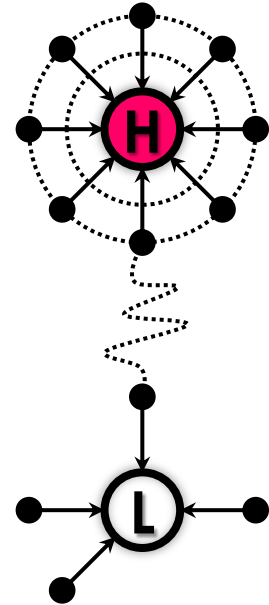


PowerLyra

Differentiated Graph Computation and Partitioning on Skewed Graphs



RONG CHEN, JIAXIN SHI, YANZHE CHEN, HAIBO CHEN

Institute of Parallel and Distributed Systems
Shanghai Jiao Tong University, China



EuroSys 2015

Graphs are Everywhere¹

Traffic network graphs

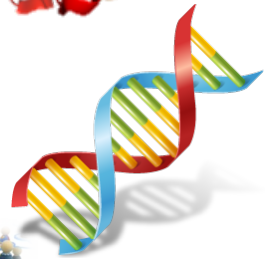
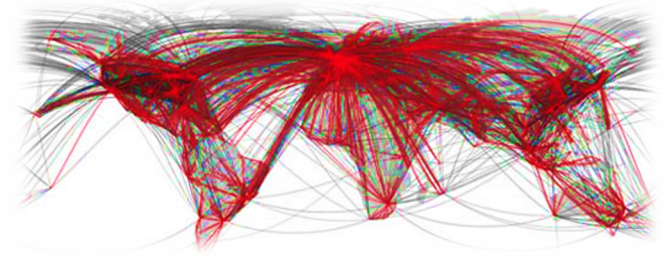
- Monitor/react to road incidents
- Analyze Internet security

Biological networks

- Predict disease outbreaks
- Model drug-protein interactions

Social networks

- Recommend people/products
- Track information flow



¹ From PPOP'15 Keynote: *Computing in a Persistent (Memory) World*, P. Faraboschi

Graph-parallel Processing

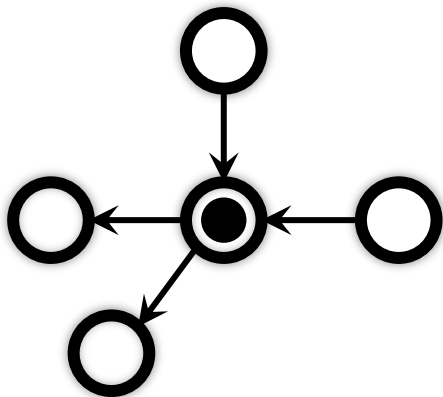
Coding graph algorithms as **vertex-centric** programs to process **vertices** in parallel and communicate along **edges**

-- "Think as a Vertex" philosophy

Example: *PageRank*

A centrality analysis algorithm to **iteratively** rank each vertex as a weighted sum of neighbors' ranks

$$R_i = 0.15 + 0.85 \sum_{(j,i) \in E} \omega_{ij} R_j$$



```
COMPUTE(v)
  foreach n in v.in_nbrs
    sum += n.rank / n.nedges
  v.rank = 0.15 + 0.85 * sum
  if !converged(v)
    foreach n in v.out_nbrs
      activate(n)
```

Graph-parallel Processing

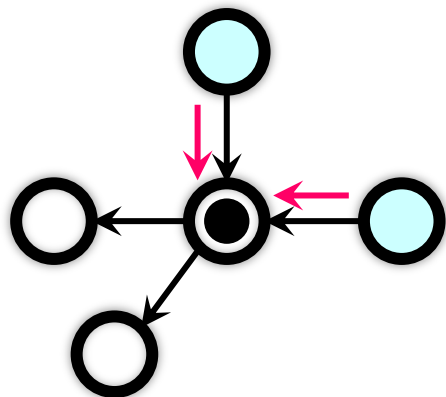
Coding graph algorithms as **vertex-centric** programs to process **vertices** in parallel and communicate along **edges**

-- "Think as a Vertex" philosophy

Example: *PageRank*

A centrality analysis algorithm to **iteratively** rank each vertex as a weighted sum of neighbors' ranks

$$R_i = 0.15 + 0.85 \sum_{(j,i) \in E} \omega_{ij} R_j$$



Gather data
from neighbors
via in-edges

COMPUTE(v)

```
foreach n in v.in_nbrs  
    sum += n.rank / n.nedges
```

```
v.rank = 0.15 + 0.85 * sum
```

```
if !converged(v)  
    foreach n in v.out_nbrs  
        activate(n)
```

Graph-parallel Processing

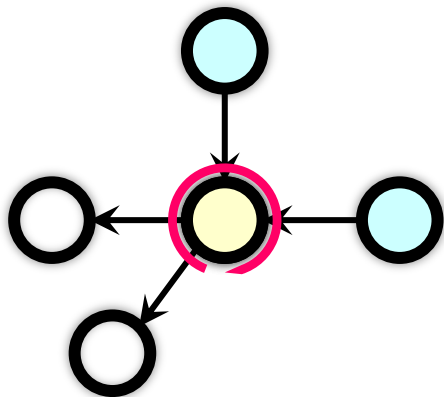
Coding graph algorithms as **vertex-centric** programs to process **vertices** in parallel and communicate along **edges**

-- "Think as a Vertex" philosophy

Example: *PageRank*

A centrality analysis algorithm to **iteratively** rank each vertex as a weighted sum of neighbors' ranks

$$R_i = 0.15 + 0.85 \sum_{(j,i) \in E} \omega_{ij} R_j$$



Update vertex
with new data

COMPUTE(v)

```
foreach n in v.in_nbrs  
    sum += n.rank / n.nedges
```

```
v.rank = 0.15 + 0.85 * sum
```

```
if !converged(v)  
    foreach n in v.out_nbrs  
        activate(n)
```

Graph-parallel Processing

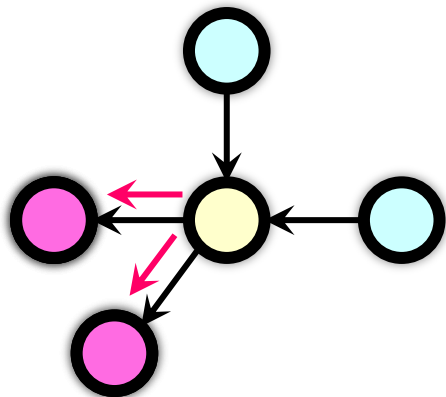
Coding graph algorithms as **vertex-centric** programs to process **vertices** in parallel and communicate along **edges**

-- "Think as a Vertex" philosophy

Example: *PageRank*

A centrality analysis algorithm to **iteratively** rank each vertex as a weighted sum of neighbors' ranks

$$R_i = 0.15 + 0.85 \sum_{(j,i) \in E} \omega_{ij} R_j$$



Scatter data
to neighbors
via out-edges

COMPUTE(v)

```
foreach n in v.in_nbrs  
    sum += n.rank / n.nedges
```

```
v.rank = 0.15 + 0.85 * sum
```

```
if !converged(v)  
    foreach n in v.out_nbrs  
        activate(n)
```

Natural Graphs are Skewed

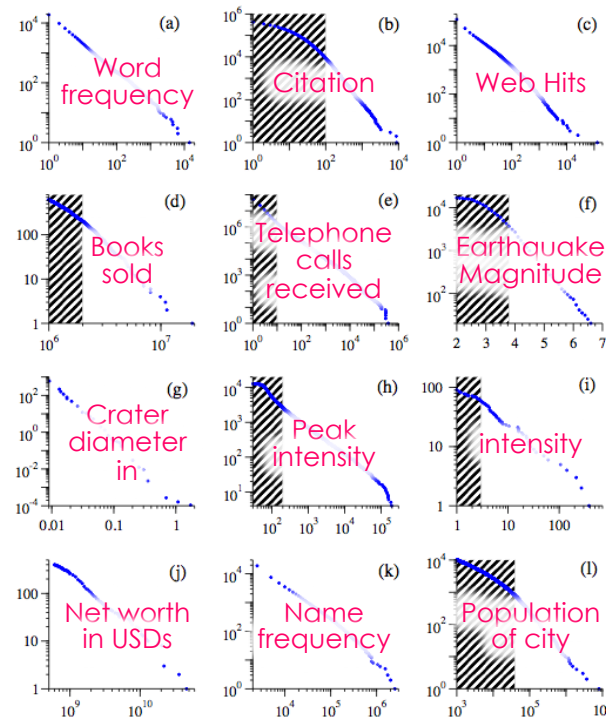
Graphs in real-world

- *Power-law* degree distribution (aka Zipf's law or Pareto)

