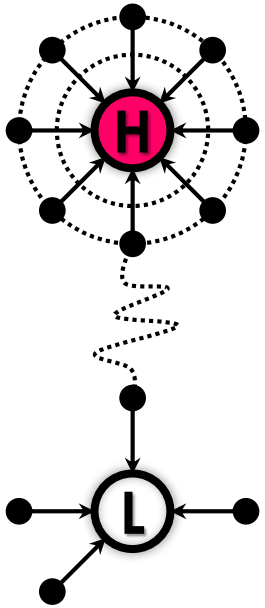
PowerLyra

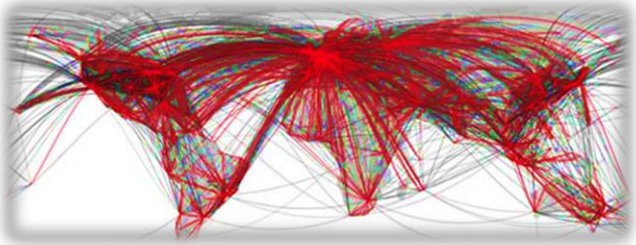
Differentiated Graph Computation and Partitioning on Skewed Graphs



RONG CHEN, JIAXIN SHI, YANZHE CHEN, HAIBO CHEN

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

Graphs are Everywhere[1]

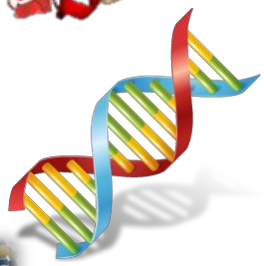


Traffic network graphs

- Monitor/react to road incidents
- Analyze Internet security

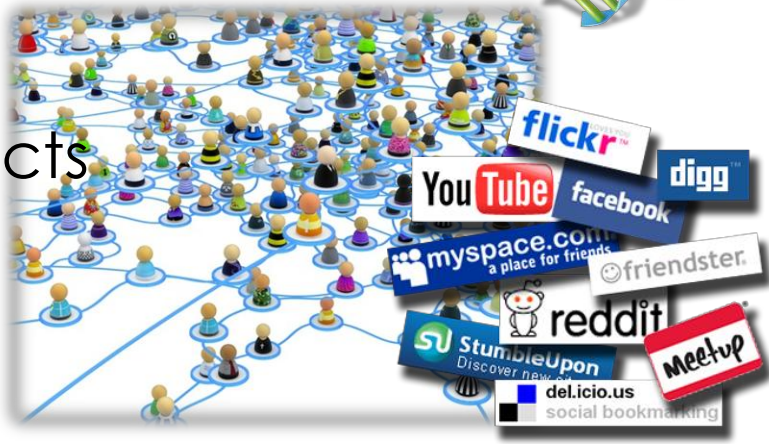
Biological networks

- Predict disease outbreaks
- Model drug-protein interactions



Social networks

- Recommend people/products
- Track information flow



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

Graph-parallel Processing

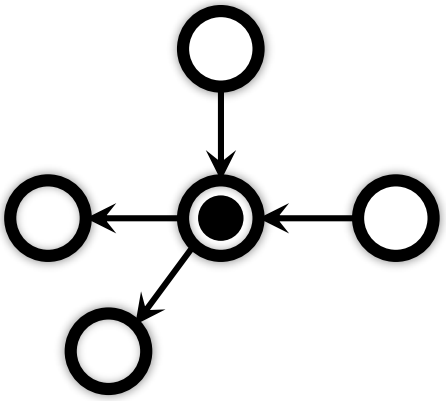
Coding graph algorithms as vertex-centric programs to process vertices in parallel and communicate along edges

-- "*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

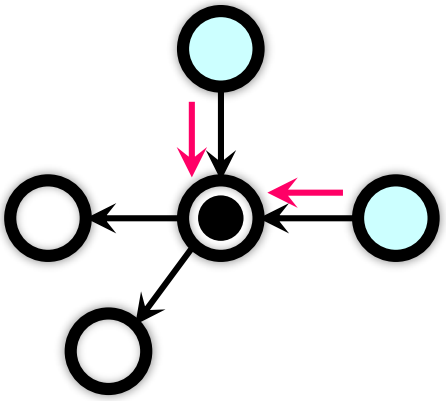
Coding graph algorithms as vertex-centric programs to process vertices in parallel and communicate along edges

-- "*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

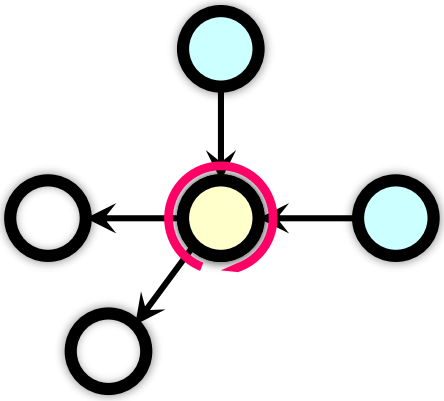
Coding graph algorithms as vertex-centric programs to process vertices in parallel and communicate along edges

-- "*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

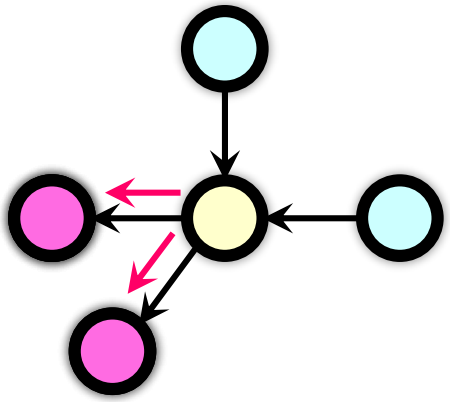
Coding graph algorithms as vertex-centric programs to process vertices in parallel and communicate along edges

-- "*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

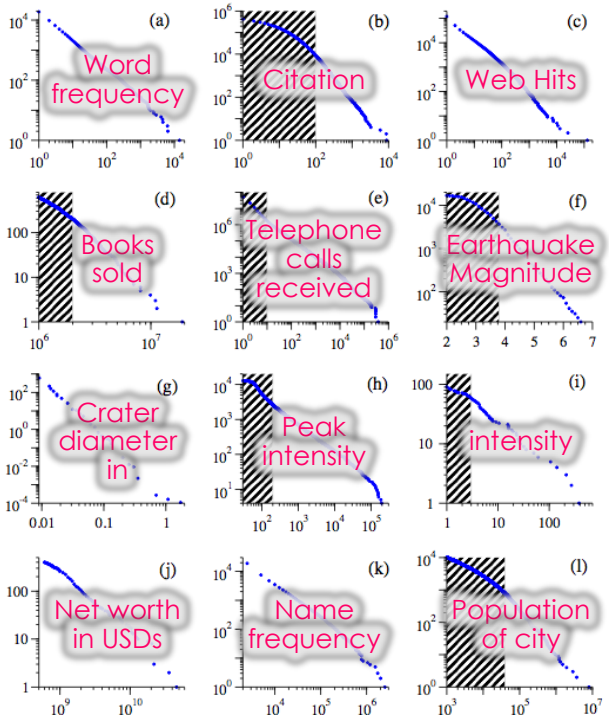
```
COMPUTE(v)
  foreach n in v.in_nbrs
    sum += n.rank / n.nedges
  v.rank = 0.15 + 0.85 * sum
  if !converged(v)
    foreach n in v.out_nbrs
      activate(n)
```

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]



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

Case: SINA *Weibo*(微博)

- 167 million active users[1]

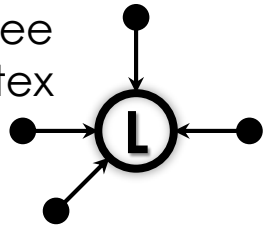
#Follower : 77,971,093 76,946,782 ··· 14,723,818 ····· 397 ····· 69 ·····

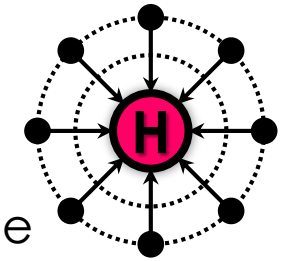
[1] http://www.chinainternetwatch.com/10735/weibo-q3-2014/

