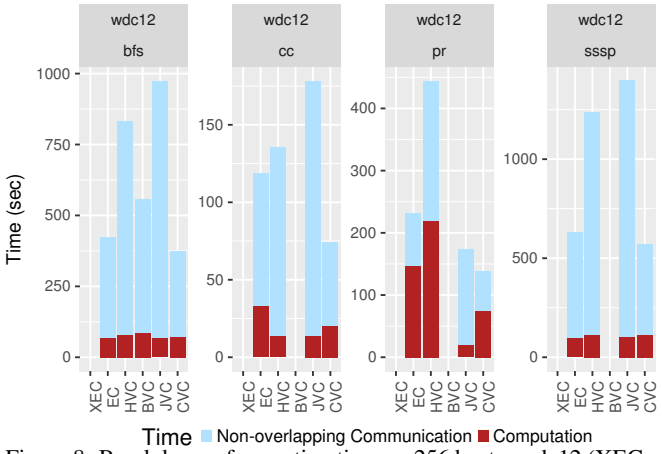


Figure 8: Breakdown of execution time on 256 hosts: wdc12 (XEC and BVC run out-of-memory).

in EC and CVC. This indicates that the difference in the policies arises due to the communication. We analyze this next.

4.6 Communication Analysis

The total communication volume is the sum of the number of bytes sent from one host to another during execution. Table 7 shows the replication factor and total communication volume for wdc12 on 256 hosts (XEC and BVC run out of memory). Figure 6 shows how the replication factor (left) and the total communication volume (right) scale as we increase the number of hosts for kron30 and clueweb12. The replication factor increases with the number of hosts for all partitioning policies as expected. Similarly, the total communication volume increases with the number of hosts as more data needs to be exchanged to synchronize those proxy nodes across hosts. However, the difference in replication factor across policies can vary from that of communication volume. This is most evident for kron30: although EC has a much higher replication factor than the other policies, the communication volume of EC is close to that of others. This demonstrates that replication factor need not be the sole determinant for the communication volume.

For kron30, EC corresponds to IEC, so there are no updates sent from mirrors to masters; only masters send updates to mirrors. Vertex-Cut policies like HVC and CVC, however, send from mirrors to masters and then from masters to mirrors. Thus, if EC has roughly twice the replication factor of HVC, EC would still communicate the same volume as HVC. This can analyzed using

Table 9: Communication volume estimated by the model vs. observed communication volume on 128 KNL hosts for kron30.

		IEC	HVC	CVC
bfs	**Replication Factor**	5.53	**3.58**	3.81
	Estimated Volume(GB)	15.63	19.73	**2.82**
	Observed Volume(GB)	19.26	20.29	**11.69**
cc	**Replication Factor**	7.89	5.07	**4.84**
	Estimated Volume(GB)	30.07	71.02	**20.44**
	Observed Volume(GB)	44.51	50.40	**26.08**
pr	**Replication Factor**	5.53	**3.58**	3.81
	Estimated Volume(GB)	1153.56	1243	**932.06**
	Observed Volume(GB)	1797.30	1867.06	**1715.68**
sssp	**Replication Factor**	5.53	**3.58**	3.81
	Estimated Volume(GB)	36.97	40.60	**6.99**
	Observed Volume(GB)	44.63	35.55	**27.64**

our analytical model in Section 3.2. We estimate the communication volume for EC, HVC, and CVC using Equations 2, 4, and 5, respectively. The values to use for the replication factor $\overline{r}$ and the size of the data b are straightforward to determine. We assume that if a node is updated, all its mirrors are updated, so we use a value of 1 for f. These equations estimate the volume for a given round or iteration. To get the total communication volume, we can sum over all rounds. We instead replace the number of updates u in a round with the total number of updates performed on the graph across all the rounds to estimate the total communication volume.