“*most* vertices have relatively *few* neighbors while a *few* have *many* neighbors” -- PowerGraph^[OSDI'12]

Case: SINA Weibo(微博)

- 167 million active users¹



Source: M. E. J. Newman, "Power laws, Pareto distributions and Zipf's law", 2005



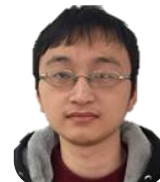
#1



#2



#100



#Follower : 77,971,093 76,946,782 ... 14,723,818 397 69

¹ <http://www.chinainternetwatch.com/10735/weibo-q3-2014/>

Challenge: LOCALITY vs. PARALLELISM



LOCALITY

1. Imbalance
2. High contention
3. Heavy network traffic

make resource
locally accessible
to hidden
network latency

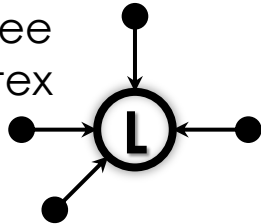


Challenge: LOCALITY vs. PARALLELISM

Low-degree vertex



397



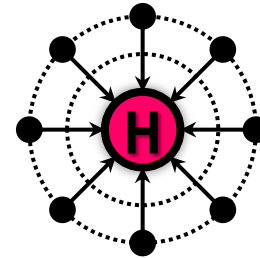
LOCALITY

make resource
locally accessible
to hidden
network latency



+

1. Redundant
comm., comp. &
sync. (not worthwhile)



High-degree vertex



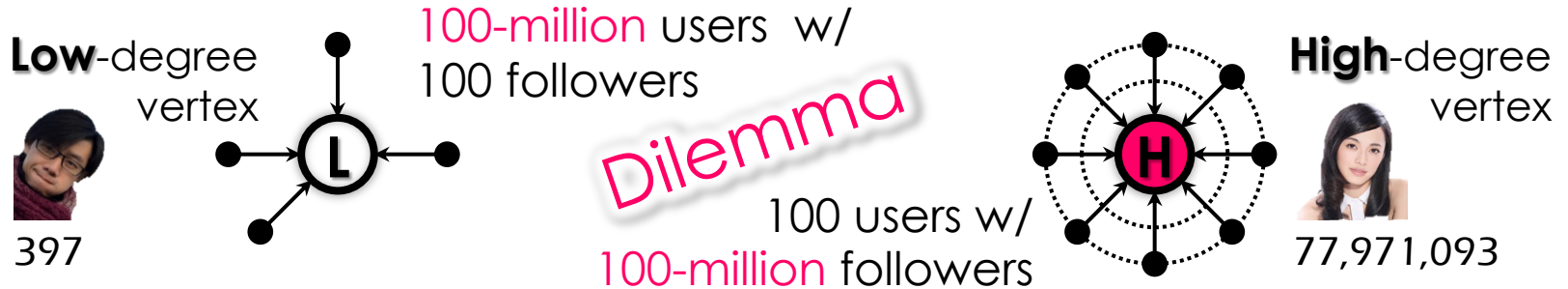
77,971,093

PARALLELISM

evenly **parallelize**
the workloads
to avoid
load imbalance



Challenge: LOCALITY vs. PARALLELISM



LOCALITY

make resource
locally accessible
to hidden
network latency



← **Conflict** →


skewed graphs

PARALLELISM

evenly **parallelize**
the workloads
to avoid
load imbalance



Existing Efforts: One Size Fits All

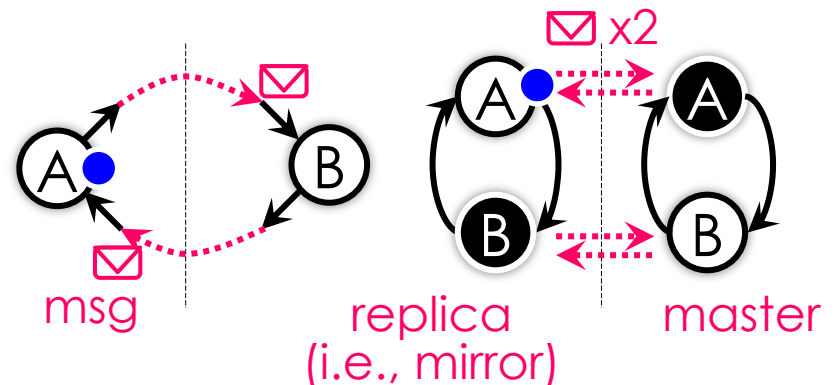
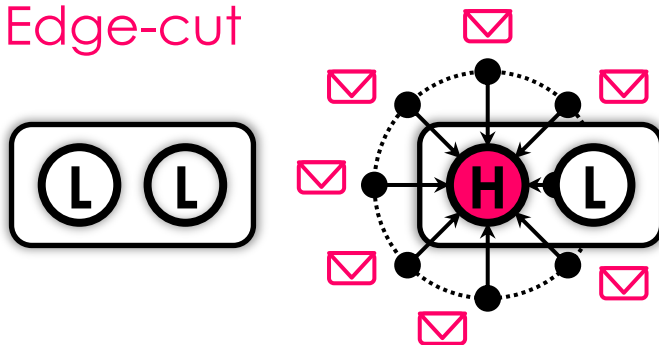


Pregel^[SIGMOD'08] and **GraphLab**^[VLDB'12]



- Focus on exploiting **LOCALITY**
- Partitioning: use **edge-cut** to evenly assign vertices along with all edges
- Computation: **aggregate** all resources (i.e., msgs or replicas) of a vertex on local machine

Edge-cut



Existing Efforts: One Size Fits All

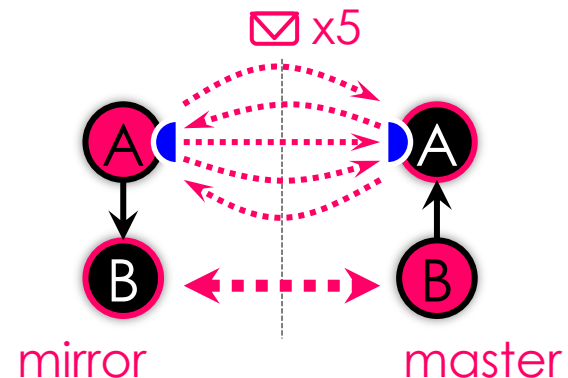
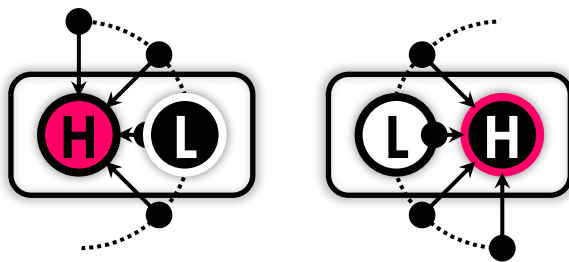


PowerGraph [OSDI'12] and GraphX [OSDI'14]



- Focus on exploiting **PARALLELISM**
- Partitioning: use **vertex-cut** to evenly assign edges with replicated vertices
- Computation: **decompose** the workload of a vertex into multiple machines

Vertex-cut



Contributions

PowerLyra: differentiated graph computation and partitioning on skewed graphs

- **Hybrid** computation and partitioning strategies
 - Embrace **locality** (low-degree vertex) and **parallelism** (high-degree vertex)
- An optimization to improve locality and reduce interference in **communication**
- Outperform PowerGraph (up to **5.53X**) and other systems for ingress and runtime