Challenge: LOCALITY vs. PARALLELISM

Low-degree vertex

397





High-degree vertex

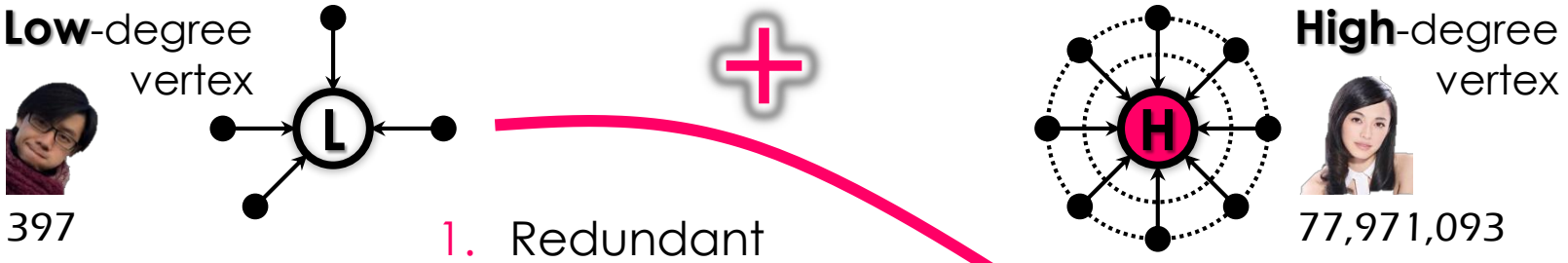
77,971,093

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

LOCALITY

make resource locally accessible to hidden network latency

Challenge: LOCALITY vs. PARALLELISM



LOCALITY

make resource locally accessible to hidden network latency

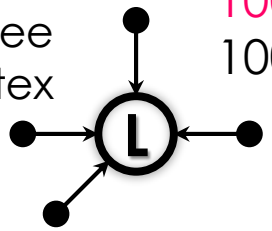
PARALLELISM

evenly parallelize the workloads to avoid load imbalance

Challenge: LOCALITY vs. PARALLELISM

Low-degree vertex

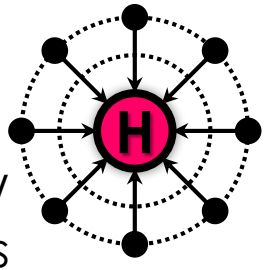
397



100-million users w/ 100 followers

Dilemma

100 users w/ 100-million followers



High-degree vertex

77,971,093

LOCALITY

make resource locally accessible to hidden network latency

Conflict

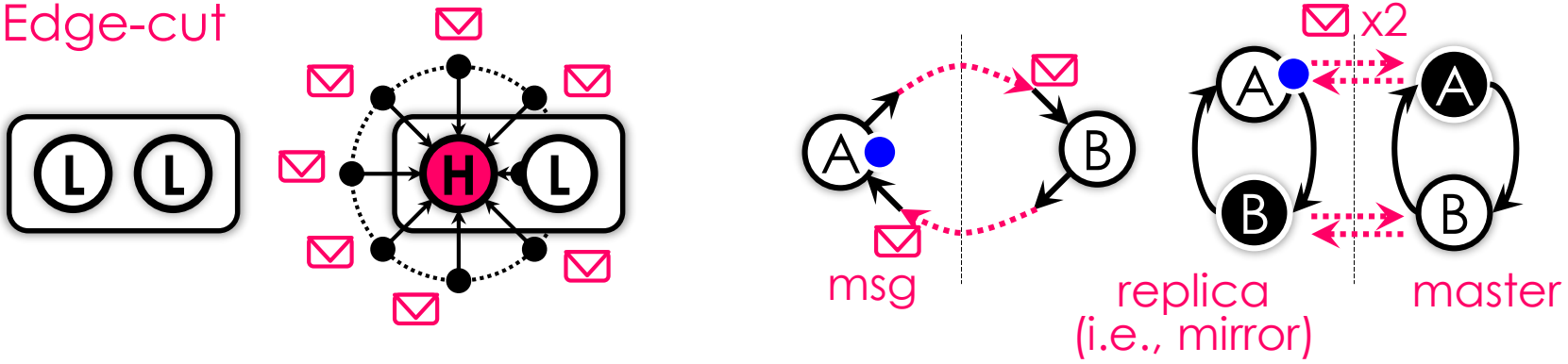
PARALLELISM

evenly parallelize the workloads to avoid load imbalance

Existing Efforts: One Size Fits All

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

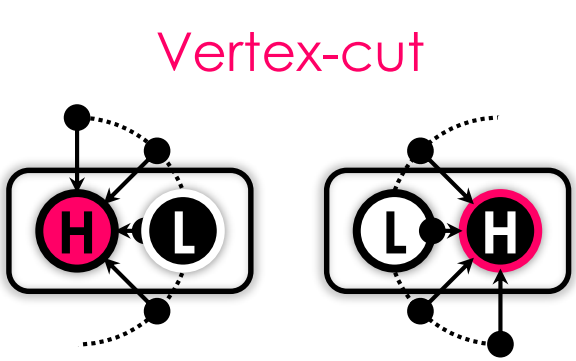
- Focus on exploiting LOCALITY
- Partitioning: use edge-cut to evenly assign vertices along with all edges
- Computation: aggregate all resources (i.e., msgs or replicas) of a vertex on local machine

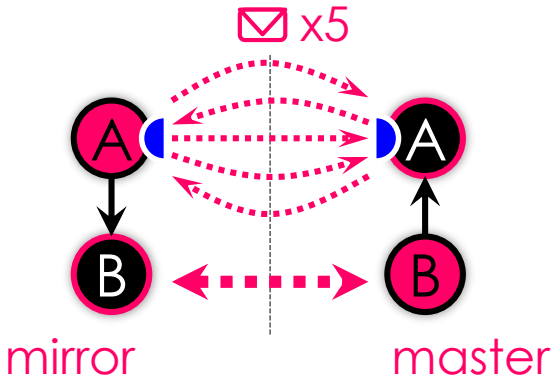


Existing Efforts: One Size Fits All

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

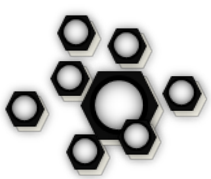




Contributions

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



Agenda

Graph Partitioning

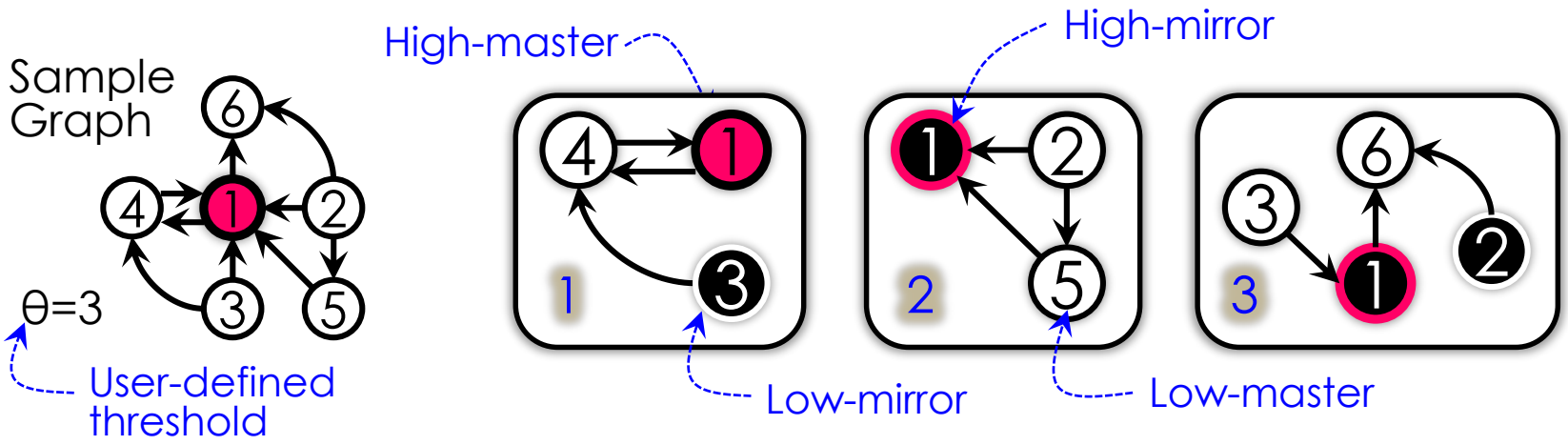
Graph Computation

Evaluation

Graph Partitioning

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

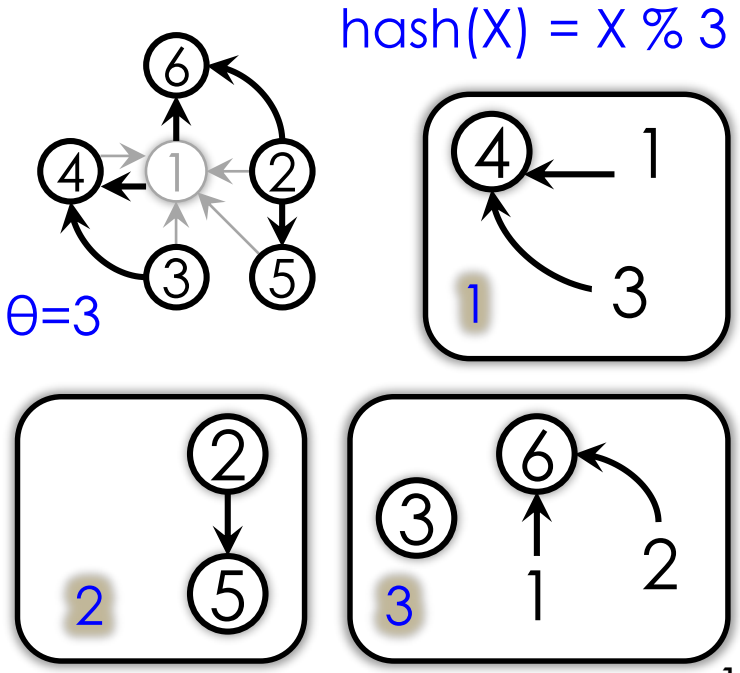
- Low-cut (inspired by *edge-cut*): reduce mirrors and exploit locality for low-degree vertices
- High-cut (inspired by *vertex-cut*): provide balance and restrict the impact of high-degree vertices
- Efficiently combine low-cut and high-cut



