

An Experimental Comparison of Partitioning Strategies in Distributed Graph Processing

26-Feb-2019

Presented by: Noshin Nawar Sadat

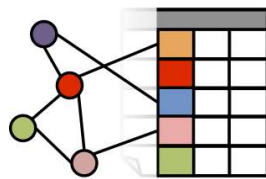


UNIVERSITY OF
WATERLOO

Graph Processing Systems

Pregel

GraphLab

 *GraphX*


powergraph

 A P A C H E
G I R A P H


PowerLyra

Graph Partitioning

Graph processing systems run in distributed manner to handle large graphs.

To distribute computations, partition graphs by assigning graph edges or vertices to individual machines.

The Problem

Plethora of graph processing systems.

Each offer their own graph partitioning strategies.

Even after user chooses a system, there are different partitioning strategies to choose from.

The Problem

Choosing the right partitioning strategy for
distributed graph processing systems

Why is the problem important?

Choice of partitioning strategy affects

- Performance
 - Replication factor
 - Completion time (Ingress/Partitioning time, Computation time)
- Resource usage
 - Memory usage
 - Network I/O

Paper Contributions

- Experimentally compared partitioning strategies of three popular distributed graph processing (vertex-cut based) systems:
 - PowerGraph
 - PowerLyra
 - GraphX
- Presented rules of thumb to help pick partitioning strategies within each system for different use cases.
- Implemented PowerGraph and GraphX's strategies into PowerLyra

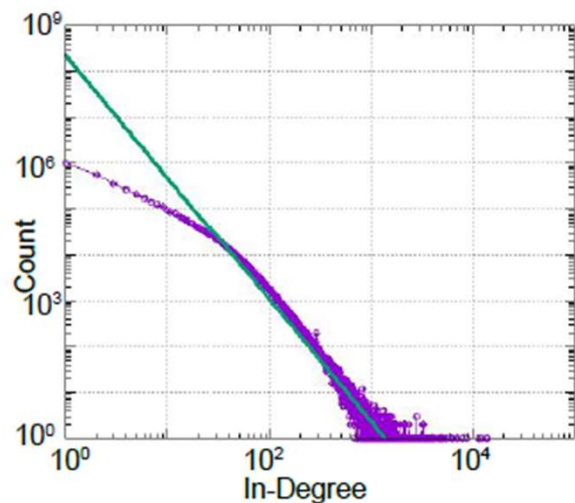
Experimental Methodology : Clusters

- PowerGraph and PowerLyra
 - A local cluster of 9 machines
 - EC2 consisting of 16 m4.2xlarge instances
 - EC2 consisting of 25 m4.2xlarge instances
- GraphX
 - Local cluster of 10 machines

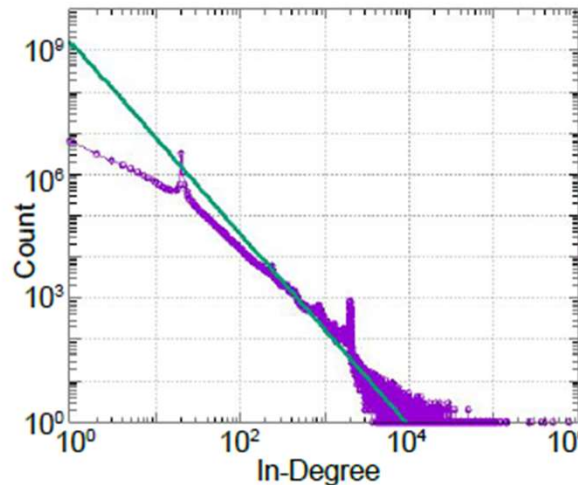
Grid requires number of partitions to be a perfect square.

Experimental Methodology : Datasets

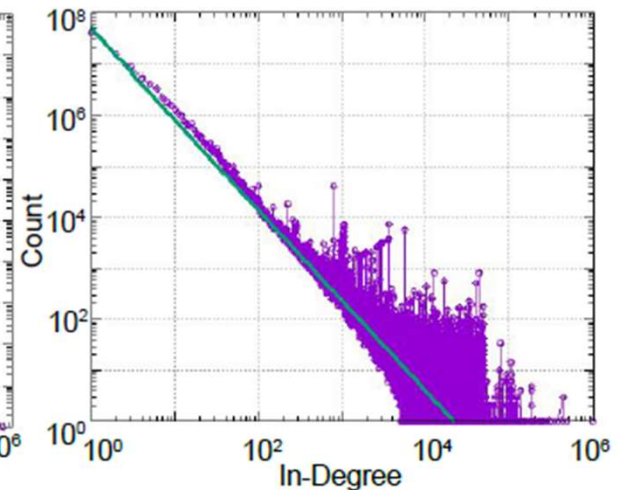
- Consisted of:
 - Low degree graphs (road-net-CA, road-net-USA)
 - Heavy tailed graphs (LiveJournal, Enwiki-2013, Twitter)
 - Power-law like graphs (UK-web)