LOCAL LOCAL PARALLELISM

One Size
fits All



Agenda

Graph Partitioning

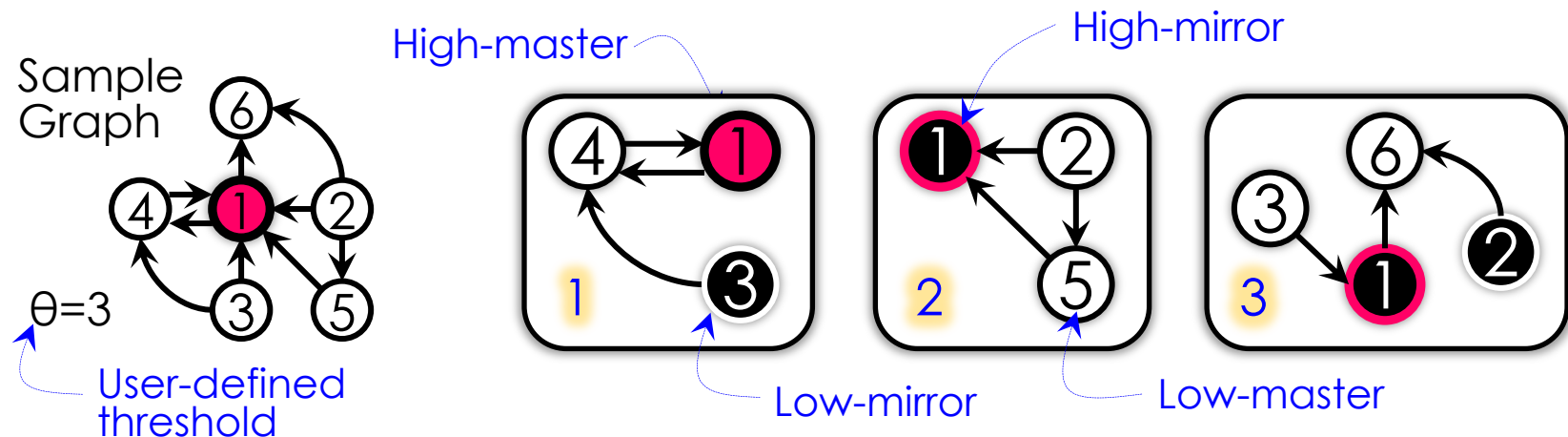
Graph Computation

Evaluation

Graph Partitioning

Hybrid-cut: differentiate graph partitioning for **low-degree** and **high-degree** vertices

- **Low-cut** (inspired by **edge-cut**): reduce mirrors and exploit locality for low-degree vertices
- **High-cut** (inspired by **vertex-cut**): provide balance and restrict the impact of high-degree vertices
- **Efficiently** combine low-cut and high-cut

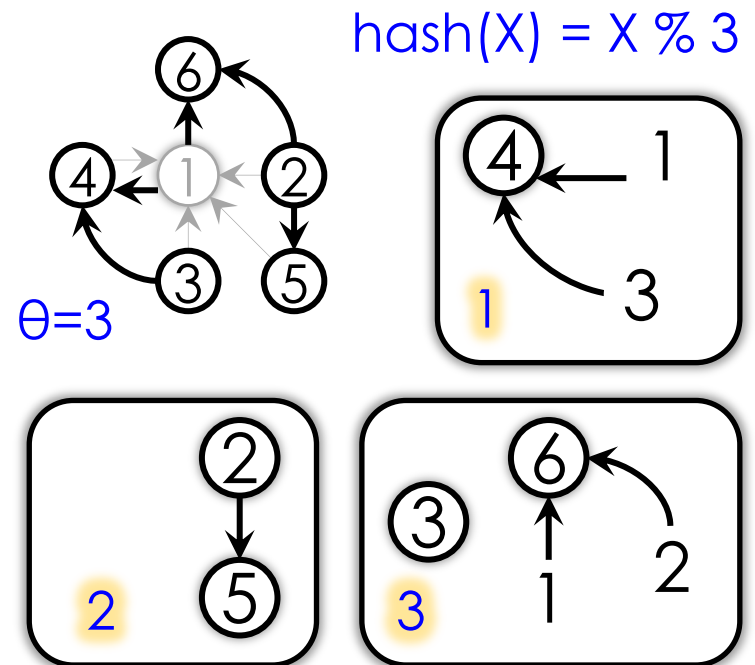


Hybrid-cut (Low)

Low-cut: evenly assign **low-degree** vertices along with edges in only **one direction** (e.g., **in-edges**) to machines by hashing the (e.g., **target**) vertex

1. **Unidirectional** access locality
2. No duplicated edges
3. Efficiency at ingress
4. Load balance (E & V)
5. Ready-made heuristics

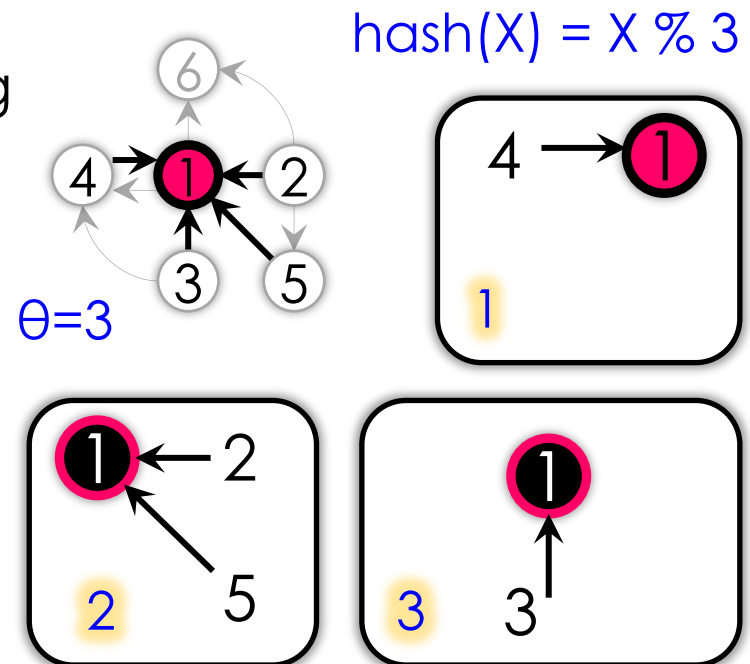
NOTE: A new heuristic, called **Ginger**, is provided to further optimize Random hybrid-cut. ([see paper](#))



Hybrid-cut (High)

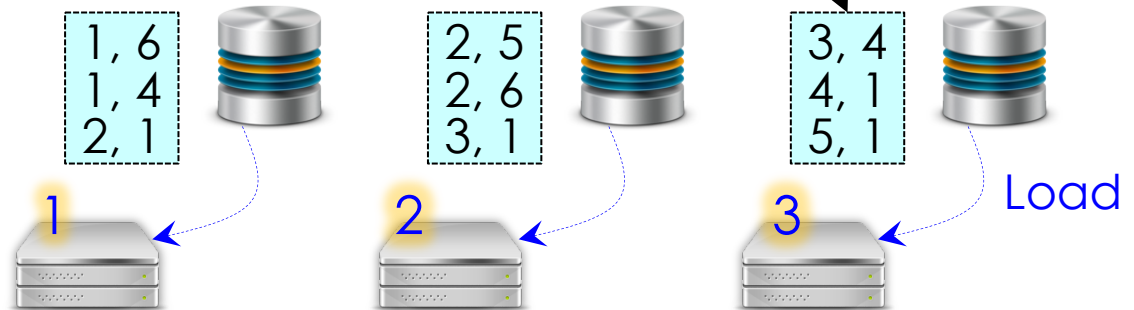
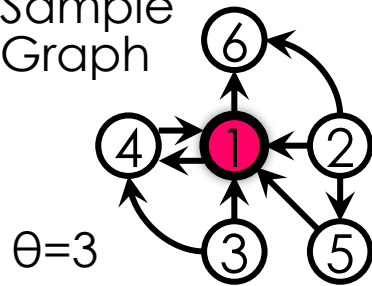
High-cut: distribute one-direction edges (e.g., *in-edges*) of *high-degree* vertices to machines by hashing opposite (e.g., *source*) vertex

1. Avoid *adding any mirrors* for low-degree vertices (i.e., an upper bound of increased mirrors for assigning a new high-degree vertex)
2. No duplicated edges
3. Efficiency at ingress
4. Load balance (E & V)