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

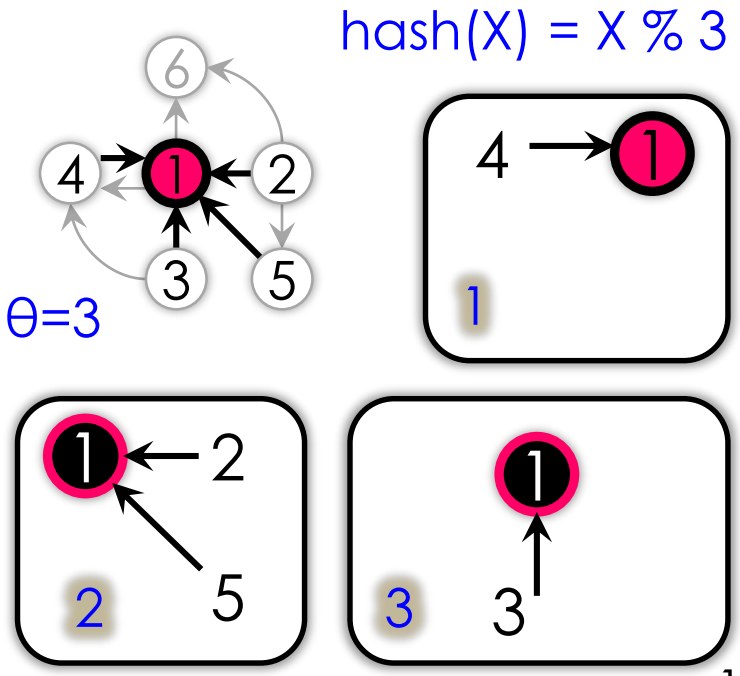


NOTE: *A new heuristic, called Ginger, is provided to further optimize Random hybrid-cut. (see paper)*

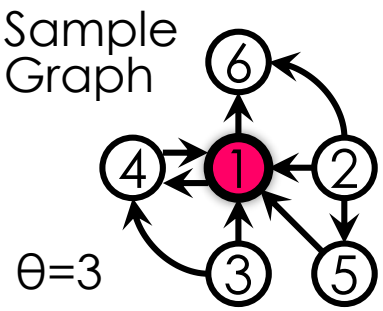
Hybrid-cut (High)

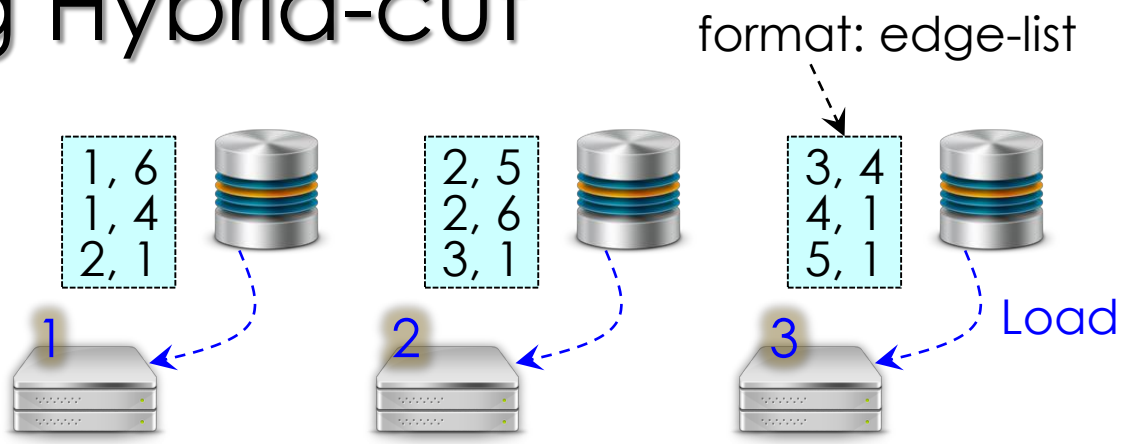
High-cut: distribute one-direction edges (e.g., *in-edges*) of high-degree vertices to machines by hashing opposite (e.g., *source*) vertex

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)