Table 9 presents the estimated communication volume of different applications and policies for kron30 along with the replication factor and the observed communication volume. The estimated volume can be more than the observed volume because we assume f is 1, which is an over-approximation. The observed volume can be higher than the estimated volume as the estimation does not account for the metadata like node IDs communicated along with node values or updates, which is an under-approximation. Such approximations are fine because the relative ordering of the communication volume among different partitioning policies is important, not the absolute values. From the estimated volume for all the benchmarks, we can see that our analytical model predicts CVC to communicate the least amount of volume even if CVC has higher replication factor than HVC. A similar pattern can also be seen in the observed communication volume for all the benchmarks, where CVC has the

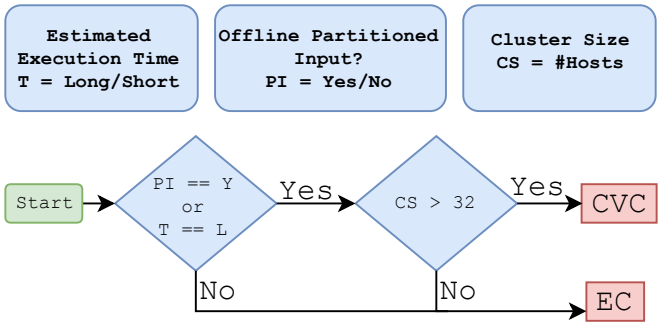


Figure 9: Decision tree to choose a partitioning policy.

minimum volume but not necessarily the minimum replication factor. This validates our analytical model, stating that the replication factor is not the sole determinant for communication volume.

In Figure 6, we see that CVC may have a higher replication factor and communication volume than the other policies, yet it performs better than the other policies in most cases. Figures 7 and 8 show the breakdown of execution time into computation time and non-overlapped communication time (the rest of the execution time) on 256 hosts. It is clear that CVC is doing better (except pagerank on kron30) because the communication time is lower. We can see that more communication volume does not always imply more communication time. For example, in pr and sssp on clueweb12, CVC has higher replication factor and more communication volume than EC and HVC but lower communication time. Figure 4 shows the percentage of rounds in those applications and policies that have low ($\leq$ 100MB), medium ($>$ 100MB and $\leq$ 1000MB), and high ($>$ 1000MB) communication volume. CVC increases the number of high volume rounds of pr and sssp over EC on both 32 and 256 hosts. Our micro-benchmarking in Section 3.3 shows that CVC yields significant speedups over EC on 256 hosts in both low and high volumes, which outweighs the increase in high volume rounds. In contrast, the increase in high volume rounds on 32 hosts for CVC over EC causes a slowdown in communication time since there is very little difference between CVC and EC at this scale for the same communication volume. This validates the claim that the communication time depends on both the communication volume and the communication pattern and shows that CVC has much less communication overhead than other policies at large scale.

5. CHOOSING A PARTITIONING

Based on the results presented in Section 4, we present a decision tree (Figure 9) to help users choose the most suitable partitioning strategy based on the following parameters:

1. Whether the input is partitioned offline: The time it takes to partition and load the graph (online) depends on the complexity of the partitioning strategy. EC takes least amount of time, whereas strategies like XEC [41] and CVC [7] take more time as they involve analysis and communication during partitioning. It makes sense to use complex partitioning strategies if the benefits gained from them outweigh the time spent in partitioning. D-Galois also supports direct loading of offline partitioned graphs, in which case partitioning time is not a factor as the partitioned graph is on disk.

2. Whether the execution time is estimated to be long or short: The amount of time spent in execution of the application plays a vital role in determining if it makes sense to invest time in a good partitioning strategy. Spending more time in partitioning makes more sense for long-running applications as they involve more rounds of communication which can benefit from a well-chosen partitioning strategy. The user can easily identify whether the application is expected to run for a long time (e.g., by using algorithm complexity). Applications such as bc tend to be more complex as they have multiple phases within each round of computation, and they involve floating-point operations; therefore, they are expected to have longer execution times. On the other hand, applications like bfs are relatively less complex and can be classified as short-running applications.

3. Cluster size: Our results show that the performance of different partitioning strategies also depend on the number of hosts on which the application is run.

Table 10: % difference in execution time (excluding partitioning time) on KNL hosts between the partitioning strategy chosen by the decision tree and the optimal one (wdc12 is omitted because chosen one is always optimal; 0% means that chosen one is optimal).