(a) LiveJournal



(b) Twitter



(c) uk-web

Experimental Methodology : Metrics

- Ingress time
- Computation time
- Replication factor
- System wide resource usage
 - memory consumption
 - ~~CPU~~ utilization
 - network I/O usage

Experimental Methodology : Applications

- PageRank
- Weakly connected components
- K-core decomposition
- Single Source Shortest Path (SSSP)
- Simple coloring

PowerGraph : Partitioning Strategies

- Random
- Oblivious
- Constrained
 - Grid
 - Perfect Difference Sets (PDS)
- High Degree replicated First (HDRF)

PowerGraph : Partitioning Strategies

- **Random**
- Oblivious
- Constrained
 - Grid
 - Perfect Difference Sets (PDS)
- High Degree replicated First (HDRF)
- Edge's hash = function of its vertices
- Fast
- Parallelizable
- Even edge distribution
- Creates large number of mirrors

PowerGraph : Partitioning Strategies

- Random
- **Oblivious**
- Constrained
 - Grid
 - Perfect Difference Sets (PDS)
- High Degree replicated First (HDRF)
- Based on a greedy heuristic
- Goal : Keep replication factor low
- Incrementally and greedily place edges
- Requires some information about previous assignments
- Not a trivial strategy to parallelize and distribute

PowerGraph : Partitioning Strategies

- Random
- Oblivious
- **Constrained**
 - Grid
 - Perfect Difference Sets (PDS)
- High Degree replicated First (HDRF)
- Hash edges
- Restrict edge placement based on vertex adjacency:
 - Constraint set of vertex $v = S(v)$
 - Place edge (u,v) in partition belonging to set $S(u) \cap S(v)$

PowerGraph : Partitioning Strategies

- Random
- Oblivious
- Constrained
 - **Grid**
 - Perfect Difference Sets (PDS)
 - High Degree replicated First (HDRF)

$h(u) == 1$	2	3
4	5	6
7	8	$h(v) == 9$

- $S(v)$ = all machines in the row and column of the machine that v hashed to
- Upper bound for replication factor: $2\sqrt{N}-1$

PowerGraph : Partitioning Strategies

- Random
 - Oblivious
 - Constrained
 - Grid
 - **Perfect Difference Sets (PDS)**
 - High Degree replicated First (HDRF)
- Uses PDS to generate constraint sets
 - Requires $(p^2 + p + 1)$ machines, where p is prime.

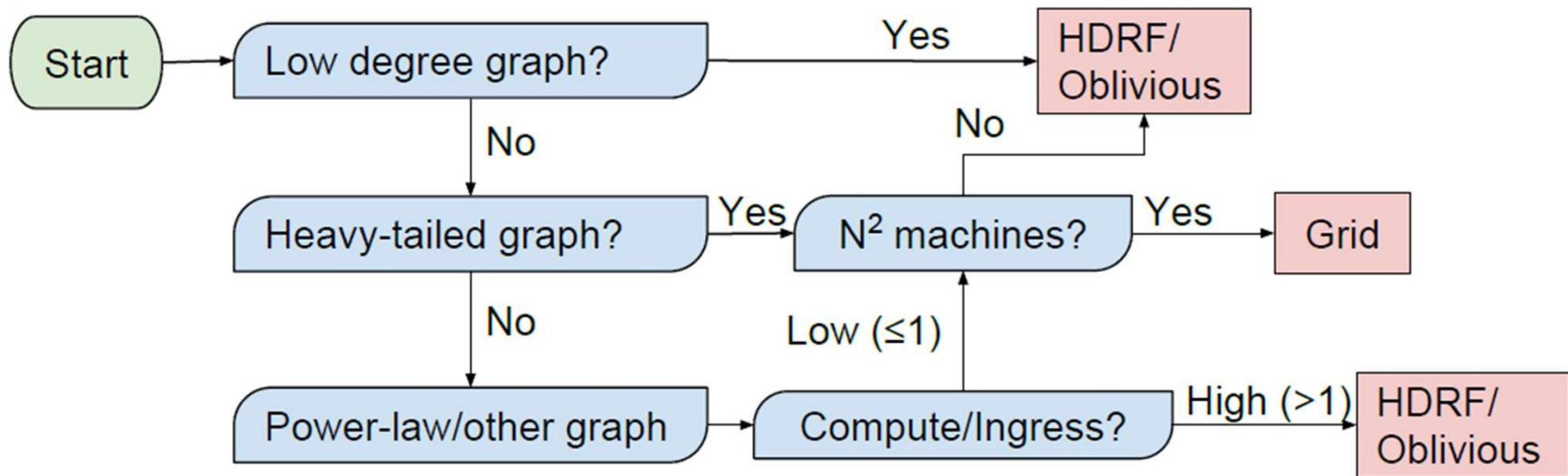
Couldn't meet requirement for number of machines of both Grid and PDS