Constructing Hybrid-cut

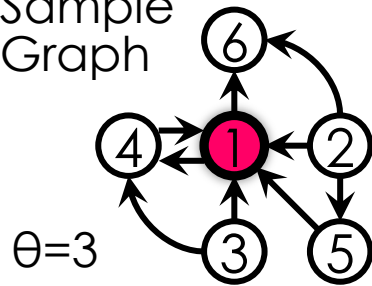
Sample
Graph



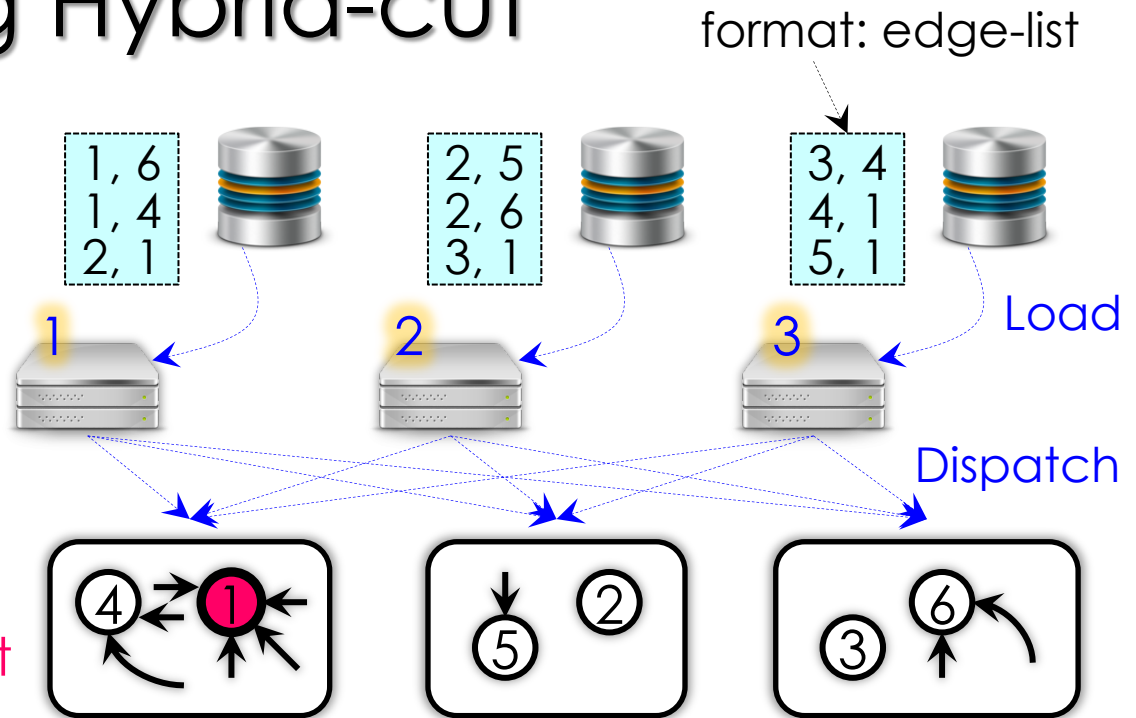
1. Load files in parallel

Constructing Hybrid-cut

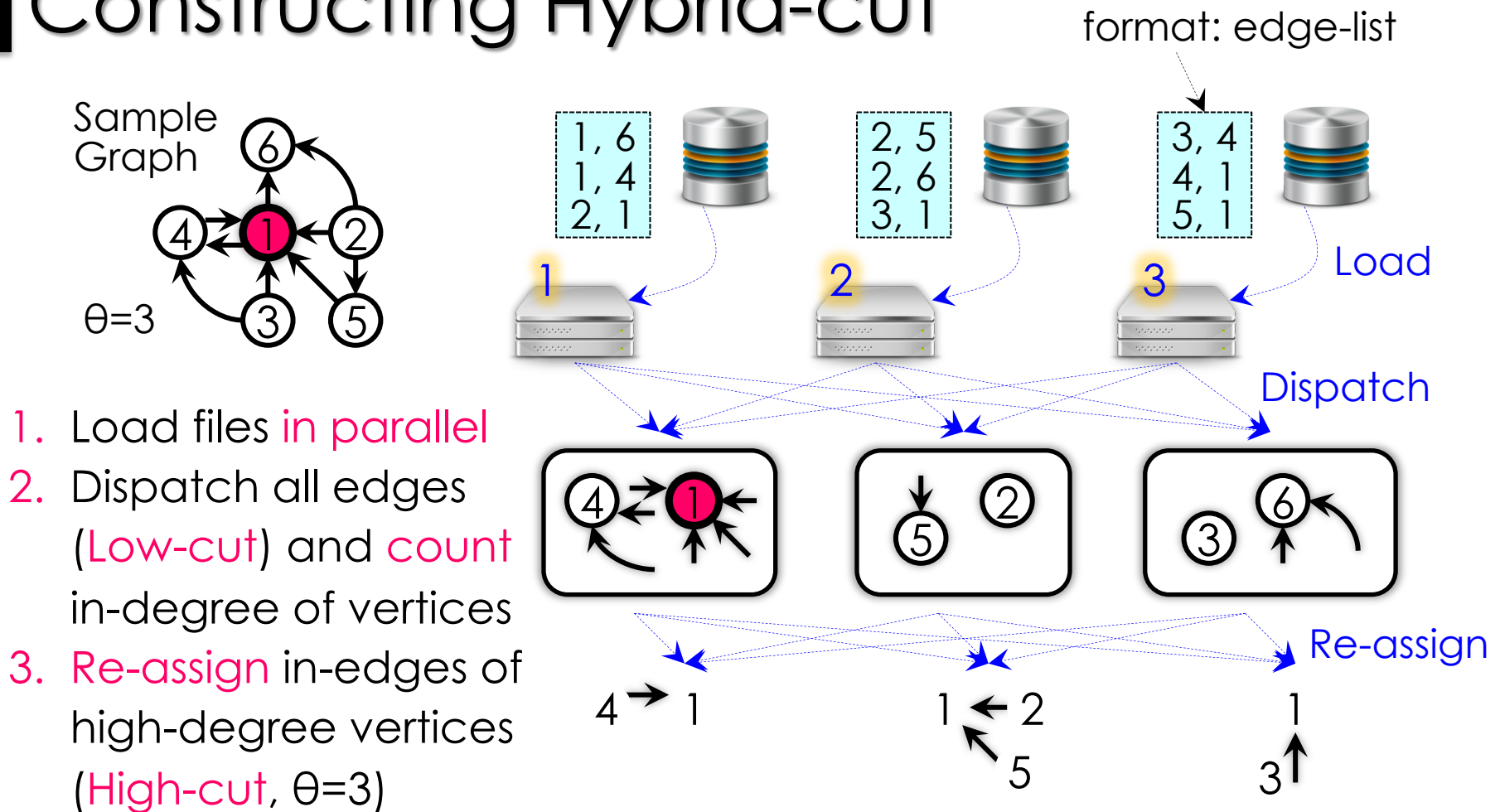
Sample Graph



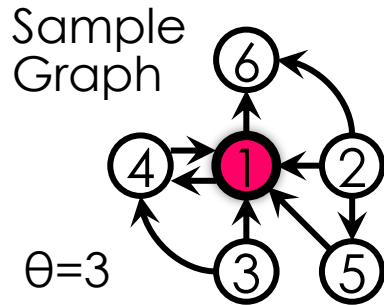
1. Load files **in parallel**
2. Dispatch all edges (**Low-cut**) and **count** in-degree of vertices