		32	64	128	256
kron30	**bc**	44.74%	0%	0%	0%
	bfs	13.33%	13.68%	0%	0%
	cc	12.5%	26.63%	0%	0%
	pr	9.31%	36.38%	30.63%	20.99%
	sssp	21.31%	23.26%	0%	0%
clueweb12	**bc**	68.98%	60.59%	21.32%	0%
	bfs	0%	37.24%	0%	0%
	cc	55.62%	52.16%	51.35%	0%
	pr	11.89%	18.69%	26.09%	3.62%
	sssp	0%	43.83%	17.33%	0%

Table 11: % difference in execution time (excluding partitioning time) on Skylake hosts between the partitioning strategy chosen by the decision tree and the optimal one.

		8	256
kron30	**bc**	21.79%	0%
	bfs	0%	0%
	cc	0%	0%
	pr	0%	0%
	sssp	0%	0%
clueweb12	**bc**	0%	0%
	bfs	0%	0%
	cc	12.84%	0%
	pr	0%	0%
	sssp	0%	11.34%

Figure 9 illustrates our decision tree to choose a partitioning strategy. For short running applications, if the graph is not already partitioned, we recommend using simple EC. Additionally, if the graph is already partitioned and the number of hosts is less than 32, simple EC should suffice. For long-running applications, the decision to use EC or CVC primarily depends on the cluster size. If the number of machines is more than 32, it makes sense to invest partitioning time in CVC. Otherwise, EC is recommended.

Choosing the best partitioning strategy is a difficult problem as it depends on several factors such as properties of input graphs, applications, number of hosts (scale), etc. Therefore, the decision tree may not always suggest or choose the best partitioning strategy. Tables 10 and 11 illustrate this point by showing the percentage difference in the application execution time between the chosen

partitioning strategy and the best-performing or optimal strategy at different number of hosts assuming the input graphs used are already partitioned (i.e., graph construction time for all strategies are same). A value of zero means that the chosen partitioning strategy performs the best. In many cases, the chosen strategy performs best, particularly at 128 and 256 hosts. For kron30, this difference in most of the cases is under 20%. For clueweb12, the difference is slightly higher, especially for bc at 32 hosts, for which XEC performs best rather than simple EC (XEC uses a community detection technique for partitioning which provides compute locality for the compute-heavy bc algorithm). Nonetheless, the decision tree chooses a partitioning strategy that performs well in most cases.

6. RELATED WORK

Several distributed-memory graph processing frameworks have been published in the past few years [9, 12, 13, 17–21, 25, 28, 32, 35, 46–49, 51]. These systems use graph partitioning to scale out computations on graphs or sparse matrices that do not fit in the memory of a single node. In the graph analytics literature, partitioning strategies are classified into Edge-Cuts [4, 22–24, 41, 42, 50, 51] and Vertex-Cuts [8,12,18,26,29,37,39,44]. In the matrix literature, they are classified into 1D and 2D partitionings [7, 11]. 1D partitionings are equivalent to the class of Edge-Cuts, whereas 2D partitionings are strictly a sub-class of Vertex-Cuts as they are more restricted.

1D partitionings or Edge-Cuts: METIS [4,23] and XtraPulp [41] partition the graph based on connected components. XtraPulp has been shown to partition large graphs in a few minutes, but they do not compare against general Vertex-Cuts. Streaming Edge-Cut policies [42, 51] partition the graph in a pass or two over the edges. This paper evaluates XtraPulp and edge-balanced Edge-Cuts to represent non-streaming and streaming Edge-Cuts, respectively.

2D partitionings: 2D partitionings have been studied in both dense matrix and sparse matrix communities [27]. CheckerBoard 2D partitioning (BVC) [11] is used in CombBLAS, a sparse matrix library. Jagged-like partitioning (JVC) [11] and Cartesian Vertex-Cut (CVC) [7] have been evaluated for generalized sparse matrix vector computation. However, these strategies have never been evaluated on work-efficient data-driven graph algorithms, and there are no comparisons with other policies like Hybrid Vertex-Cut [12].

Vertex-Cuts that are neither 1D nor 2D partitionings: PowerGraph [18] is the first graph analytical system to develop a streaming Vertex-Cut partitioning heuristic targeting power-law graphs. PowerLyra [12] proposed a streaming Vertex-Cut heuristic called Hybrid Vertex-Cut (HVC) that handles high-degree nodes differently from low-degree nodes. Bourse et al. [8] analyze balanced Vertex-Cut partitions theoretically and propose a least incremental cost (LIC) heuristic with approximation guarantees. Petroni et al. [37] proposed High-Degree (are) Replicated First (HDRF), a novel streaming Vertex-Cut graph partitioning algorithm that exploits skewed degree distributions by explicitly taking into account vertex degree in the placement decision. These papers do not compare their approaches to 2D block partitionings.