PowerGraph : Partitioning Strategies

- Random
- Oblivious
- Constrained
 - Grid
 - ~~Perfect Difference Sets (PDS)~~
- **High Degree replicated First (HDRF)**
- Similar to Oblivious
- Assigns edges by looking at both partition size and vertex degrees
- Prefers replicating high degree vertices

PowerGraph : Results

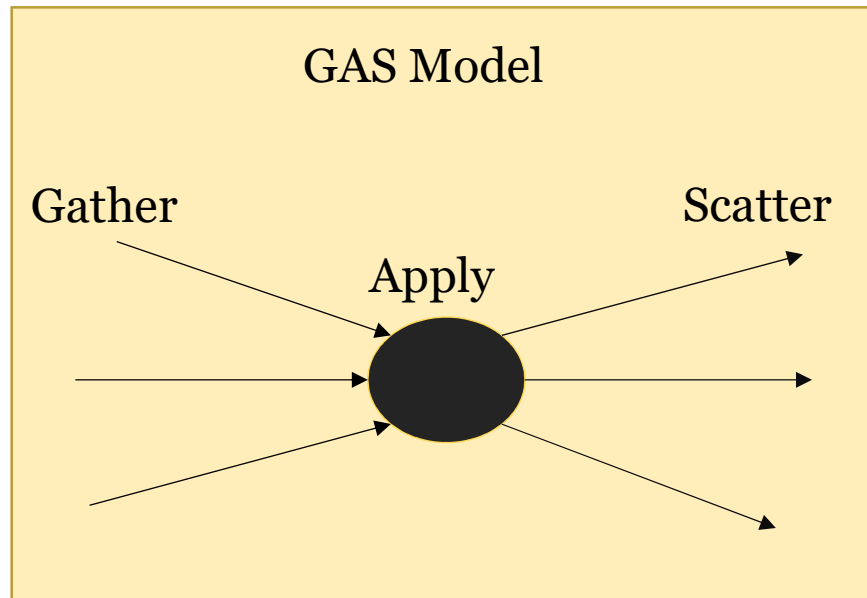
Strategy	PageRank (Conv.)			K-Core Decomp.		
	ingress	compute	total	ingress	compute	total
Grid	206.4	146.0	352.4	203.6	3794.9	3998.5
HDRF	322.0	103.6	425.6	320.6	3225.1	3545.7



PowerLyra : Partitioning Strategies

- Random
- Oblivious
- Constrained
 - Grid
 - ~~Perfect Difference Sets (PDS)~~
- Hybrid
- Hybrid-Ginger

PowerLyra : Partitioning Strategies



- Low degree destinations :
hash destination vertex
 - Colocate with all in-edges
- High degree destinations :
hash source vertex
 - Colocate with all out-edges

