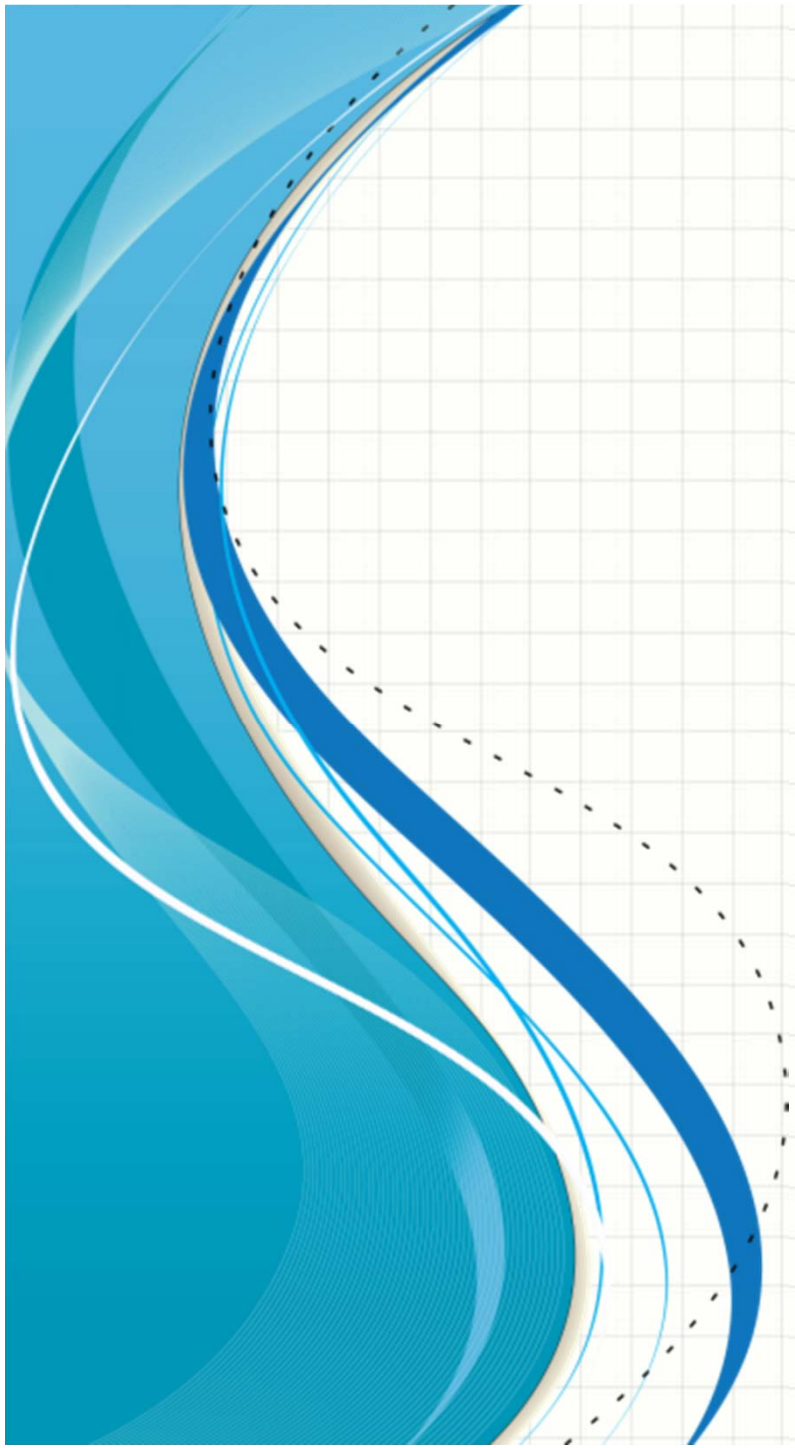


GRAPHCHI

Patrick Short

Thursday, November 13th

GraphChi(huahua) Overview

- The Punchline
- Quick Overview
- Novel Method
  - Parallel sliding windows
- Use Cases and Caveats