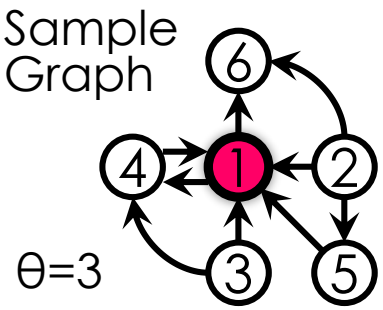
Constructing Hybrid-cut

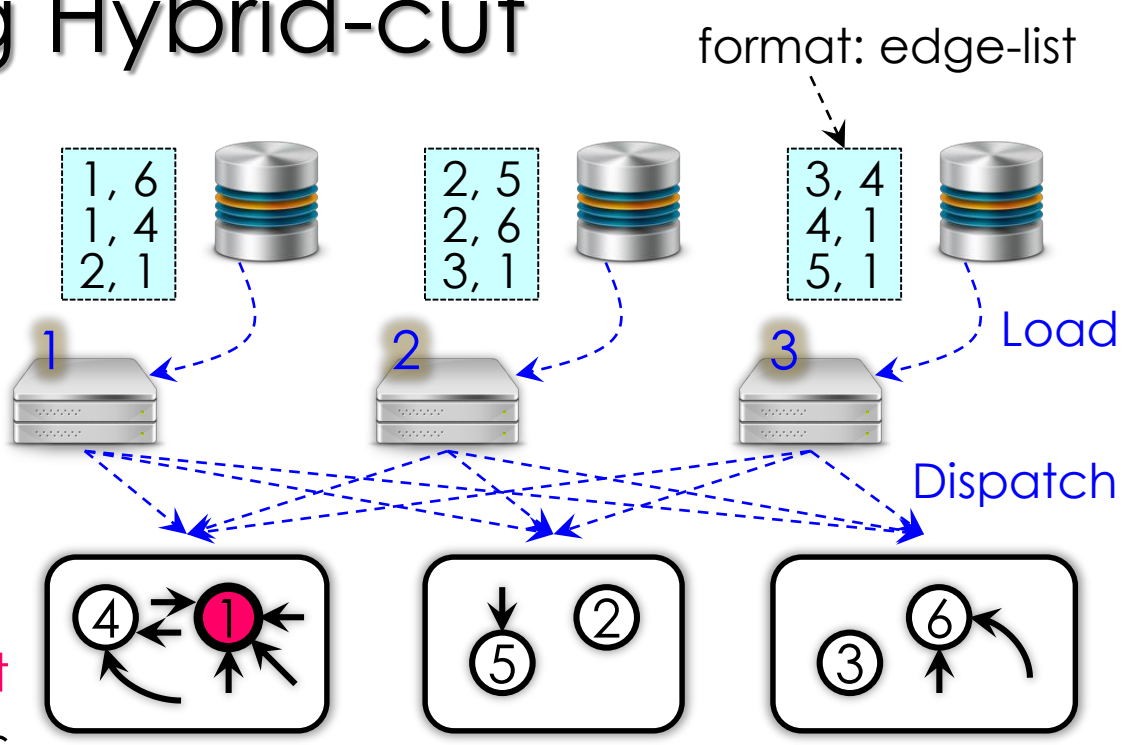




1. Load files in parallel

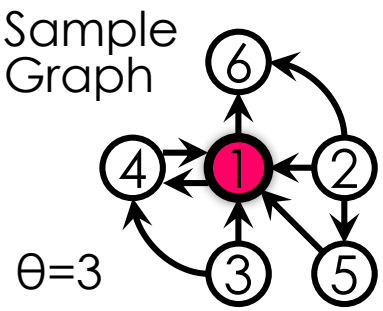
Constructing Hybrid-cut



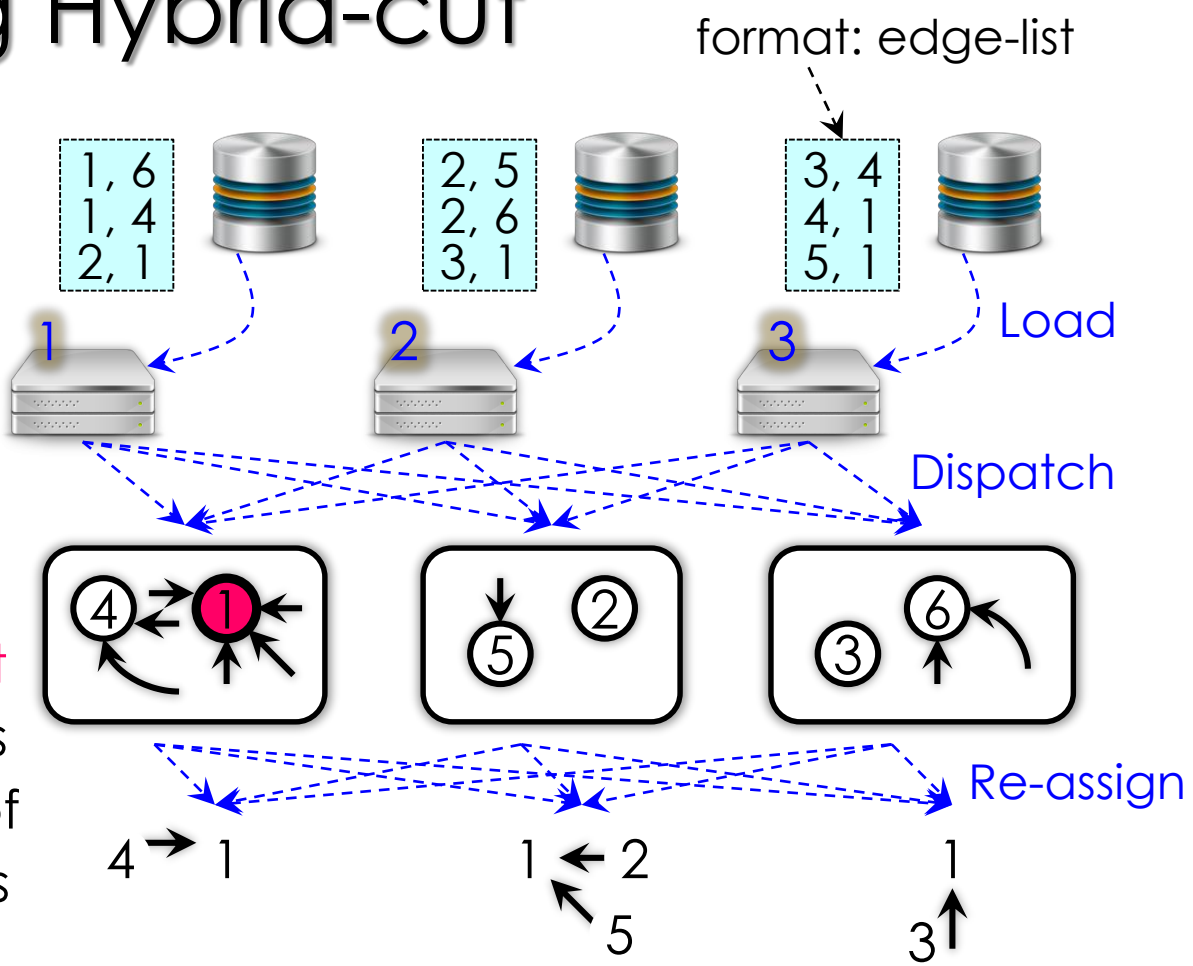


1. Load files in parallel
2. Dispatch all edges (Low-cut) and count in-degree of vertices

Constructing Hybrid-cut

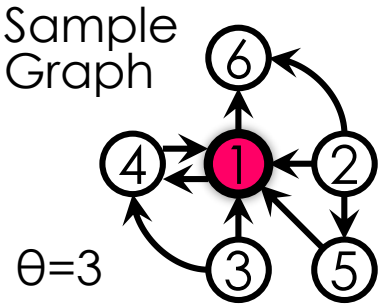


1. Load files in parallel
2. Dispatch all edges (Low-cut) and count in-degree of vertices
3. Re-assign in-edges of high-degree vertices (High-cut, θ=3)

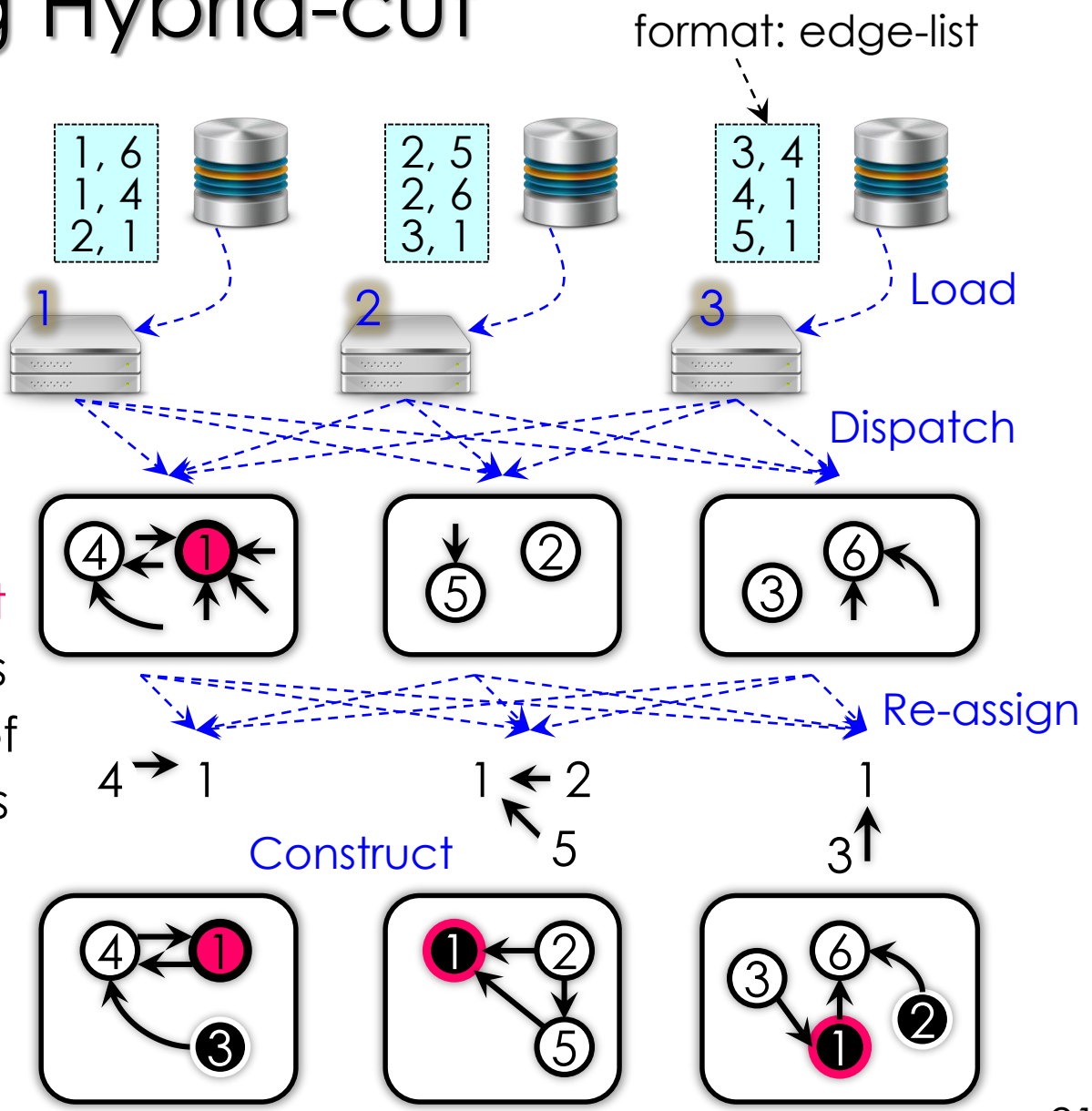


NOTE: *some formats (e.g., adjacent list) can skip counting and re-assignment*

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, θ=3)
4. Create mirrors to construct a local graph



Agenda

Graph Partitioning

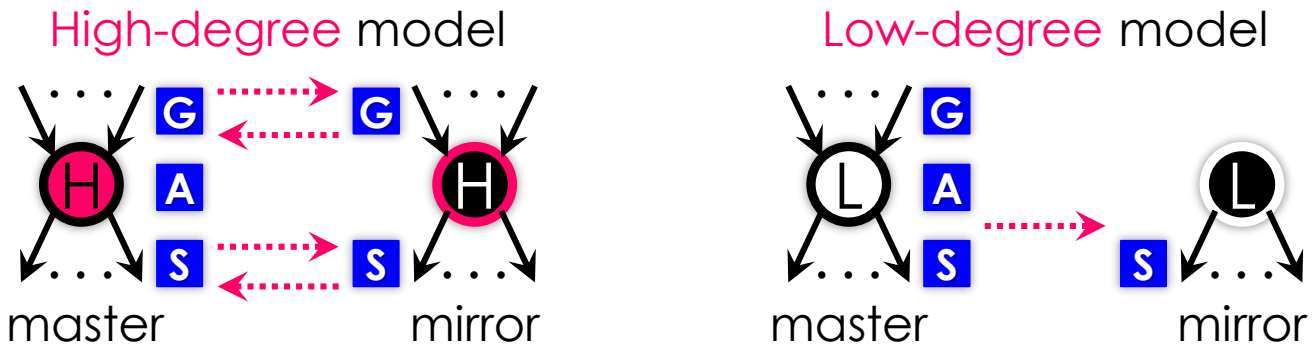
Graph Computation

Evaluation

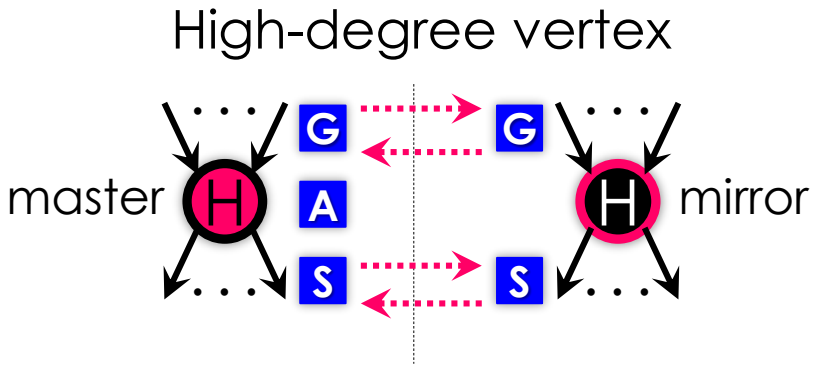
Graph Computation

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: ≤ 4 x #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
```

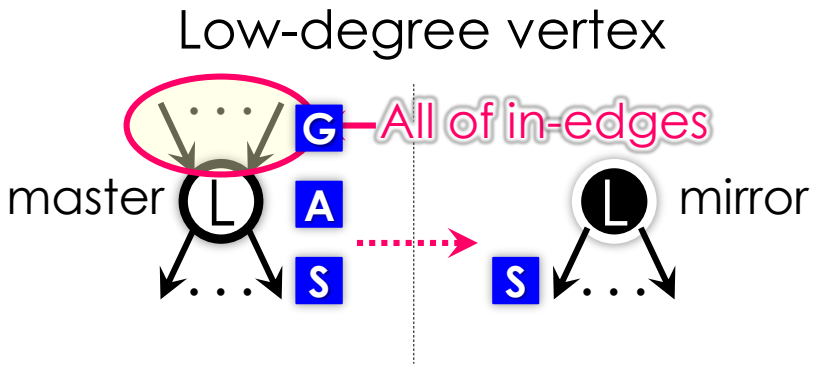
A — local