- **Hybrid**
- Hybrid-Ginger

PowerLyra : Partitioning Strategies

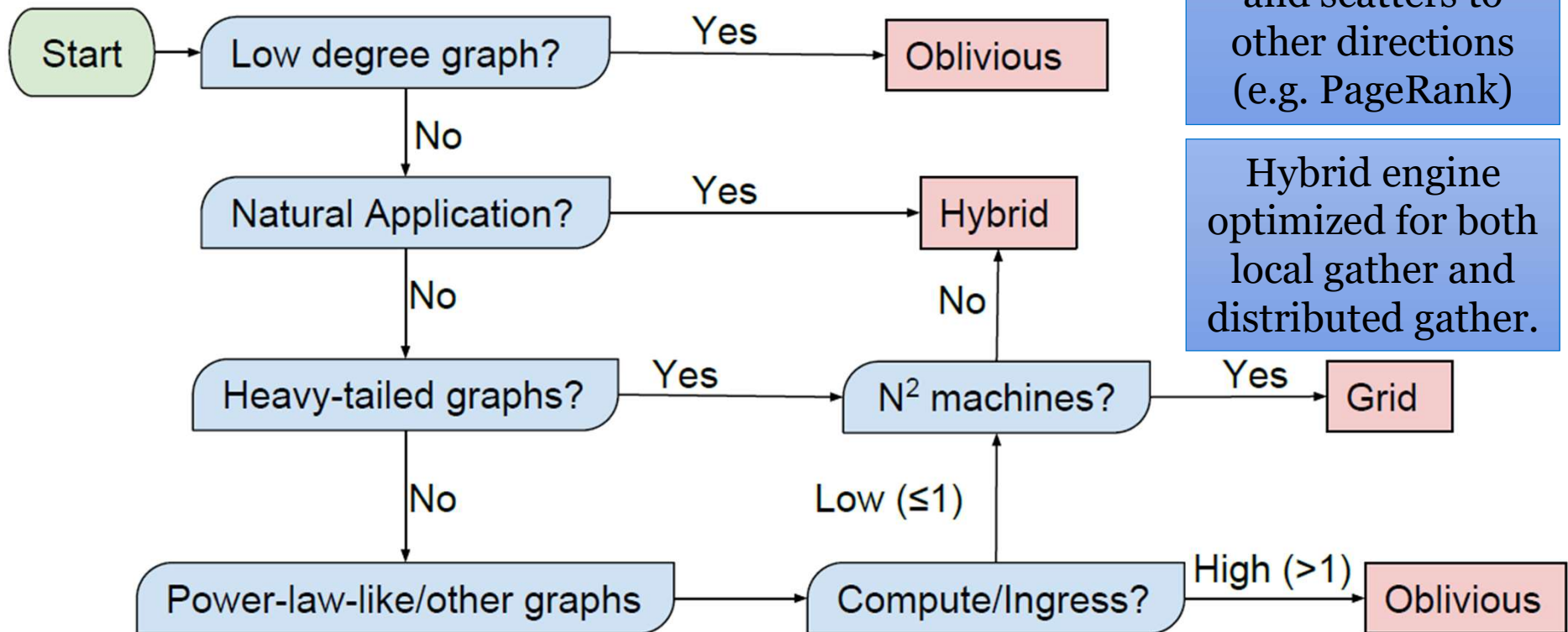
- Random
- Oblivious
- Constrained
 - Grid
 - ~~Perfect Difference Sets (PDS)~~
- **Hybrid**
- Hybrid-Ginger
- Two phases:
 1. Performs edge cuts on all vertices and updates degree counters
 2. Reassignment phase: performs vertex cuts on the vertices whose degree is above a certain threshold

PowerLyra : Partitioning Strategies

- Random
- Oblivious
- Constrained
 - Grid
 - ~~Perfect Difference Sets (PDS)~~
- Hybrid
- **Hybrid-Ginger**
 - A greedy streaming edge-cut strategy
 - Three phases:
 - 1 & 2. Same as Hybrid
 3. Put low degree vertex v in partition which has more of its in-neighbors
 - Gather locally

PowerLyra : Results

- Hybrid-Ginger consistently showed slower ingress and higher memory footprint.



Natural Applications:
Vertex gathers from one direction and scatters to other directions (e.g. PageRank)

Hybrid engine optimized for both local gather and distributed gather.

GraphX : Partitioning Strategies

- Random
- Canonical Random
- 1D Edge Partitioning
- 2D Edge Partitioning

GraphX : Partitioning Strategies

- **Random**
- Canonical Random
- 1D Edge Partitioning
- 2D Edge Partitioning
- Hashing the source and vertex IDs to assign edges
- Edges (u,v) and (v,u) go to the same partition

GraphX : Partitioning Strategies

- Random
- **Canonical Random**
- 1D Edge Partitioning
- 2D Edge Partitioning
- Hashing the source and vertex IDs in canonical direction to assign edges
- Edges (u,v) and (v,u) do not necessarily go to same partition
- Similar to Random of PowerGraph/PowerLyra

GraphX : Partitioning Strategies

- Random
- Canonical Random
- **1D Edge Partitioning**
- 2D Edge Partitioning
- Hash all edges by their source vertex
- Ensure all edges with the same source are in same partition
- Similar to PowerLyra's Hybrid's low degree vertex partitioning