GraphChi is in the ballpark with massive distributed systems

- 50% slower than shared-memory GraphLab for three iterations of PageRank.
- 40% slower than Spark (50 machines, 100 CPUs vs 1 Machine 2 CPUs) on five iterations of PageRank (twitter-2010 data set)
- Triangle counting in *twitter-2010* data set completes in 400 minutes on Hadoop-based algorithm (90 minutes on GraphChi)

Vertex-centric, asynchronous updates on evolving graphs (in a single PC).

- Created in parallel with GraphLab and uses **vertex-centric update function**.
- **Dynamic Selective Scheduling** (not covered in detail, but supported)
- Edges (but not vertices) can be added or removed.

Random Access Problem must be solved for disk storage approach.

- Graph is stored simultaneously in compressed sparse row and compressed sparse column (efficient out-edge and in-edge loading)
- Graph must be split into shards in a *clever* way -> parallel sliding window approach.

Parallel sliding window introduced to solve Random Access Problem.

- Large graphs are written to disk.
- Vertices are separated into shards:

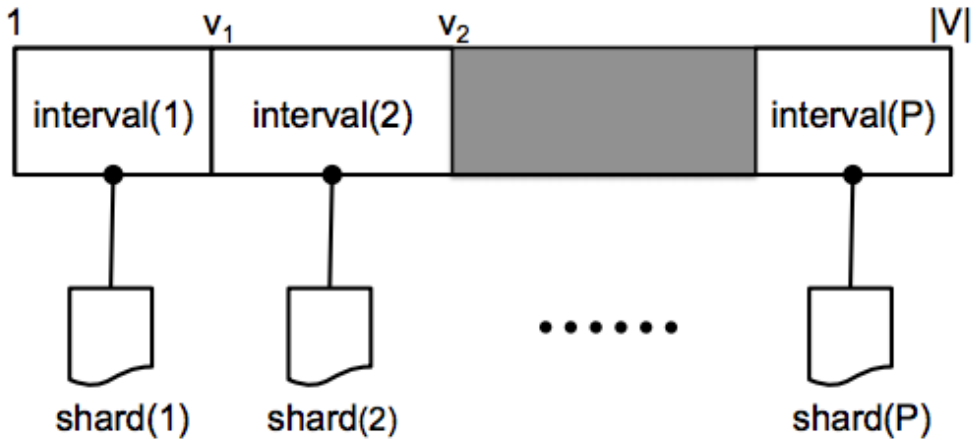


Figure 1: The vertices of graph (V, E) are divided into P intervals. Each interval is associated with a shard, which stores all edges that have destination vertex in that interval.

Parallel sliding window introduced to solve Random Access Problem.

- Large graphs are written to disk.
- Vertices are separated into shards:

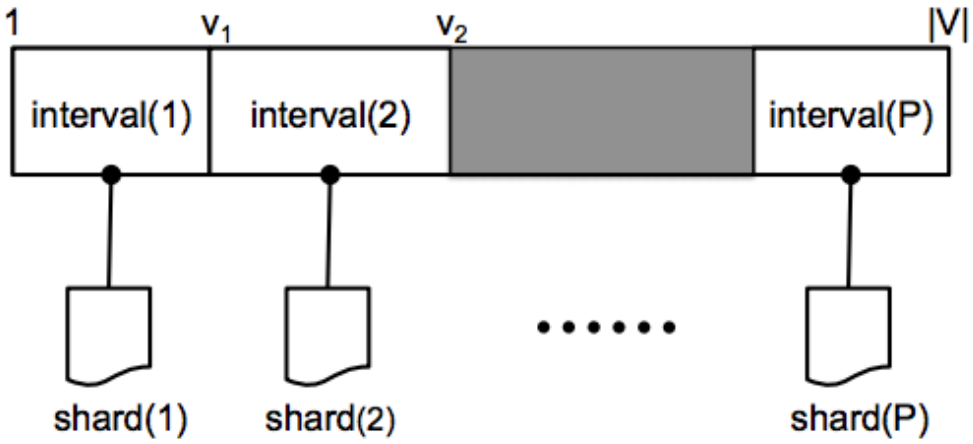
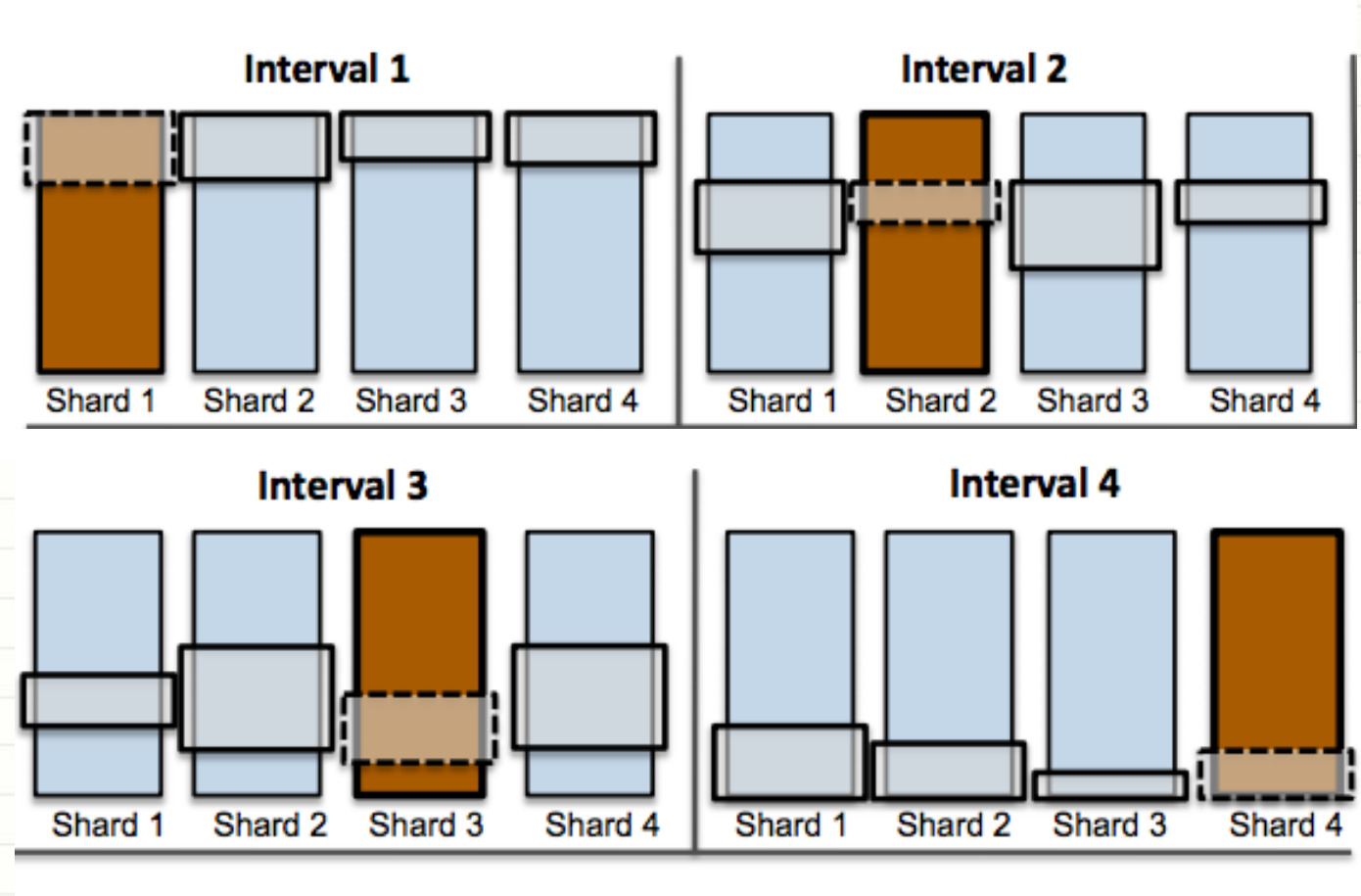


Figure 1: The vertices of graph (V, E) are divided into P intervals. Each interval is associated with a shard, which stores all edges that have destination vertex in that interval.