```
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: ≤ 1 x #mirrors

local

G

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

local

A

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

distributed

S

```
SCATTER_EDGES = OUT
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

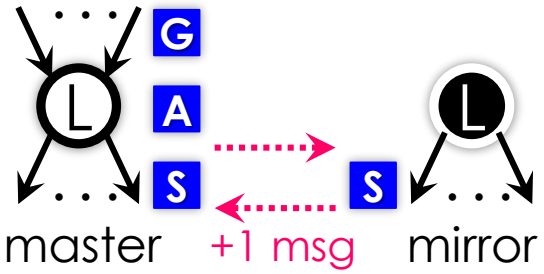
- Adaptively downgrade model (+ N msgs, $1 \leq N \leq 3$)
- Detect at runtime w/o any changes to APPs

e.g., Connected Components (CC)

→ G/NONE and S/ALL

→ +1 msg in SCATTER (mirror → master)

→ Comm. Cost: ≤ 2 x #mirrors



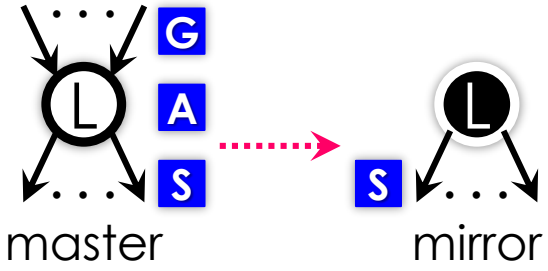
Downgrade

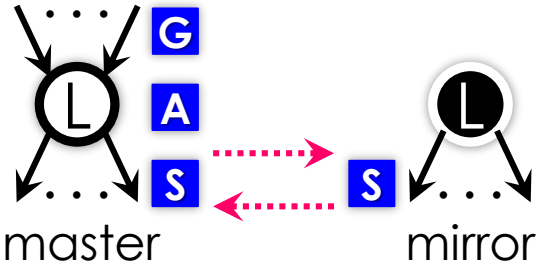
OBSERVATION:

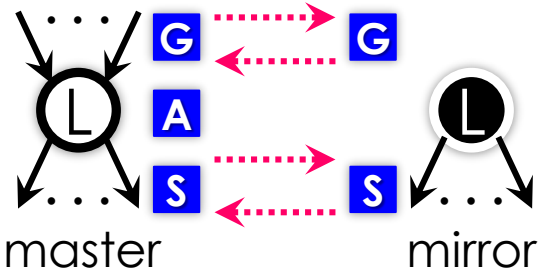
most algorithms only gather or scatter in one direction

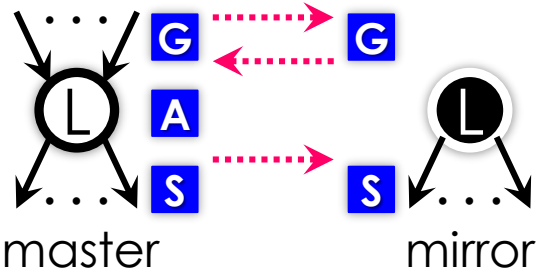
#msgs (SCATTER \ GATHER)	ALL	IN	OUT	NONE
ALL	4	2	2	2
IN	3	2	1	1
OUT	3	1	2	1
NONE	3	1	1	1

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 GraphX[1]

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 (Grid/Oblivious/Coordinated)

Platforms 48-node 6-node

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

Algorithms and graphs

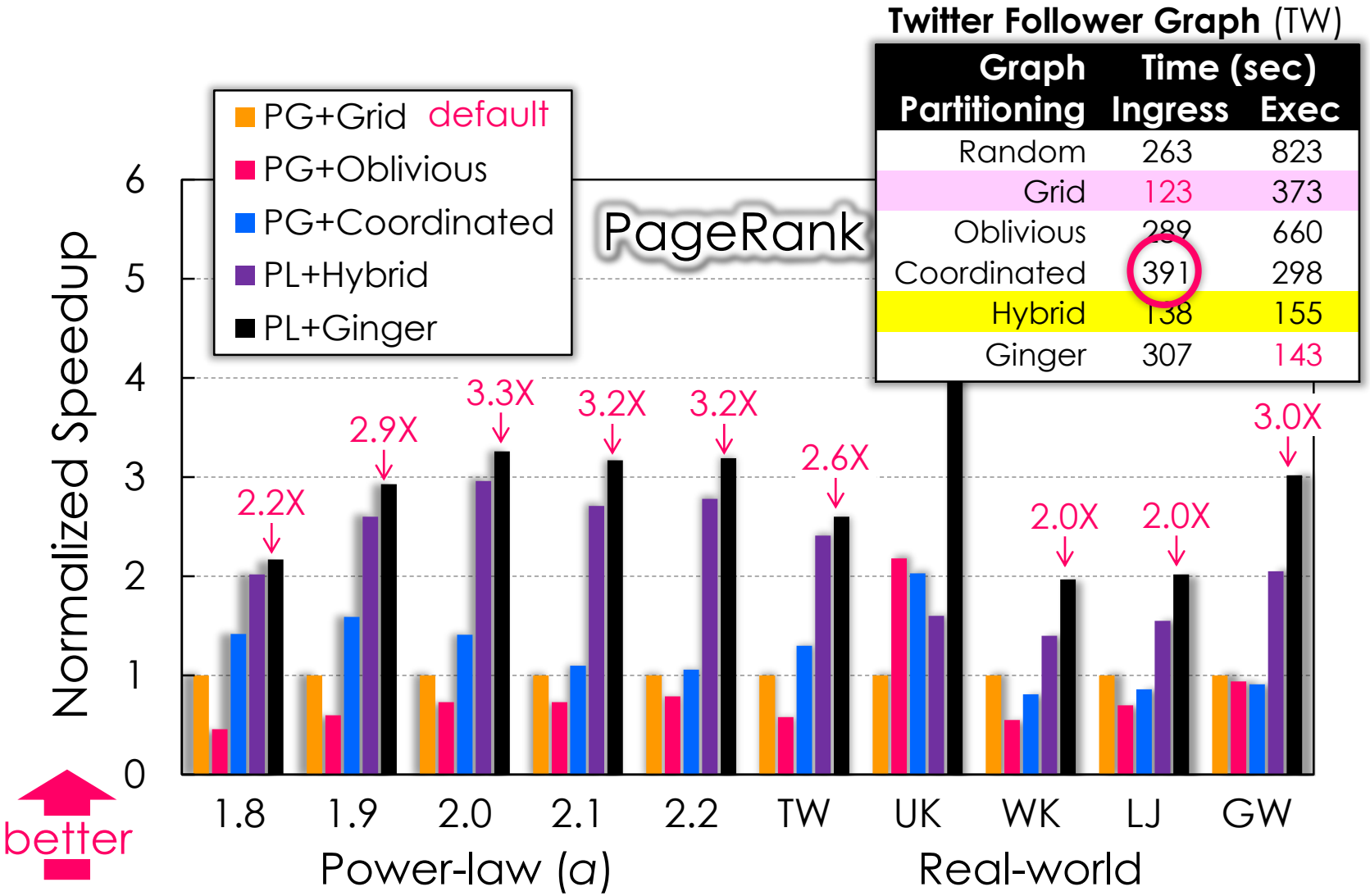
- 3 graph-analytics algorithms (PR[2], CC, DIA) and 2 MLDM algorithms (ALS, SGD)
- 5 real-world graphs, 5 synthetic graphs[3], etc.

Graph	|V|	|E|
Twitter	42M	1.47B
UK-2005	40M	936M
Wiki	5.7M	130M
LJournal	5.4M	79M
GWeb	0.9M	5.1M

α	|E|