Studies of partitioning policies: There are several studies [3, 16, 29, 45] that have compared the impact of partitioning strategies on application execution time. Yun et al [16] compares various distributed graph analytics systems on different design aspects including graph distribution policies and concludes that the Vertex-Cut partitioning strategy always outperforms the Edge-Cut on vertex (neighbor-based) programs, which is not the case as shown by this paper. LeBeane et al. [29] studies the impact of relative computational throughput of hosts in heterogeneous setting on various partitioning strategies for graph analytics workloads using PowerGraph. Verma et al. [45] evaluates different partitioning strategies provided by distributed graph analytics systems, namely, PowerGraph, GraphX, and PowerLyra, and it suggests the best partitioning strategy for each system among the strategies provided by that system. These studies were done at a very small scale of 10 to 25 hosts. Verma et al. [45] also compares various partitioning strategies on PowerLyra. However, PowerLyra does not optimize communication for the non-native partitioning strategies adopted from other systems. In a recent study, Abbas et al. [3] compares various streaming partitioning policies using a distributed runtime based on Apache Flink [10] and concludes that low-cut algorithms (with low replication factor) perform better for communication-intensive applications. However, the study was done on a small 17 host cluster, and the largest graph considered was Friendster, which easily fits in the memory of a single host in the cluster used in our study.

To the best of our knowledge, no previous study performs a quantitative comparison of partitioning strategies with communication optimized for each partitioning strategy at scale for work-efficient graph analytics applications. In this paper, we used D-Galois, an efficient distributed-memory graph processing system based on Gluon runtime [15] that optimizes communication specifically for each partitioning strategy, on 256 KNL hosts with a total of 69K threads. However, our study and observations are not limited to D-Galois, and they should generalize to other systems that optimize communication based on the partitioning strategy.

7. CONCLUSIONS

This paper presented a detailed performance study of graph partitioning strategies, including Edge-Cuts, 2D block partitioning strategies, and general Vertex-Cuts, using state-of-the-art, work-efficient graph analytics algorithms on a large-scale cluster with 256 hosts and roughly 69K threads. The experiments used the D-Galois system, a distributed-memory version of the Galois system [36] based on the Gluon runtime [15], which implements partitioning-specific communication optimizations.

A key lesson for designers of high-performance graph analytics systems from our study is that these systems must support optimized communication for different partitioning strategies like the Gluon runtime. Our results clearly show that the best-performing or optimal partitioning strategy depends on the application, the input, and the number of hosts or scale. We presented a simple decision tree that helps the user to choose a partitioning strategy for a particular combination of these. We showed that the partitioning strategy thus chosen performs well in most cases.

Our results show that although Edge-Cuts perform well on small-scale clusters, a 2D partitioning strategy called Cartesian Vertex-Cut [7] performs significantly better on large clusters even though it has a higher replication factor and a higher communication volume than other partitioning strategies. The main reason is that CVC requires fewer pairs of processors to communicate, permitting CVC to communicate larger volumes of data with less overhead. Another important lesson for designers of efficient graph partitioning policies from our study is that replication factor and the number of edges/vertices split between partitions are not adequate proxies for communication overhead during application execution.

Acknowledgments

This research was supported by NSF grants 1337217, 1337281, 1406355, 1618425, 1725322 and by DARPA contracts FA8750-16-2-0004 and FA8650-15-C-7563. This work used XSEDE grant ACI-1548562 through allocation TG-CIE170005. We used the Stampede system at Texas Advanced Computing Center.

8. REFERENCES

[1] http://www.graph500.org.

[2] Texas Advanced Computing Center (TACC), The University of Texas at Austin, 2018.

[3] Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov. Streaming Graph Partitioning: An Experimental Study. *Proc. VLDB Endow.*, 11(11):1590–1603, July 2018.

[4] A. Abou-Rjeili and G. Karypis. Multilevel Algorithms for Partitioning Power-law Graphs. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, IPDPS'06, pages 124–124, Washington, DC, USA, 2006. IEEE Computer Society.

