GraphChi

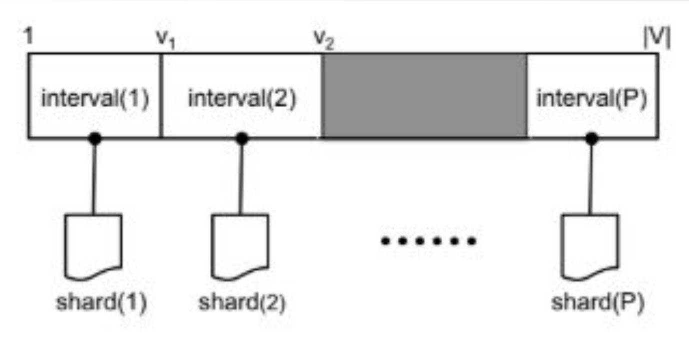
Steven Krieg

Background & Big Idea

- Carnegie Mellon, 2012
- “Large-Scale Graph Computation on Just a PC”

How do we process graphs that exceed available memory?

The Solution: Secondary Storage



Graphs are divided into groups of vertices (intervals) and edges (shards).

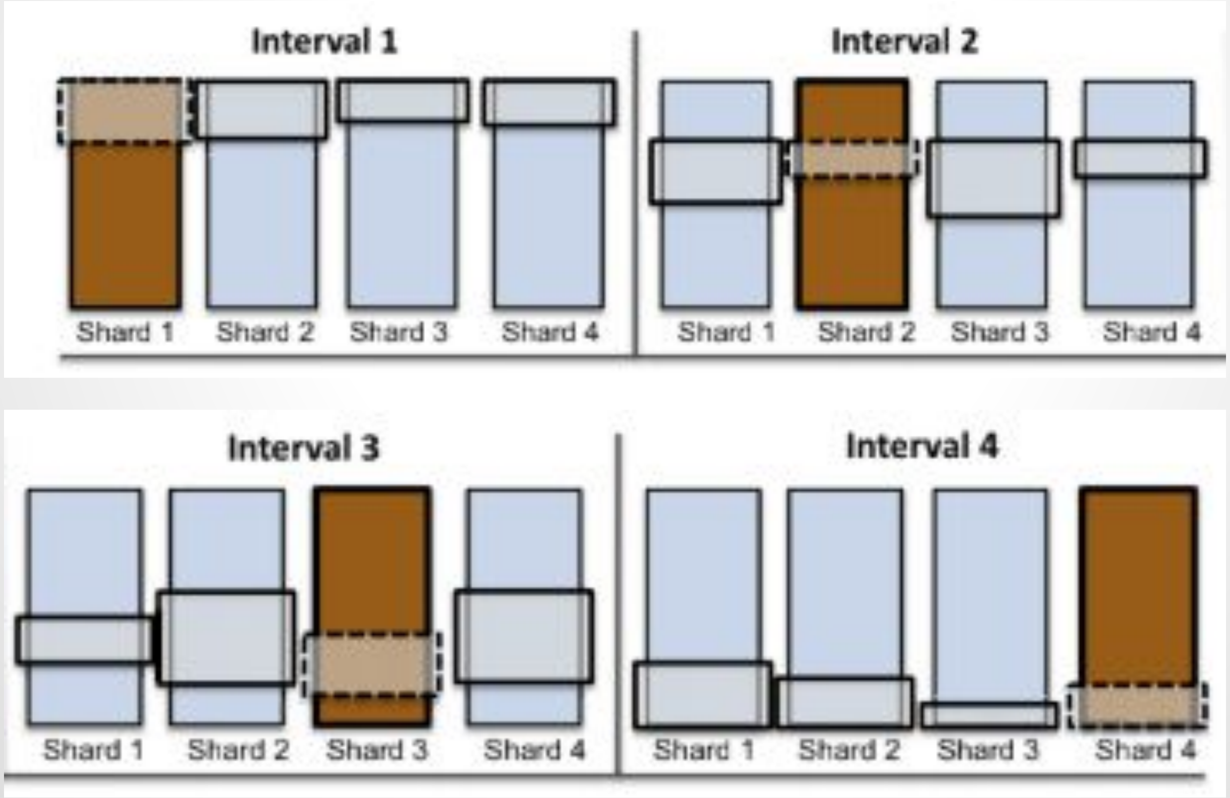
Intervals are loaded one at a time into memory for processing.