1.8	673M
1.9	249M
2.0	105M
2.1	54M
2.2	39M

[1] Clusters are connected via 1GigE
[2] We run 10 iterations for PageRank.
[3] Synthetic graphs has fixed 10 million vertices

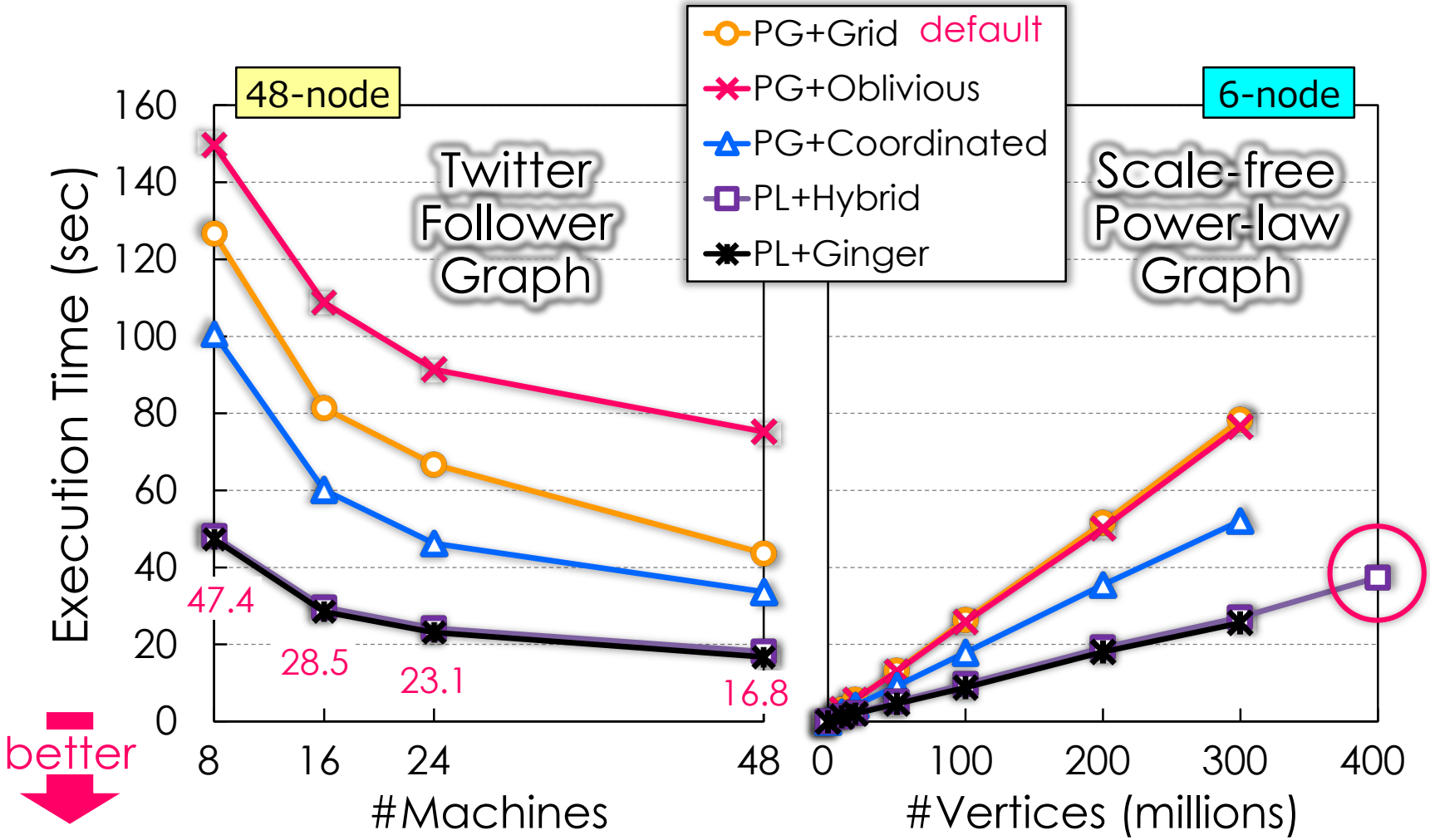
Performance



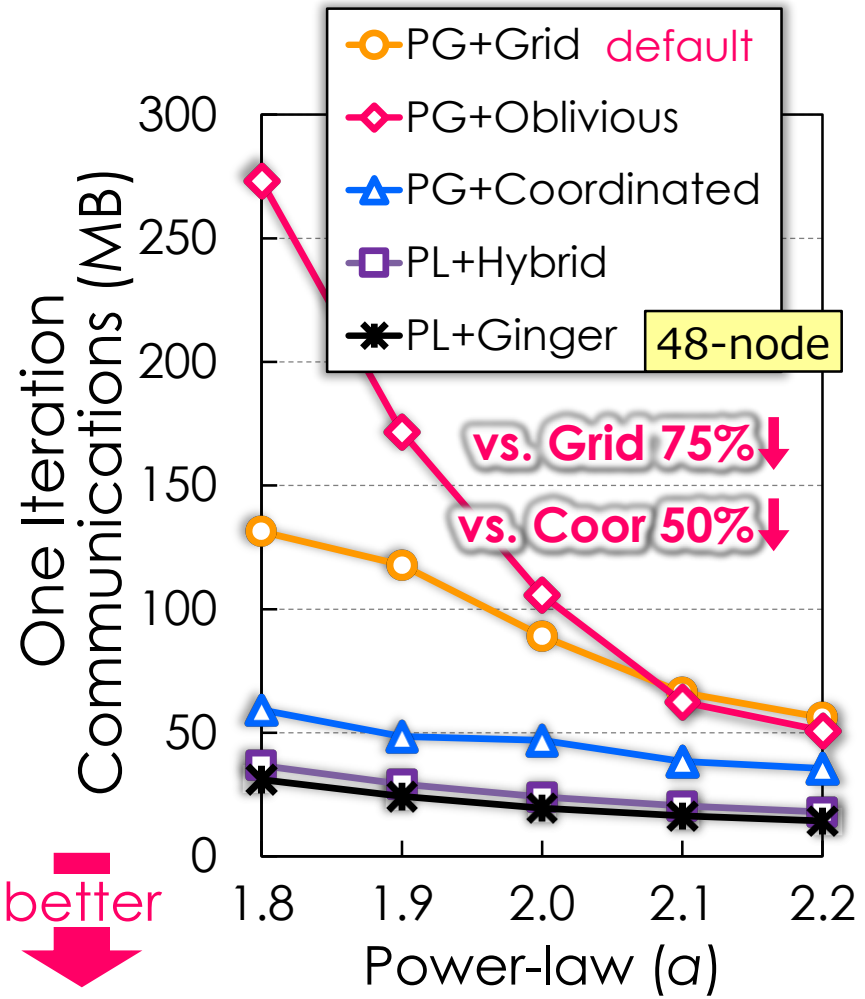
Twitter Follower Graph (TW)

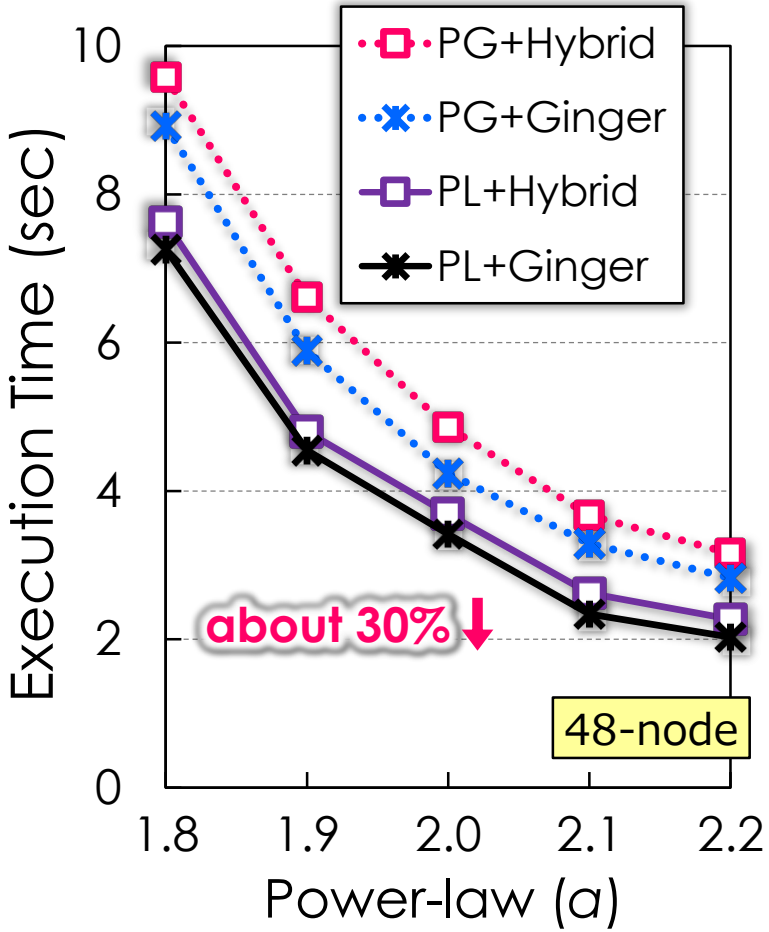
Graph Partitioning	Time (sec) Ingress	Exec
Random	263	823
Grid	123	373
Oblivious	289	660
Coordinated	391	298
Hybrid	138	155
Ginger	307	143

Scalability



Breakdown


PG+Grid default
PG+Oblivious
PG+Coordinated
PL+Hybrid
PL+Ginger
48-node
vs. Grid 75%
vs. Coor 50%
One Iteration Communications (MB)
300
250
200
150
100
50
0
better
1.8
1.9
2.0
2.1
2.2
Power-law (α)

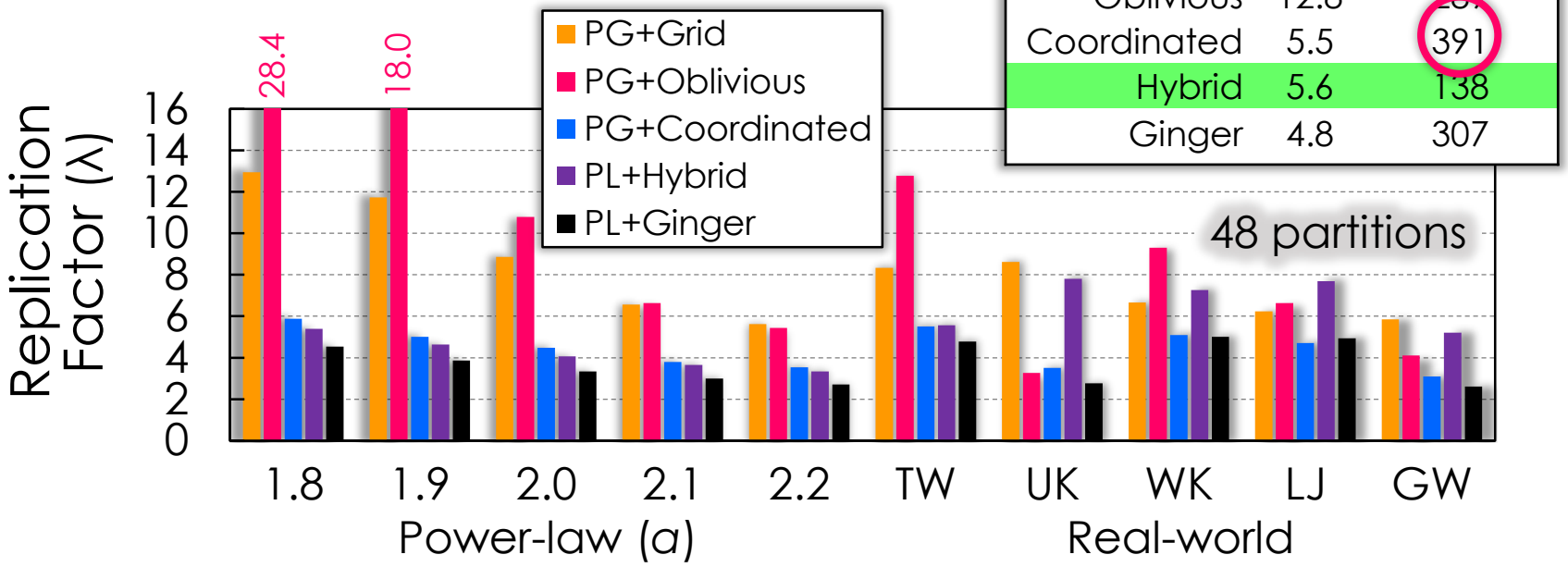

PG+Hybrid
PG+Ginger
PL+Hybrid
PL+Ginger
about 30%
48-node
Execution Time (sec)
10
8
6
4
2
0
1.8
1.9
2.0
2.1
2.2
Power-law (α)

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	
Coordinated	Greedy	Great	Bad	
Hybrid(R)	Hashing	Good	Great	
Ginger	Greedy	Great	Bad	

Twitter Follower Graph (TW)

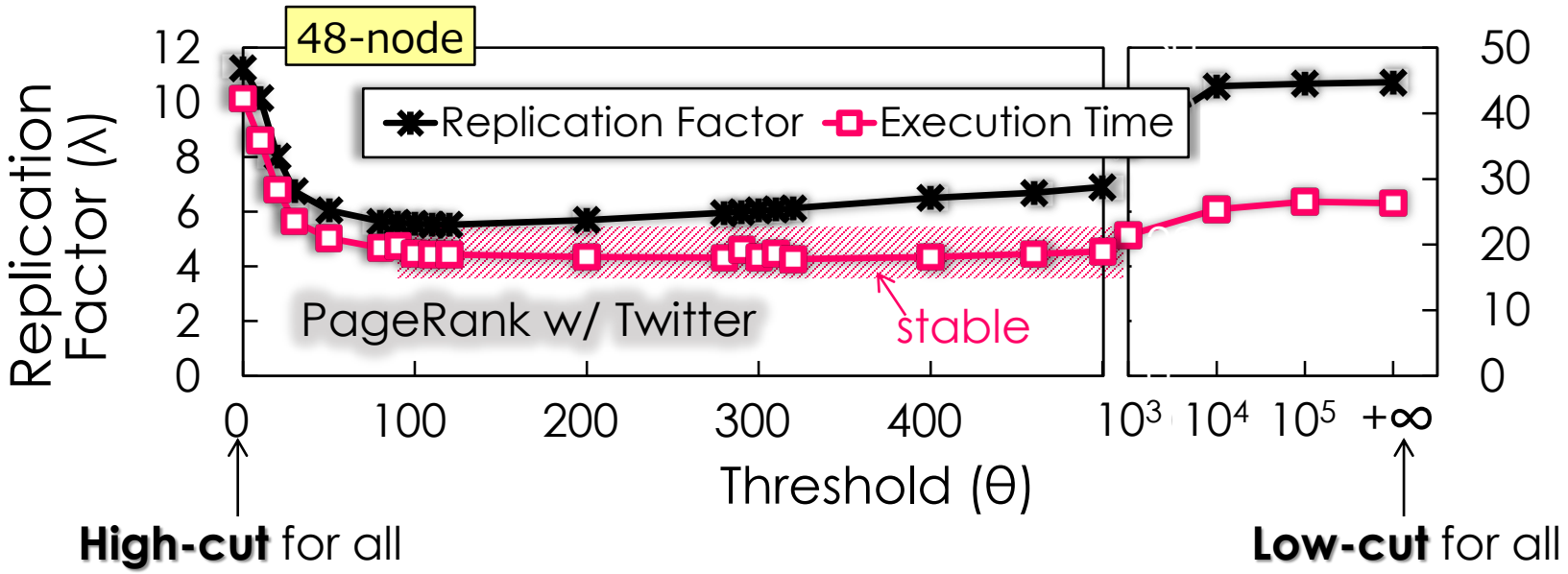
Partitioning	λ	Ingress
Random	16.0	263
Grid	8.3	123
Oblivious	12.8	289
Coordinated	5.5	391
Hybrid	5.6	138
Ginger	4.8	307



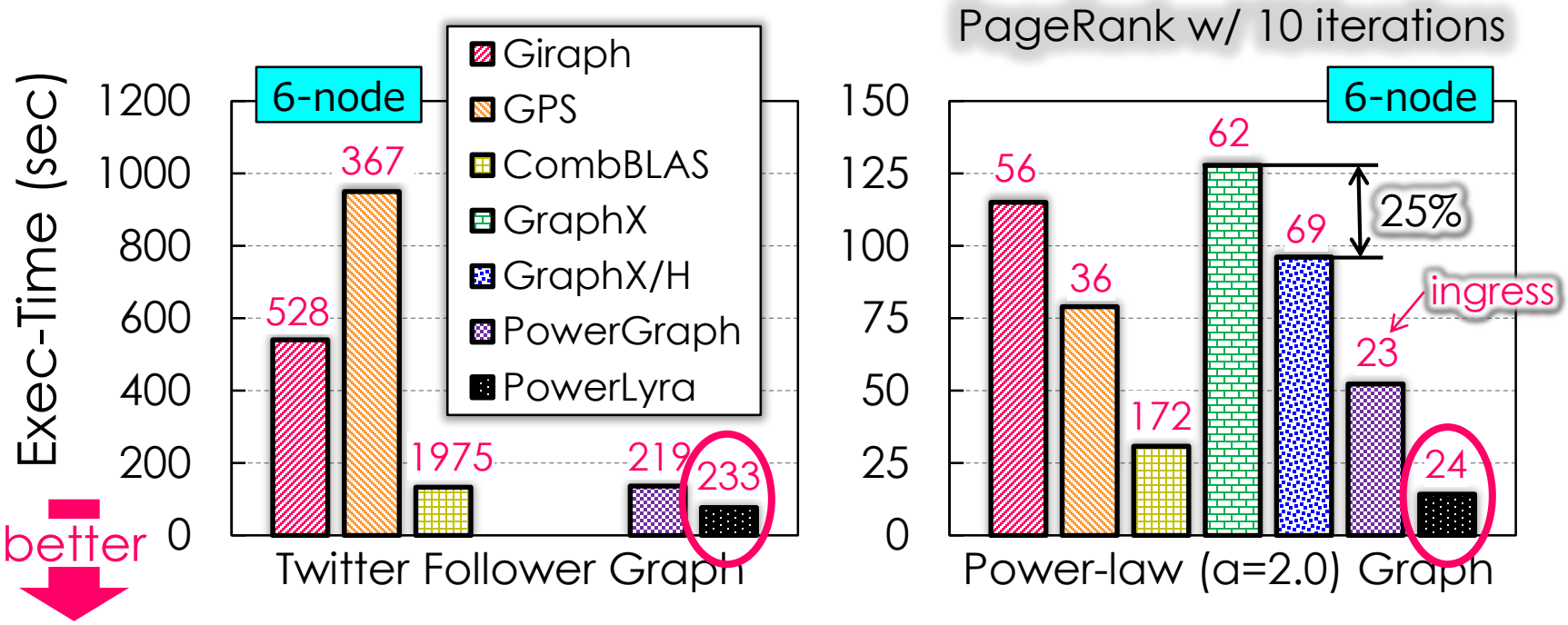
Threshold

Impact of different thresholds

- Using high-cut or low-cut for all vertices is poor
- Execution time is stable for a large range of θ
  - → Difference is lower than 1sec (100 ≤ θ ≤ 500)