[5] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM Press, 2011.

[6] P. Boldi and S. Vigna. The WebGraph Framework I: Compression Techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[7] E. G. Boman, K. D. Devine, and S. Rajamanickam. Scalable matrix computations on large scale-free graphs using 2D graph partitioning. In *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, Nov 2013.

[8] F. Bourse, M. Lelarge, and M. Vojnovic. Balanced Graph Edge Partition. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1456–1465, New York, NY, USA, 2014. ACM.

[9] A. Buluc and J. R. Gilbert. The Combinatorial BLAS: Design, Implementation, and Applications. *Int. J. High Perform. Comput. Appl.*, 25(4):496–509, Nov. 2011.

[10] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas. State management in apache flink®: Consistent stateful distributed stream processing. *Proc. VLDB Endow.*, 10(12):1718–1729, Aug. 2017.

[11] U. V. Çatalyürek, C. Aykanat, and B. Uçar. On Two-Dimensional Sparse Matrix Partitioning: Models, Methods, and a Recipe. *SIAM J. Sci. Comput.*, 32(2):656–683, Feb. 2010.

[12] R. Chen, J. Shi, Y. Chen, and H. Chen. Powerlyra: Differentiated graph computation and partitioning on skewed graphs. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 1:1–1:15, New York, NY, USA, 2015. ACM.

[13] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proc. VLDB Endow.*, 8(12):1804–1815, Aug. 2015.

[14] H.-V. Dang, R. Dathathri, G. Gill, A. Brooks, N. Dryden, A. Lenharth, L. Hoang, K. Pingali, and M. Snir. A lightweight communication runtime for distributed graph analytics. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2018.

[15] R. Dathathri, G. Gill, L. Hoang, H.-V. Dang, A. Brooks, N. Dryden, M. Snir, and K. Pingali. Gluon: A Communication Optimizing Framework for Distributed Heterogeneous Graph Analytics. PLDI, 2018.

[16] Y. Gao, W. Zhou, J. Han, D. Meng, Z. Zhang, and Z. Xu. An Evaluation and Analysis of Graph Processing Frameworks on Five Key Issues. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, CF '15, pages 11:1–11:8, New York, NY, USA, 2015. ACM.

[17] Apache Giraph. http://giraph.apache.org/, 2013.

[18] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. PowerGraph: Distributed Graph-parallel Computation on Natural Graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 17–30, Berkeley, CA, USA, 2012. USENIX Association.

[19] S. Hong, S. Depner, T. Manhardt, J. Van Der Lugt, M. Verstraaten, and H. Chafi. PGX.D: A Fast Distributed Graph Processing Engine. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 58:1–58:12, New York, NY, USA, 2015. ACM.

[20] I. Hoque and I. Gupta. LFGraph: Simple and Fast Distributed Graph Analytics. In *Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems*, TRIOS '13, pages 9:1–9:17, New York, NY, USA, 2013. ACM.

[21] N. Jain, G. Liao, and T. L. Willke. GraphBuilder: Scalable Graph ETL Framework. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, pages 4:1–4:6, New York, NY, USA, 2013. ACM.

[22] G. Karypis and V. Kumar. Parallel Multilevel K-way Partitioning Scheme for Irregular Graphs. In *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, Supercomputing '96, Washington, DC, USA, 1996. IEEE Computer Society.

[23] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, Dec. 1998.

[24] G. Karypis and V. Kumar. Multilevel K-way Hypergraph Partitioning. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, DAC '99, pages 343–348, New York, NY, USA, 1999. ACM.

[25] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis. Mizan: A System for Dynamic Load Balancing in Large-scale Graph Processing. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 169–182, New York, NY, USA, 2013. ACM.

[26] M. Kim and K. S. Candan. SBV-Cut: Vertex-cut Based Graph Partitioning Using Structural Balance Vertices. *Data Knowl. Eng.*, 72:285–303, Feb. 2012.

[27] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[28] M. S. Lam, S. Guo, and J. Seo. SociaLite: Datalog Extensions for Efficient Social Network Analysis. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pages 278–289, Washington, DC, USA, 2013. IEEE Computer Society.