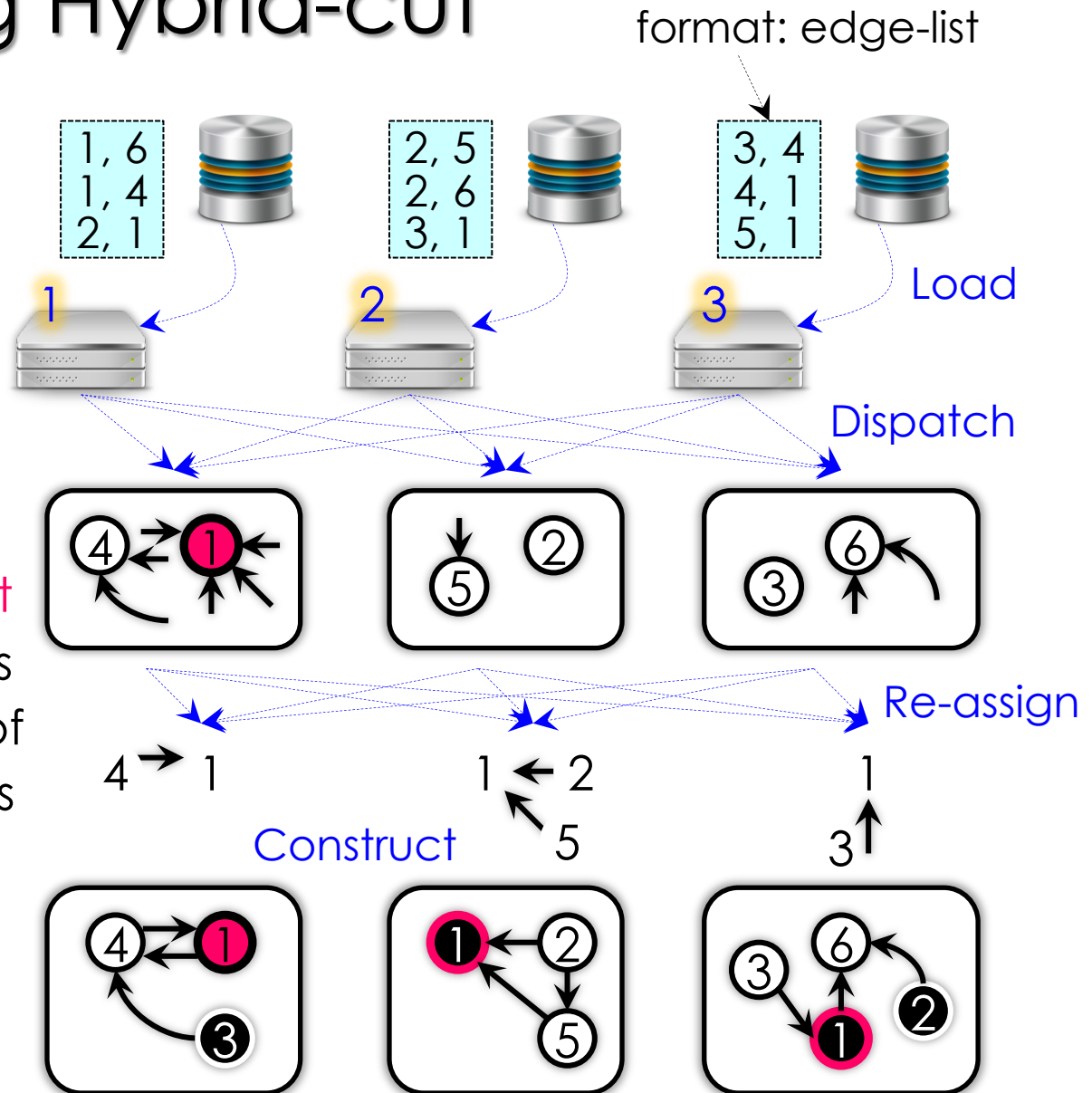
Constructing Hybrid-cut



Constructing Hybrid-cut



1. Load files **in parallel**
2. Dispatch all edges (**Low-cut**) and **count** in-degree of vertices
3. **Re-assign** in-edges of high-degree vertices (High-cut, $\theta=3$)
4. Create mirrors to **construct** a local graph



Agenda

Graph Partitioning

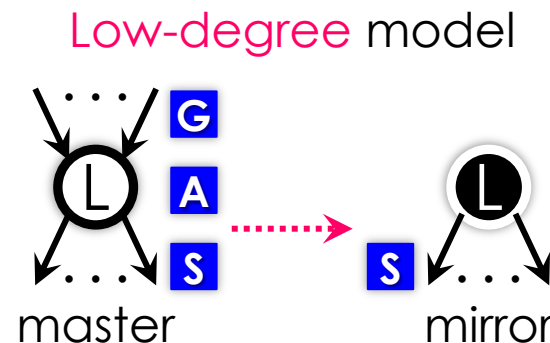
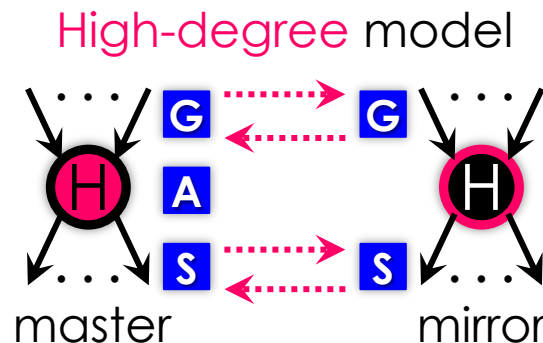
Graph Computation

Evaluation

Graph Computation

Hybrid-model: differentiate the graph computation model to high-degree and low-degree vertices

- **High-degree** model: decompose workload of high-degree vertices for load balance
- **Low-degree** model: minimize the communication cost for low-degree vertices
- **Seamlessly** run all of existing graph algorithms



Hybrid-model (High)

High-degree vertex

- Exploit **PARALLELISM**
- Follow **GAS** model
(inspired by *POWERGRAPH*)
- Comm. Cost: $\leq 4 \times \# \text{mirrors}$

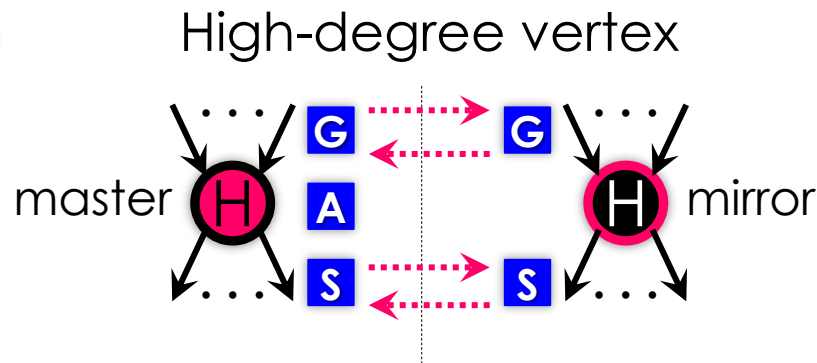
Example: PageRank

COMPUTE(v)

```
foreach n in v.in_nbrs  
    sum += n.rank / n.nedges
```

```
v.rank = 0.15 + 0.85 * sum
```

```
if !converged(v)  
    foreach n in v.out_nbrs  
        activate(n)
```



G **distributed**

```
GATHER_EDGES = IN  
GATHER(v, n):  
    return n.rank / n.nedges  
SUM(a, b): return a + b
```

A **local**

```
APPLY(v, sum):  
    v.rank = 0.15 + 0.85 * sum
```

S **distributed**

```
SCATTER_EDGES = OUT  
SCATTER(v, n):  
    if !converged(v)  
        activate(n)
```

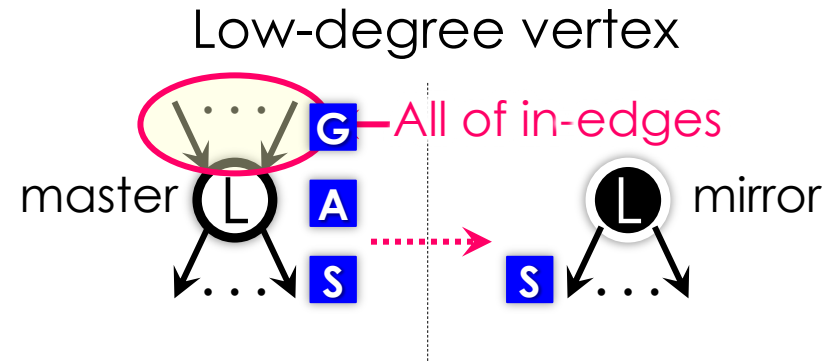
Hybrid-model (Low)

Low-degree vertex

- Exploit LOCALITY
- Reserve GAS interface

OBSERVATION: most algorithms only gather or scatter in one direction (e.g., PageRank: G/*IN* and S/*OUT*)

- Unidirectional access locality
- Comm. Cost: $\leq 1 \times \# \text{mirrors}$



G **GATHER_EDGES = IN** **local**
GATHER(v, n):
 return n.rank / n.nedges
SUM(a, b): return a + b

A **APPLY (v, sum):** **local**
 v.rank = 0.15 + 0.85 * sum

S **SCATTER_EDGES = OUT** **distributed**
SCATTER(v, n):
 if !converged(v)
 activate(n)

Generalization

One-direction locality limits the expressiveness to some OTHER algorithms (i.e., G or S in both IN and OUT)

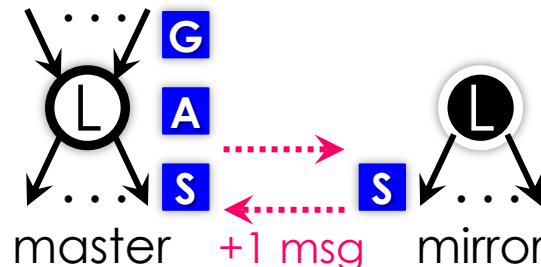
TYPE	GATHER_EDGES	SCATTER_EDGES	EX. ALGO.
NATURAL	IN or NONE	OUT or NONE	PR, SSSP
	OUT or NONE	IN or NONE	DIA
OTHER	ANY	ANY	CC, ALS