vs. Other Systems[1]



PageRank with 10 iterations

6-node

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

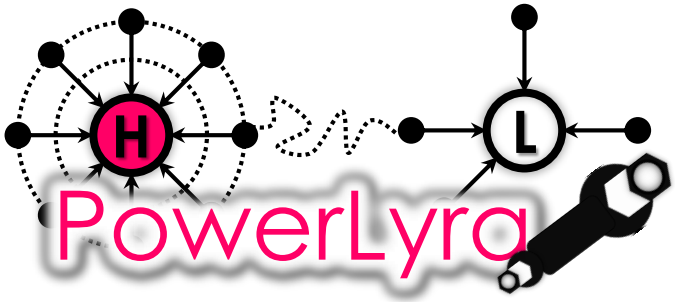
Existing systems use a "*ONE SIZE FITS ALL*" design for skewed graphs, resulting in suboptimal performance

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

Embrace LOCALITY and PARALLELISM

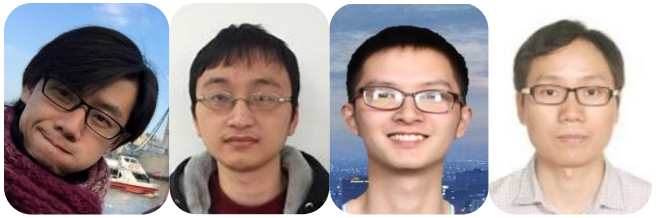
- Outperform *PowerGraph* by up to 5.53X
- Also much faster than other systems like *GraphX*, *Giraph*, and *CombBLAS* for ingress and runtime
- Random hybrid-cut improves GraphX by 1.33X

Thanks



http://ipads.se.sjtu.edu.cn/projects/powerlyra.html

Institute of Parallel and Distributed Systems



Questions

Related Work

Other graph processing systems

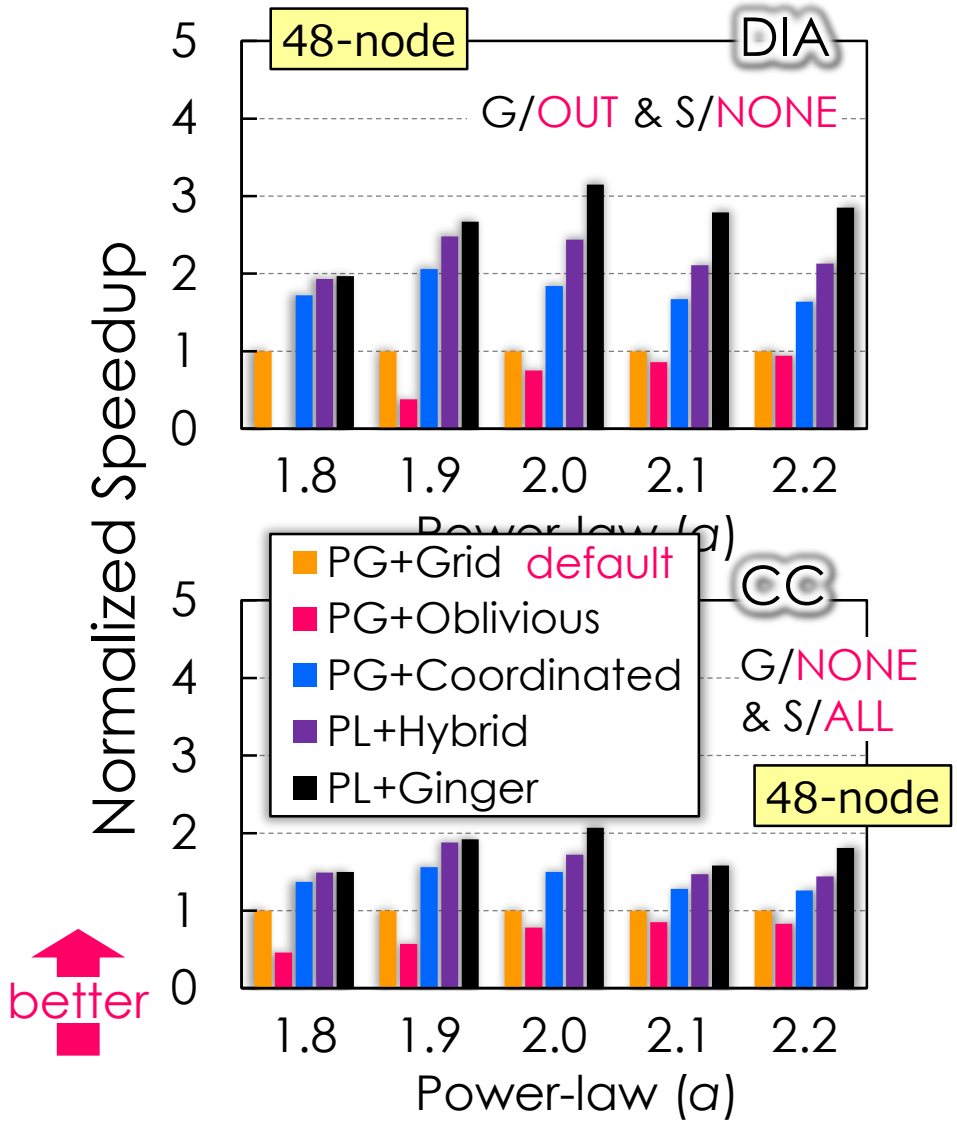
GENERAL	Pregel[SIGMOD'09], GraphLab[VLDB'12], PowerGraph[OSDI'12], GraphX[OSDI'14]
OPTIMIZATION	Mizan[EuroSys'13], Giraph++[VLDB'13], PowerSwitch[PPoPP'15]
SINGLE MACHINE	GraphChi[OSDI'12], X-Stream[SOSP'13], Galois[SOSP'13], Polymer[PPoPP'15]
OTHER PARADIGMS	CombBLAS[IJHPCA'11], SociaLite[VLDB'13]

Graph replication and partitioning

EDGE-CUT	Stanton *et al.* [SIGKDD'12], Fennel[WSDM'14]
VERTEX-CUT	Greedy[OSDI'12], GraphBuilder[GRADES'13]
HEURISTICS	2D[SC'05], Degree-Based Hashing[NIPS'15]

Backup

Other Algorithms and Graphs



MLDM Algorithms 48-node

Netflix: |V| =0.5M, |E| =99M

V_DATA	E_DATA	Grid	Hybrid