[29] M. LeBeane, S. Song, R. Panda, J. H. Ryoo, and L. K. John. Data Partitioning Strategies for Graph Workloads on Heterogeneous Clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 56:1–56:12, New York,

NY, USA, 2015. ACM.

[30] A. Lenharth, D. Nguyen, and K. Pingali. Parallel Graph Analytics. *Commun. ACM*, 59(5):78–87, Apr. 2016.

[31] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker Graphs: An Approach to Modeling Networks. *J. Mach. Learn. Res.*, 11:985–1042, Mar. 2010.

[32] 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 *Proc. ACM SIGMOD Intl Conf. on Management of Data*, SIGMOD '10, pages 135–146, 2010.

[33] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer. Web Data Commons - Hyperlink Graphs, 2012.

[34] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer. Graph Structure in the Web — Revisited: A Trick of the Heavy Tail. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 427–432, New York, NY, USA, 2014. ACM.

[35] J. Nelson, B. Holt, B. Myers, P. Briggs, L. Ceze, S. Kahan, and M. Oskin. Latency-tolerant Software Distributed Shared Memory. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, pages 291–305, Berkeley, CA, USA, 2015. USENIX Association.

[36] D. Nguyen, A. Lenharth, and K. Pingali. A Lightweight Infrastructure for Graph Analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 456–471, New York, NY, USA, 2013. ACM.

[37] 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*, CIKM '15, pages 243–252, New York, NY, USA, 2015. ACM.

[38] T. L. Project. The ClueWeb12 Dataset, 2013.

[39] Z. Shi, J. Li, P. Guo, S. Li, D. Feng, and Y. Su. Partitioning Dynamic Graph Asynchronously with Distributed FENNEL. *Future Gener. Comput. Syst.*, 71(C):32–42, June 2017.

[40] J. Shun and G. E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, PPoPP '13, pages 135–146, 2013.

[41] G. M. Slota, S. Rajamanickam, K. Devine, and K. Madduri. Partitioning Trillion-Edge Graphs in Minutes. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 646–655, May 2017.

[42] 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*, KDD '12, pages 1222–1230, New York, NY, USA, 2012. ACM.

[43] D. Stanzione, B. Barth, N. Gaffney, K. Gaither, C. Hempel, T. Minyard, S. Mehringer, E. Wernert, H. Tufo, D. Panda, and P. Teller. Stampede 2: The Evolution of an XSEDE Supercomputer. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, PEARC17, pages 15:1–15:8, New York, NY, USA, 2017. ACM.

[44] 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*, WSDM '14, pages 333–342, New York, NY, USA, 2014. ACM.

[45] S. Verma, L. M. Leslie, Y. Shin, and I. Gupta. An Experimental Comparison of Partitioning Strategies in Distributed Graph Processing. *Proc. VLDB Endow.*, 10(5):493–504, Jan. 2017.

[46] K. Vora, S. C. Koduru, and R. Gupta. ASPIRE: Exploiting Asynchronous Parallelism in Iterative Algorithms Using a Relaxed Consistency Based DSM. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pages 861–878, New York, NY, USA, 2014. ACM.

[47] W. Xiao, J. Xue, Y. Miao, Z. Li, C. Chen, M. Wu, W. Li, and L. Zhou. Tux²: Distributed Graph Computation for Machine Learning. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 669–682, Boston, MA, 2017. USENIX Association.

[48] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica. GraphX: A Resilient Distributed Graph System on Spark. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, 2013.

[49] F. Yang, M. Wu, J. Xue, W. Xiao, Y. Miao, L. Wei, H. Lin, Y. Dai, and L. Zhou. GraM: Scaling Graph Computation to the Trillions. In *SoCC*. ACM – Association for Computing Machinery, August 2015.

[50] Y. Zhang, Y. Liu, J. Yu, P. Liu, and L. Guo. VSEP: A Distributed Algorithm for Graph Edge Partitioning. In *Proceedings of the ICA3PP International Workshops and Symposiums on Algorithms and Architectures for Parallel Processing - Volume 9532*, pages 71–84, Berlin, Heidelberg, 2015. Springer-Verlag.

[51] X. Zhu, W. Chen, W. Zheng, and X. Ma. Gemini: A Computation-centric Distributed Graph Processing System. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 301–316, Berkeley, CA, USA, 2016. USENIX Association.