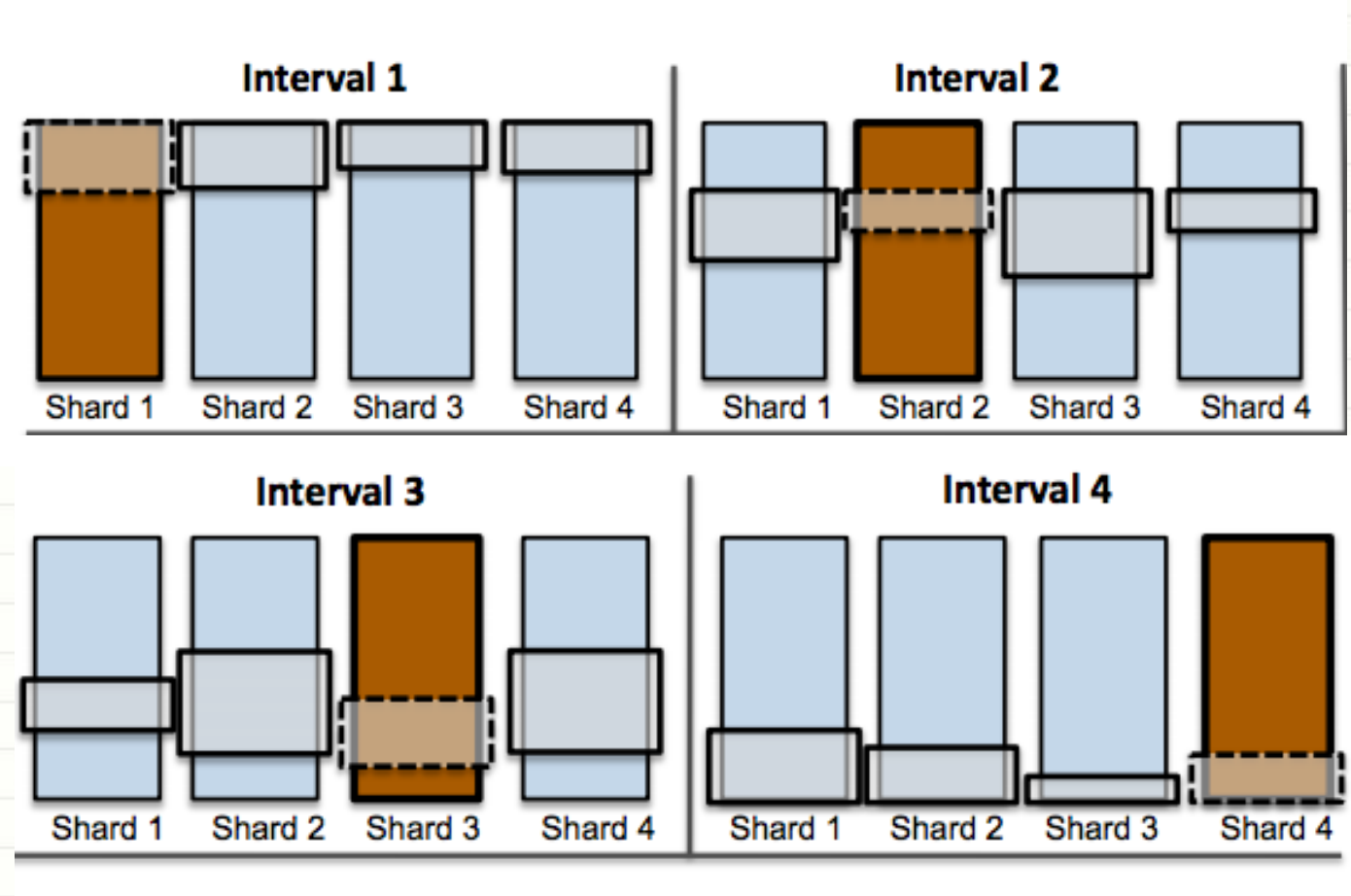
Visualizing the PSW Method

- In edges are read from dark (memory) shard, out edges read from window on disk shards.



Visualizing the PSW Method

- Edges are ordered by **source** within each shard (this is the key).



Shard 1

src	dst	value
1		
	2	0.3
3		
	2	0.2
4		
	1	1.4
5		
	1	0.5
	2	0.6
6		
	2	0.8

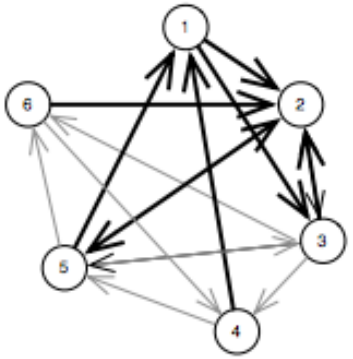
Shard 2

src	dst	value
1		
	3	0.4
2		
	3	0.3
3		
	4	0.8
5		
	3	0.2
6		
	4	1.9

Shard 3

src	dst	value
2		
	5	0.6