GraphX : Partitioning Strategies

- Random
- Canonical Random
- 1D Edge Partitioning
- **2D Edge Partitioning**
 - Arrange all partitions into square matrix
 - Pick column on the basis of source vertex's hash and the row on the basis of destination vertex's hash
 - Ideal if number of partitions = perfect square
 - Similar to PowerGraph/PowerLyra's Grid

GraphX : Results

- Canonical random for low degree and high diameter graphs (e.g. road networks)
- 2D Edge Partitioning for power-law like graphs (e.g. Enwiki-2013)

PowerLyra: All Strategies

- HDRF
- 1D Edge partitioning
- 2D Edge partitioning
- Asymmetric Random (GraphX's Random)
- 1D-Target (new strategy)
 - Hash edges by destination vertex and colocate them

PowerLyra: All Strategies - Results

- No change in PowerLyra's decision tree
 - HDRF same as Oblivious
- Both Asymmetric Random and Random's performance were the worst
- PowerLyra's Hybrid engine enhances 2D edge partitioning

Summary

- No best fit for all situations
- Choice depends on:
 - Degree of distributed graph
 - Type and duration of application
 - Cluster size
- Choice significantly affects resource usage and application runtime
- Partitioning strategies tightly integrated with underlying engine perform better.

Discussion

- Why did they choose PowerGraph, PowerLyra and GraphX?
- Was leaving out the PDS partitioning strategy a good idea?
- PowerGraph can be used with both synchronous and asynchronous engines. In the paper, they mentioned that they used asynchronous engine for simple coloring, which led to longer computation time and higher network I/O usage. Why did they not use synchronous engine for it?
- Why did Hybrid-Ginger give such bad performance?

Discussion

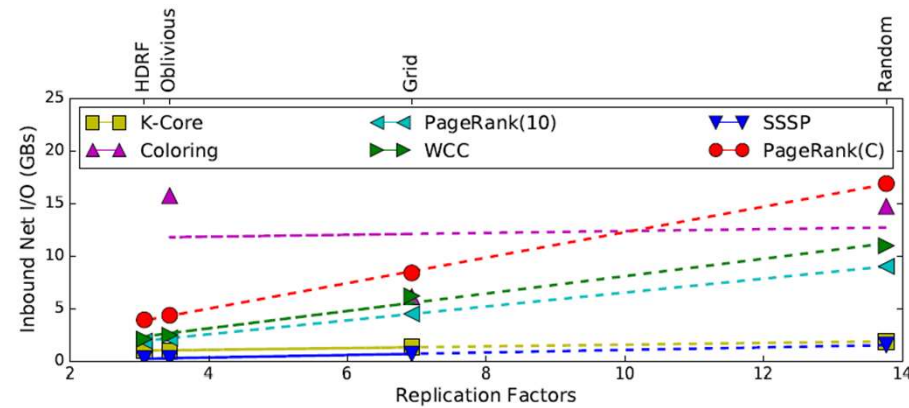


Figure 3: Incoming Network IO vs. Replication Factors. (PowerGraph, EC2-25, UK-Web).

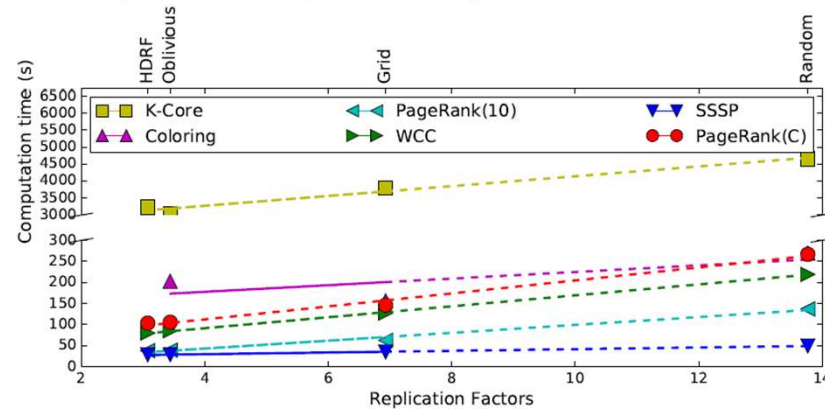


Figure 4: Computation Time in seconds vs. Replication Factors. (PowerGraph, EC2-25, UK-Web).