Interval: a group of vertices that will be updated in the same execution step

Shard: list of edges whose destination vertex is in the interval

1:1 relationship

“Parallel Sliding Windows”



“Parallel Sliding Windows”

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

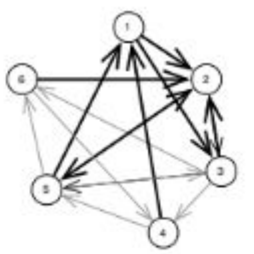
Shard 2

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

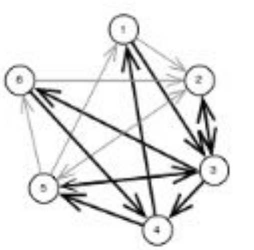
Shard 2

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)



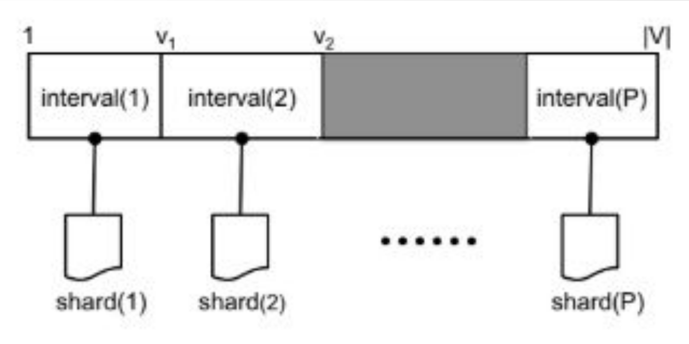
(d) Execution interval (vertices 3-4)

A Specific Purpose

Key performance metric: size (not time).

Use case: large-scale computation (look elsewhere for traversals or queries)

Graph Expression



Graphs are divided into groups of vertices (intervals) and edges (shards), which are processed as subgraphs.

Programmer can specify interval size, or default is ¼ available memory.

Interval: a group of vertices that will be updated in the same execution step

Shard: list of edges whose destination vertex is in the interval

1:1 relationship

Graph Primitives

Weighted, directed graphs.

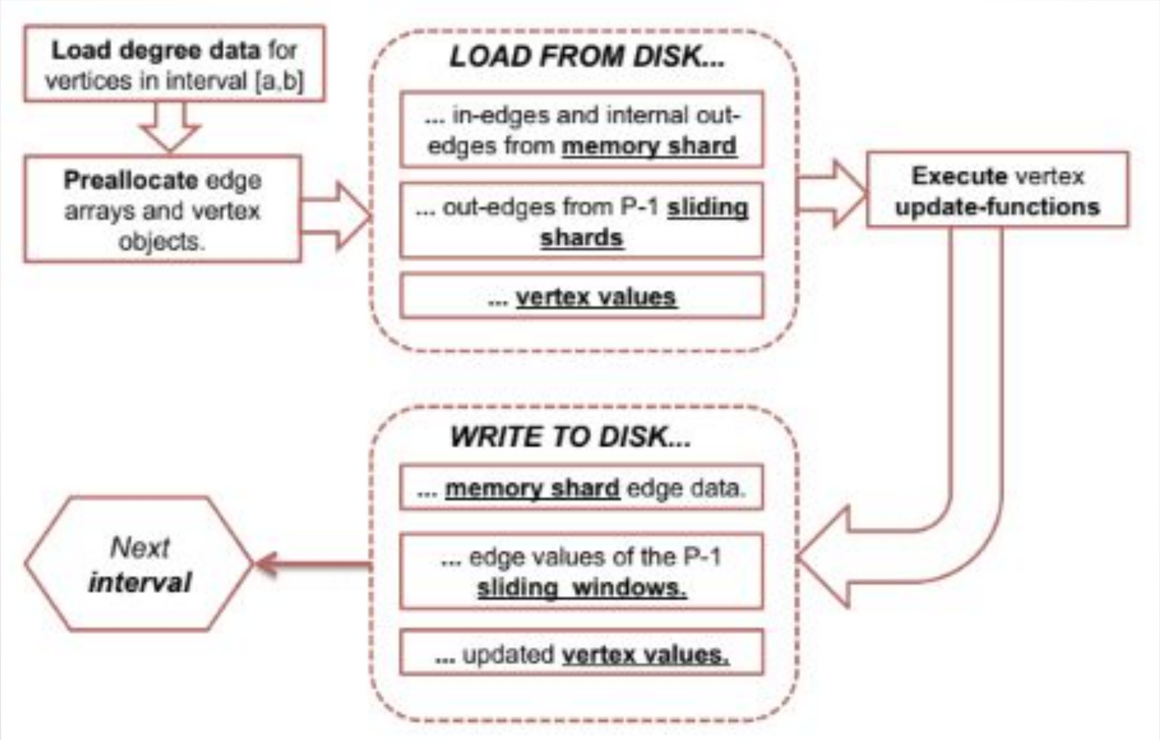
(You could in theory use unweighted or undirected graphs, but I’m guessing there are better frameworks for those)