3		
	5	0.9
	6	1.2
4		
	5	0.3
5		
	6	1.1

(a) Execution interval (vertices 1-2)



(b) Execution interval (vertices 1-2)

Shard 1

src	dst	value
1		
	2	0.273
3		
	2	0.22
4		
	1	1.54
5		
	1	0.55
	2	0.66
6		
	2	0.88

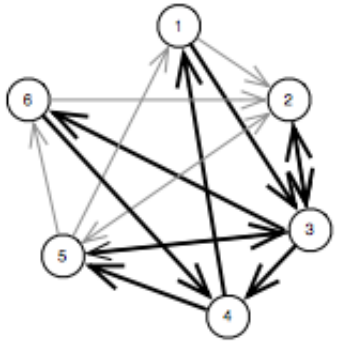
Shard 2

src	dst	value
1		
	3	0.364
2		
	3	0.273
3		
	4	0.8
5		
	3	0.2
6		
	4	1.9

Shard 3

src	dst	value
2		
	5	0.545
3		
	5	0.9
	6	1.2
4		
	5	0.3
5		
	6	1.1

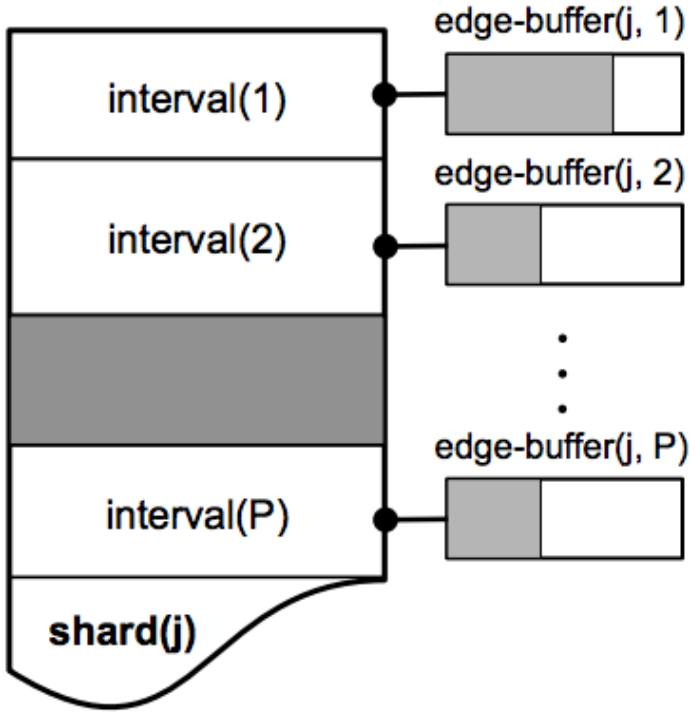
(c) Execution interval (vertices 3-4)