Seamlessly run all graph algorithms

- Adaptively downgrade model (+ N msgs, $1 \leq N \leq 3$)
- Detect at runtime w/o any changes to APPs

e.g., Connected Components (CC)

- G/NONE and S/ALL
- +1 msg in SCATTER (mirror → master)
- Comm. Cost: $\leq 2 \times \text{\#mirrors}$



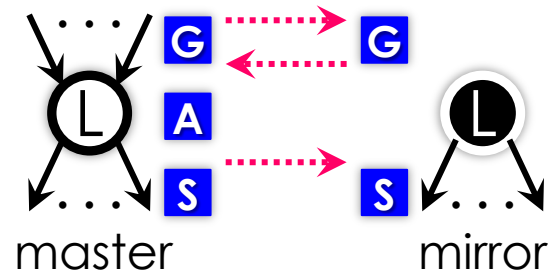
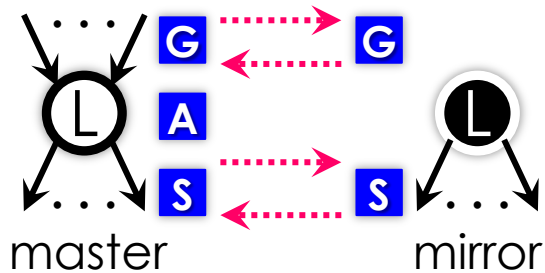
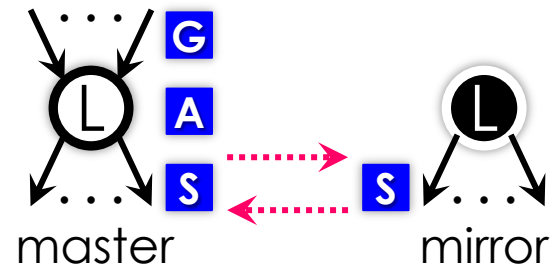
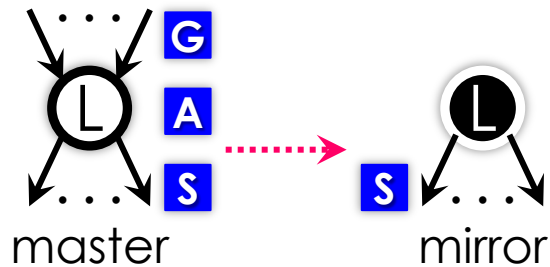
Downgrade

OBSERVATION:

most algorithms only gather or scatter **in one direction**

	#msgs	GATHER			
		ALL	IN	OUT	NONE
ALL	4	4	2	2	2
IN	3		2	1	1
OUT	3		1	2	1
NONE	3		1	1	1

Sweet Spot



Agenda

Graph Partitioning

Graph Computation

Evaluation

Implementation

POWERLYRA

- Based on latest PowerGraph (v2.2, Mar. 2014)
- A separate engine and partitioning algorithms
- Support both **sync** and **async** execution engines
- **Seamlessly** run all graph toolkits in GraphLab and respect the fault tolerance model
- Port the Random hybrid-cut to GraphX¹

The source code and a brief instruction are available at <http://ipads.se.sjtu.edu.cn/projects/powerlyra.html>

¹ We only port Random hybrid-cut for preserving the graph partitioning interface of GraphX.

Evaluation

Baseline: latest *POWERGRAPH* (2.2 Mar. 2014) with various vertex-cuts (*G*rid/*O*blivious/*C*oordinated)

Platforms 48-node 6-node

1. A 48-node VM-based cluster¹ (Each: 4 Cores, 12GB RAM)
2. A 6-node physical cluster¹ (Each: 24 Cores, 64GB RAM)

Algorithms and graphs

- 3 graph-analytics algorithms (PR², CC, DIA) and 2 MLDM algorithms (ALS, SGD)
- 5 real-world graphs, 5 synthetic graphs³, etc.

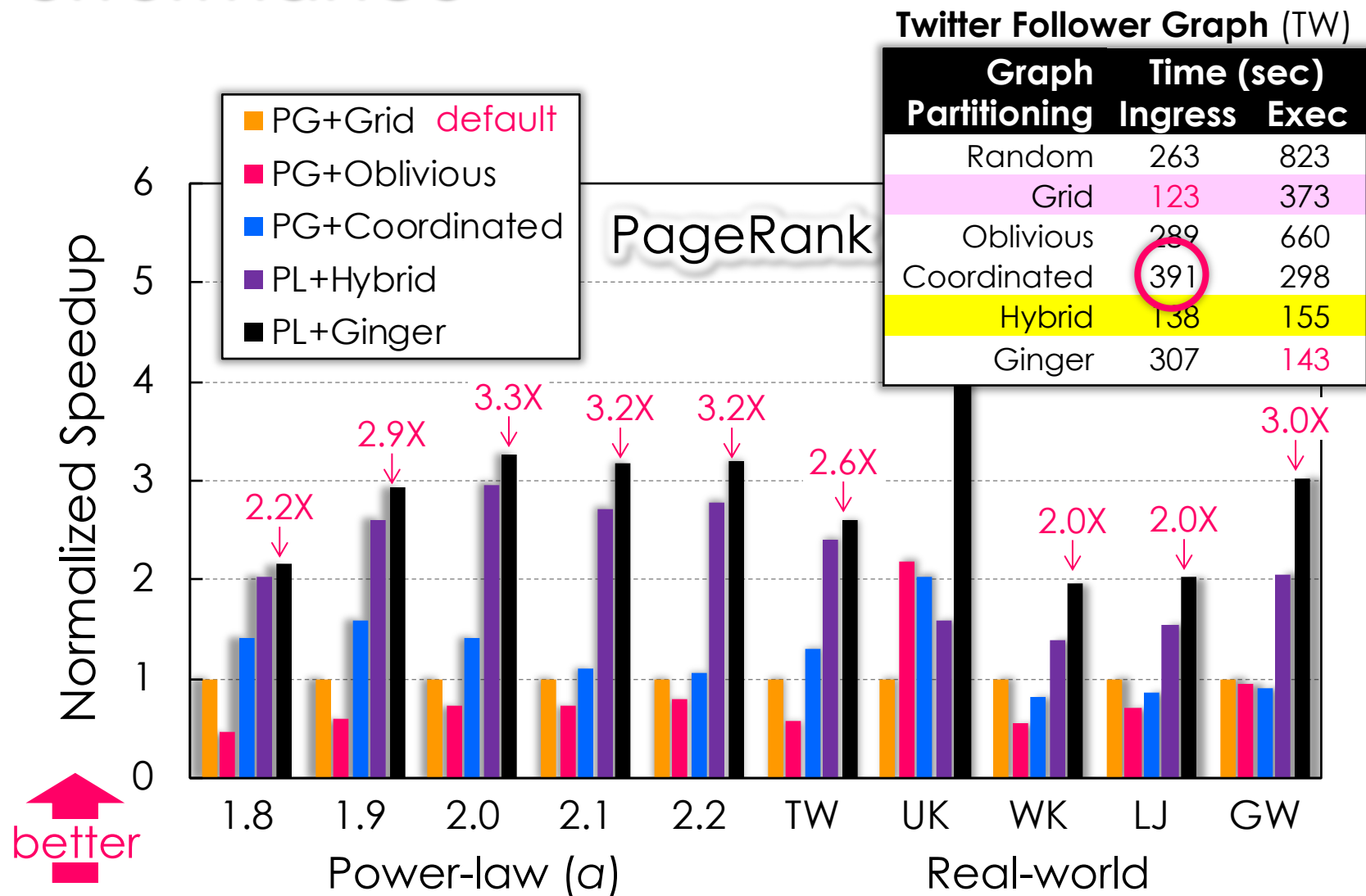
Graph	V	E	α	E
Twitter	42M	1.47B	1.8	673M
UK-2005	40M	936M	1.9	249M
Wiki	5.7M	130M	2.0	105M
LJournal	5.4M	79M	2.1	54M
GWeb	0.9M	5.1M	2.2	39M