Preprocessing

1. Divide vertices into intervals such that there is an approximately uniform in-degree distribution
2. Write each edge to a scratch file (shards)
3. Pass through each shard file and order edges
4. Compute a binary “degree file” with in- and out-degrees of each vertex

Can read from several standard graph formats.

Execution Model



How to Use (C++)

1. Extend GraphChiProgram class & template functions
2. Define parameters (memory budget, edge/vertex types, number of iterations, etc.)
3. Instantiate custom object and pass it to a graphchi_engine object

Sample Functions

```
before_iteration(int iteration, graphchi_context &gcontext)

after_iteration(int iteration, graphchi_context &gcontext)

before_exec_interval(vid_t window_st, vid_t window_en, graphchi_context &gcontext)

after_exec_interval(vid_t window_st, vid_t window_en, graphchi_context &gcontext)

update(vertex_t &v, graphchi_context &gcontext)
```

Example (Pagerank)

```
struct PagerankProgram : public GraphChiProgram<VertexDataType, EdgeDataType> {
...
void update(graphchi_vertex<VertexDataType, EdgeDataType> &v, graphchi_context &ginfo) {
...
       /* Compute the sum of neighbors' weighted pageranks by
              reading from the in-edges. */
           for(int i=0; i < v.num_inedges(); i++) {
               float val = v.inedge(i)->get_data();
               sum += val;
           }

           /* Compute my pagerank */
           float pagerank = RANDOMRESETPROB + (1 - RANDOMRESETPROB) * sum;
```

Example (Pagerank cont'd)

```
...
/* Write my pagerank divided by the number of out-edges to
              each of my out-edges. */
           if (v.num_outedges() > 0) {
               float pagerankcont = pagerank / v.num_outedges();
               for(int i=0; i < v.num_outedges(); i++) {
                   graphchi_edge<float> * edge = v.outedge(i);
                   edge->set_data(pagerankcont);
               }
           }
```

Performance

Application & Graph	Iter.	Comparative result	GraphChi (Mac Mini)	Ref
Pagerank & domain	3	GraphLab[30] on AMD server (8 CPUs) **87 s**	**132 s**	-
Pagerank & twitter-2010	5	Spark [45] with 50 nodes (100 CPUs): **486.6 s**	**790 s**	[38]
Pagerank & V=105M, E=3.7B	100	Stanford GPS, 30 EC2 nodes (60 virt. cores), **144 min**	approx. **581 min**	[37]
Pagerank & V=1.0B, E=18.5B	1	Piccolo, 100 EC2 instances (200 cores) **70 s**	approx. **26 min**	[36]
Webgraph-BP & yahoo-web	1	Pegasus (Hadoop) on 100 machines: **22 min**	**27 min**	[22]
ALS & netflix-mm, D=20	10	GraphLab on AMD server: **4.7 min**	**9.8 min** (in-mem) **40 min** (edge-repl.)	[30]
Triangle-count & twitter-2010	-	Hadoop, 1636 nodes: **423 min**	**60 min**	[39]
Pagerank & twitter-2010	1	PowerGraph, 64 x 8 cores: **3.6 s**	**158 s**	[20]
Triange-count & twitter- 2010	-	PowerGraph, 64 x 8 cores: **1.5 min**	**60 min**	[20]

Further Resources

[1] Aapo Kyrola, Guy E. Blelloch, & Carlos Guestrin. (2018). GraphChi: Large-Scale Graph Computation on Just a PC.
[2] https://github.com/GraphChi
[3] Moon, Seunghyeon, Lee, Jae-Gil, Kang, Minseo, Choy, Minsoo, & Lee, Jin-Woo. (2016). Parallel community detection on large graphs with MapReduce and GraphChi. Data & Knowledge Engineering, 104, 17-31.
[5] Lu, J., & Thomo, A. (2016). An experimental evaluation of giraph and GraphChi. Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on, 993-996.