(d) Execution interval (vertices 3-4)

Evolving Graphs

- Shard ordering and edge buffers allow for removal or addition of edges.



Use Cases

- This system was developed alongside GraphLab and relies on a similar vertex-centric model.
- Two major use cases:
  - Exploratory data analysis
  - Tool for building and debugging applications *before* deploying to a high performance cluster.

Caveats

- PowerGraph (presentation forthcoming) still knocks GraphChi out of the park (30 – 40x) performance.
- The paper presented does not truly assess worst-case scenario performance.

Performance

Application & Graph	Iter.	Comparative result	GraphChi (Mac Mini)	Ref
Pagerank & domain	3	GraphLab[31] on AMD server (8 CPUs) **87 s**	**132 s**	-
Pagerank & twitter-2010	5	Spark [48] with 50 nodes (100 CPUs): **486.6 s**	**790 s**	[42]
Pagerank & V=105M, E=3.7B	100	Stanford GPS, 30 EC2 nodes (60 virt. cores), **144 min**	approx. **581 min**	[41]
Pagerank & V=1.0B, E=18.5B	1	Piccolo, 100 EC2 instances (200 cores) **70 s**	approx. **26 min**	[40]
Webgraph-BP & yahoo-web	1	Pegasus (Hadoop) on 100 machines: **22 min**	**27 min**	[24]
ALS & netflix-mm, D=20	10	GraphLab on AMD server: **4.7 min**	**9.8 min** (in-mem) **40 min** (edge-repl.)	[31]
Triangle-count & twitter-2010	-	Hadoop, 1636 nodes: **423 min**	**60 min**	[43]
Pagerank & twitter-2010	1	PowerGraph, 64 x 8 cores: **3.6 s**	**158 s**	[21]
Triange-count & twitter- 2010	-	PowerGraph, 64 x 8 cores: **1.5 min**	**60 min**	[21]

Table 2: **Comparative performance.** Table shows a selection of recent running time reports from the literature.

Performance

Application & Graph	Iter.	Comparative result	GraphChi (Mac Mini)	Ref
Pagerank & domain	3	GraphLab[31] on AMD server (8 CPUs) **87 s**	**132 s**	-
Pagerank & twitter-2010	5	Spark [48] with 50 nodes (100 CPUs): **486.6 s**	**790 s**	[42]
Pagerank & V=105M, E=3.7B	100	Stanford GPS, 30 EC2 nodes (60 virt. cores), **144 min**	approx. **581 min**	[4]
Pagerank & V=1.0B, E=18.5B	1	Piccolo, 100 EC2 instances (200 cores) **70 s**	approx. **26 min**	[4( ]
Webgraph BP & yahoo-web	1	Pegasus (Hadoop) on 100 machines: **22 min**	**27 min**	[2]
ALS & netflix-mm, D=20	10	GraphLab on AMD server: **4.7 min**	**9.8 min** (in-mem) **40 min** (edge-repl.)	[31]
Triangle-count & twitter-2010	-	Hadoop, 1636 nodes: **423 min**	**60 min**	[43]
Pagerank & twitter-2010	1	PowerGraph, 64 x 8 cores: **3.6 s**	**158 s**	[21]
Triange-count & twitter- 2010	-	PowerGraph, 64 x 8 cores: **1.5 min**	**60 min**	[21]

Table 2: **Comparative performance.** Table shows a selection of recent running time reports from the literature.

One iteration, 26 minutes

Questions?