¹ Clusters are connected via 1GigE

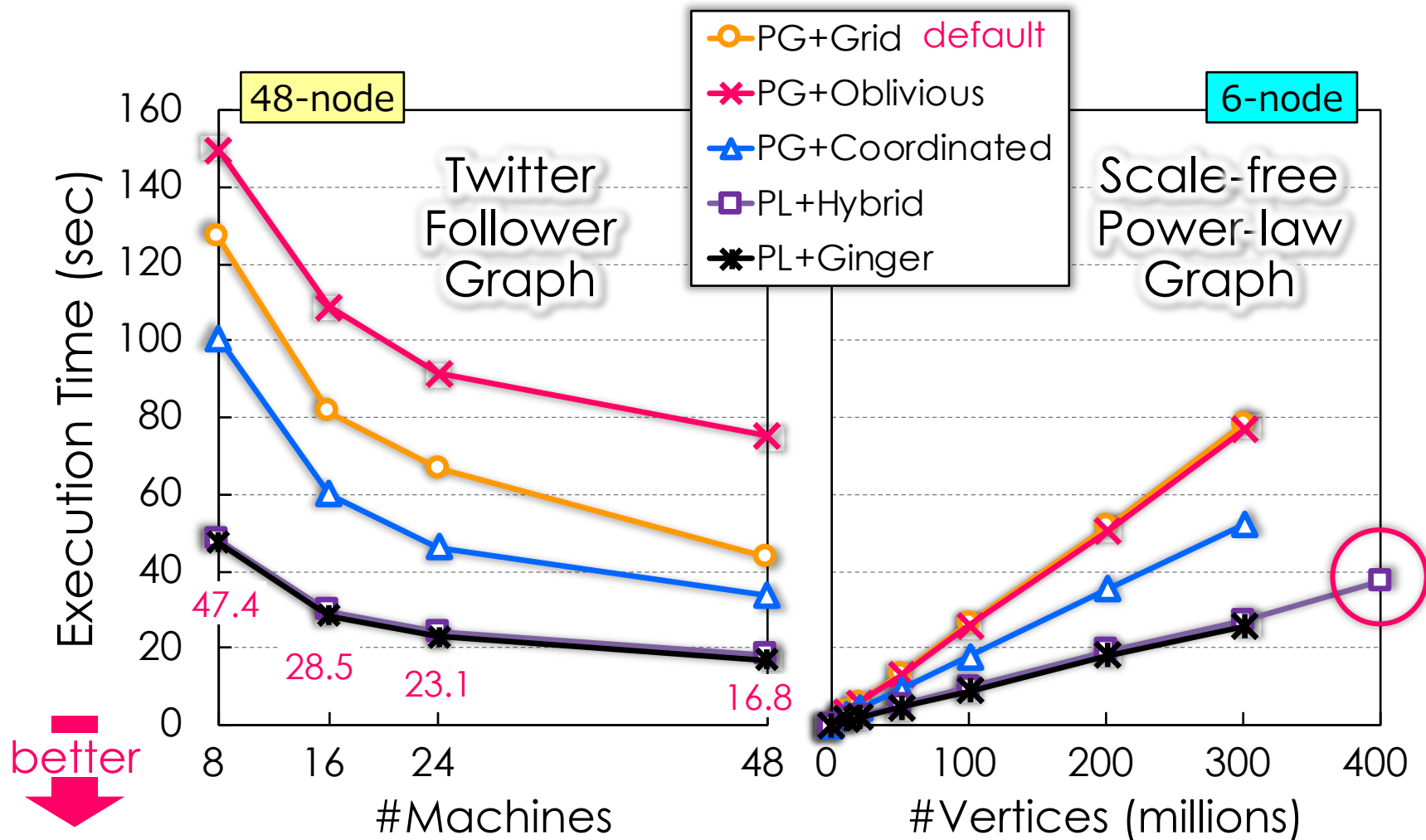
² We run 10 iterations for PageRank.

³ Synthetic graphs has fixed 10 million vertices

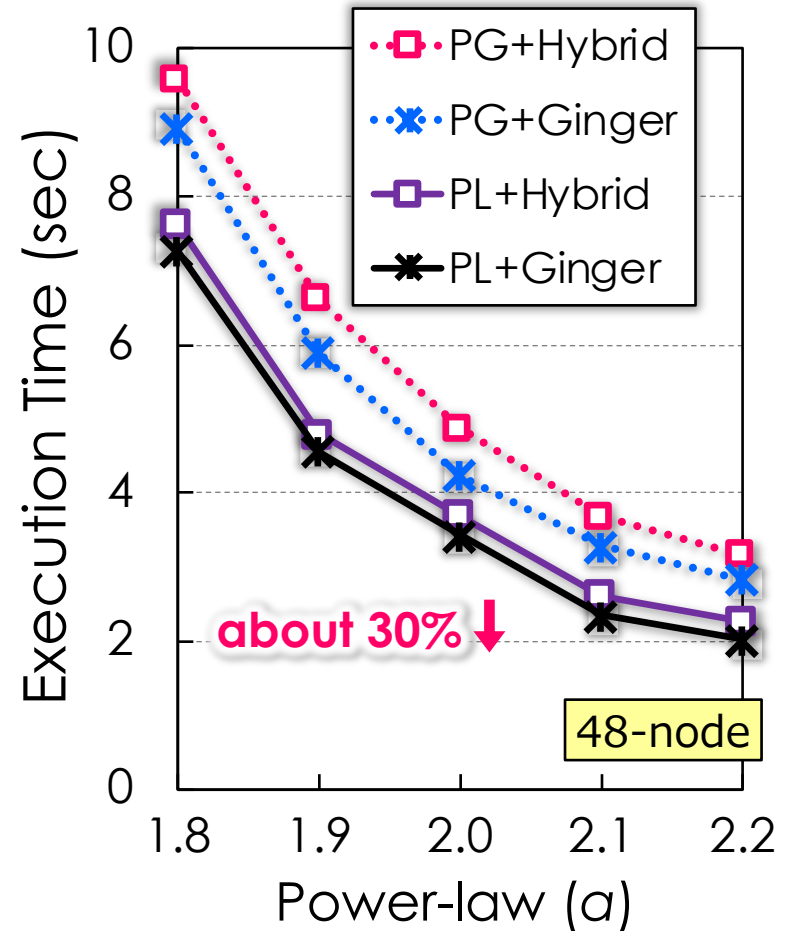
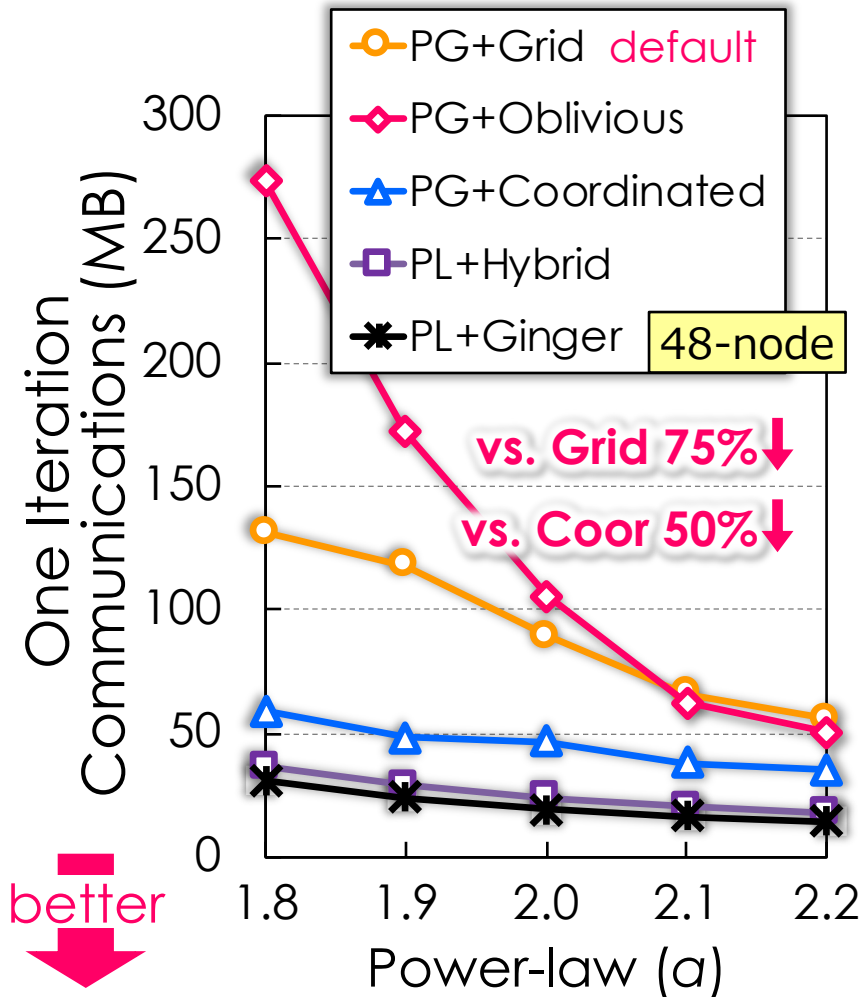
Performance