8d + 13	16	12.3	2.6

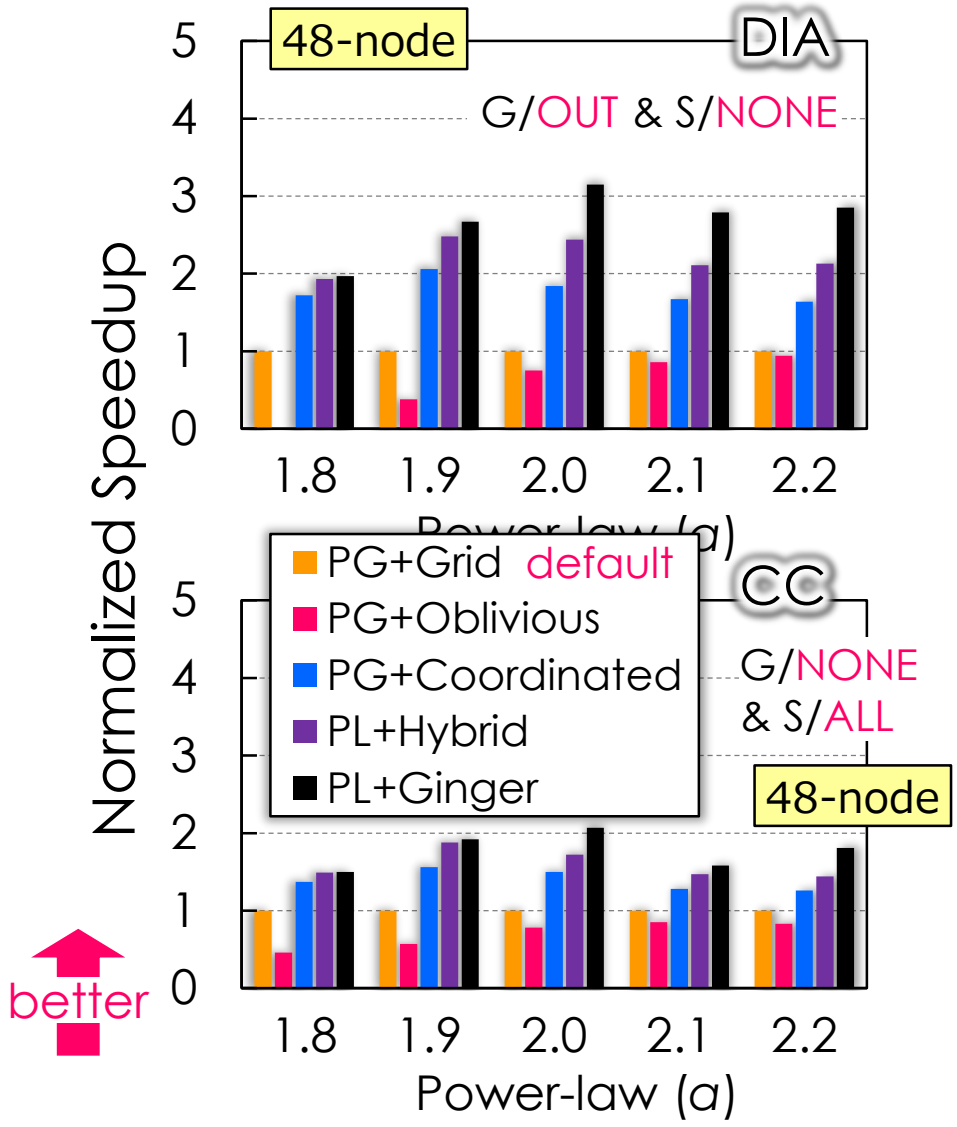
ALS	d=5	d=20	d=50	d=100
PG	10 | 33	11 | 144	16 | 732	Failed
PL	13 | 23	13 | 51	14 | 177	15 | 614

SGD	d=5	d=20	d=50	d=100
PG	15 | 35	17 | 48	21 | 73	28 | 115
PL	16 | 26	19 | 33	19 | 43	20 | 59

Latent dimension

ingress | runtime

Other Algorithms and Graphs



MLDM Algorithms — 48-node

Netflix: |V|=0.5M, |E|=99M

V_DATA	E_DATA	Grid	Hybrid
8d + 13	16	12.3	2.6

ALS	d=5	d=20	d=50	d=100
PG	10 | 33	11 | 144	16 | 732	Failed
PL	13 | 23	13 | 51	14 | 177	15 | 614
SGD	**d=5**	**d=20**	**d=50**	**d=100**
PG	15 | 35	17 | 48	21 | 73	28 | 115
PL	16 | 26	19 | 33	19 | 43	20 | 59

Non-skewed Graph — 48-node

RoadUS: |V|=23.9M, |E|=58.3M

Graph Partitioning	λ	Time (sec) Ingress	Exec
Grid	3.2	15.5	57.3
Oblivious	2.3	13.8	51.8
Coordinated	2.3	26.9	50.4
Hybrid	3.3	14.0	32.2
Ginger	2.8	28.8	31.3

Ingress Time vs. Execution Time

PageRank

Twitter Follower Graph			
Graph Partitioning	λ	Time (sec) Ingress	Exec
Random	16.0	263	823
Grid	8.3	123	373
Oblivious	12.8	289	660
Coordinated	5.5	391	298
Hybrid	5.6	138	155

ALS (d=20)

Netflix Movie Recommendation			
Graph Partitioning	λ	Time (sec) Ingress	Exec
Random	36.9	21	547
Grid	12.3	12	174
Oblivious	31.5	25	476
Coordinated	5.3	31	105
Hybrid	2.6	14	67

Fast CPU & Networking

Testbed: 6-node physical clusters

1. SLOW: 24 Cores (AMD 6168 1.9GHz L3:12MB), 64GB RAM, 1GbE
2. FAST: 20 Cores (Intel E5-2650 2.3GHz L3:24MB), 64GB RAM, 10GbE