Scalability



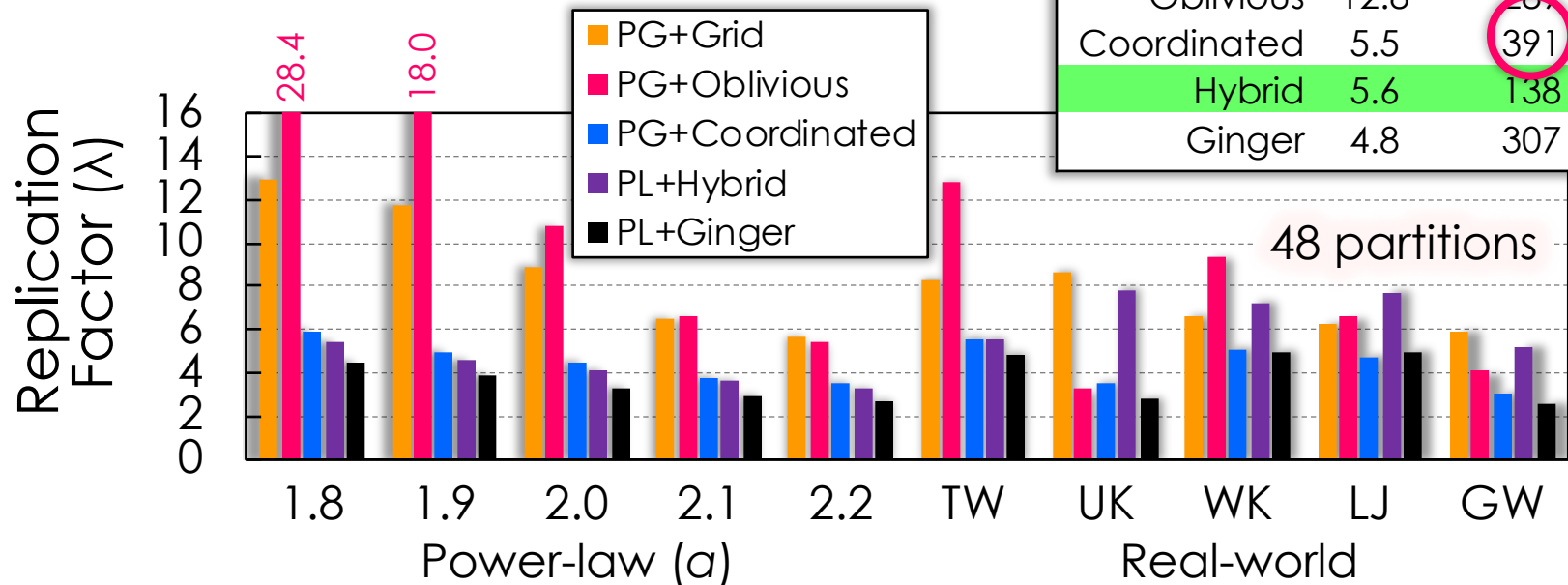
Breakdown



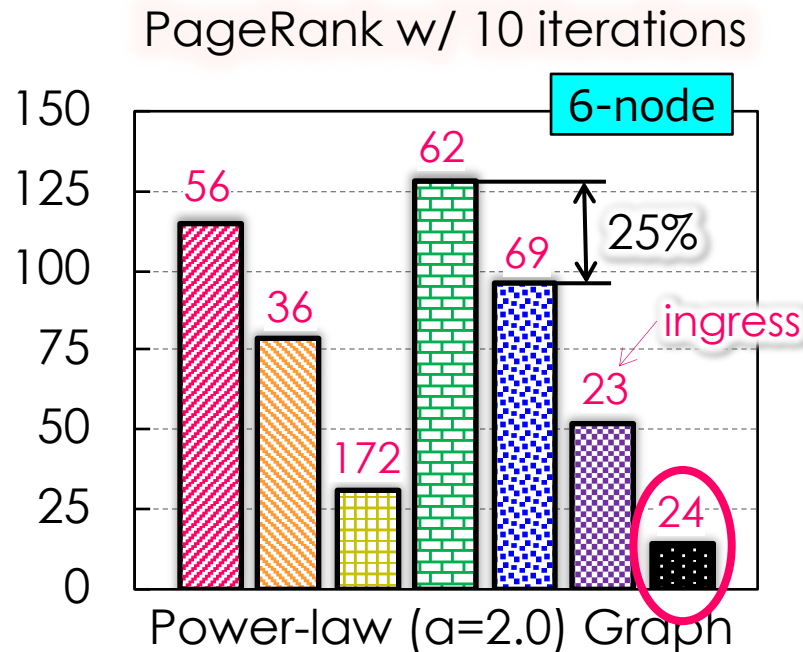
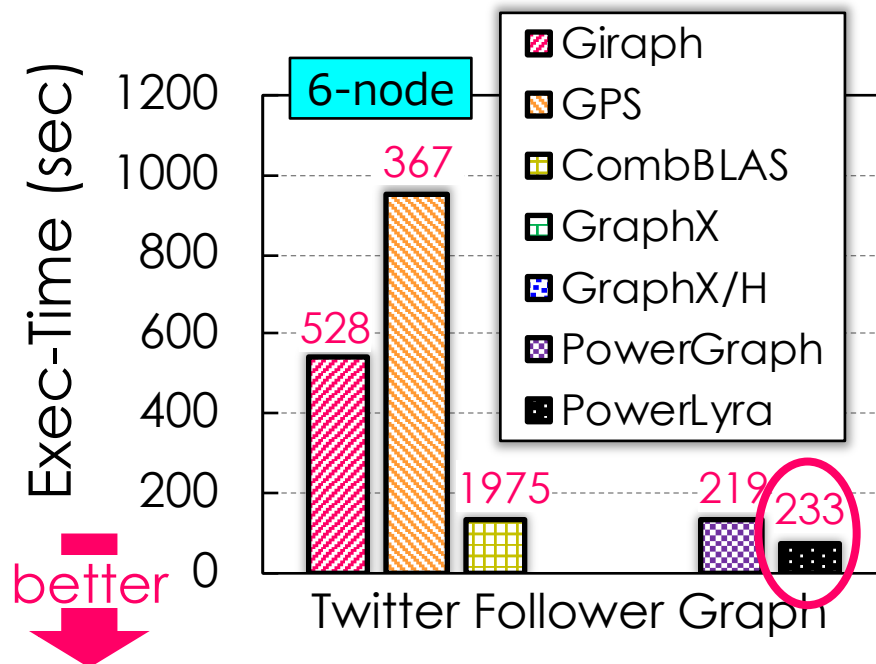
Graph Partitioning Comparison

Partitioning Algorithm	Type	Replication Factor (λ)	Ingress Time	Load Balance
Random	Hashing	Bad	Modest	Good
Grid	2D Hashing	Modest	Great	Modest
Oblivious	Greedy	Modest	Modest	Great
Coordinated	Greedy	Great	Bad	Modest
Hybrid(R)	Hashing	Good	Great	Modest
Ginger	Greedy	Great	Bad	Modest

Twitter Follower Graph (TW)		
Partitioning	λ	Ingress
Random	16.0	263
Grid	8.3	123
Oblivious	12.8	289
Coordinated	5.5	391
Hybrid	5.6	138
Ginger	4.8	307



vs. Other Systems¹



PageRank with 10 iterations

	Power-law Graphs	PowerLyra (6/1)	Polymer	Galois	X-Stream	GraphChi
In-memory →	V =10M	14 / 45	6.3	9.8	9.0	115
Out-of-core →	V =400M	186 / --	--	--	710	1666

¹ We use Giraph 1.1, CombBLAS 1.4, GraphX 1.1, Galois 2.2.1, X-Stream 1.0, as well as the latest version of GPS, PowerGraph, Polymer and GraphChi.

Conclusion

Existing systems use a “*ONE SIZE FITS ALL*” design for skewed graphs, resulting in suboptimal performance

POWERLYRA: a hybrid and adaptive design that differentiates *computation* and *partitioning* on low- and high-degree vertices

Embrace *LOCALITY* and *PARALLELISM*

- Outperform *PowerGraph* by up to *5.53X*
- Also much faster than other systems like *GraphX*, *Giraph*, and *CombBLAS* for ingress and runtime
- Random hybrid-cut improves *GraphX* by 1.33X