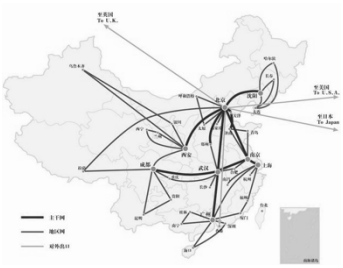
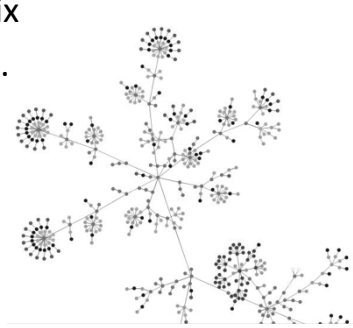
Single Node Machine Learning Systems - When big data meet Moore's Law

Wenguang CHEN
PACMAN Group
Department of Computer Science and Technology
Tsinghua University

Graph Computation

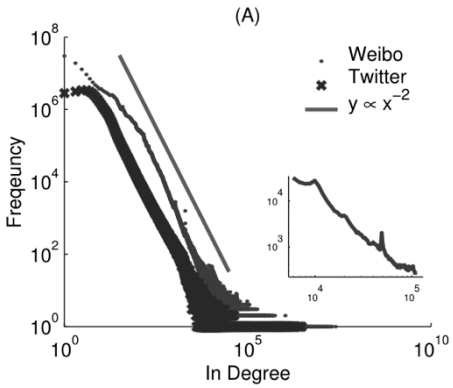
- Graph is one of the most general data structure and has many important applications
  - Twitter / Facebook / Weibo
  - Amazon user-item rating matrix
  - Bioinformatic / astrophysics / ..



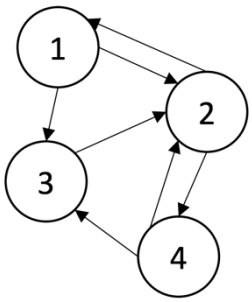


Challenges in graph computing

- Large scale
  - Billions vertices, trillions of edges
- Poor locality
  - Random access on vertices
- Irregular topology
  - Power-law distributic in real-world graphs



A graph example



Vertex:
1,2, 3, 4

Edges:
(1,2) (1,3)
(2,1) (2,4)
(4,2) (4,3)

For Vertex 1:
In edge: (2,1)
Out edge: (1,2) (1,3)

Current Graph Computing Systems

- Distributed systems
  - Pregel, GraphLab (PowerGraph), GraphX(Spark)
  - "Distributed" issues
    - Fault tolerance overhead
    - Load imbalance
    - Slow convergence
    - Unexpected performance problems

Current Graph Computing Systems

- Out-of-core systems
  - Use just one node to reduce the complexity involved in distributed systems.
  - Use SSDs/disks if the graph can not be fit into memory
    - The problem is how to reduce the number of I/O operations and the amount of I/O data
  - GraphChi, X-Stream, …
    - Use preprocessing to change the memory/disk access patterns from random accesses to sequential accesses
      - Reduce number of I/O operations
    - Performance compared with distributed systems
      - Reasonable yet not as fast as PowerGraph/GraphX.
    - Scalability
      - Partially

X-Stream : Edge−Centric model*

```
for each vertex v                    Scatter
  if v has update
    for each edge e from v
      scatter update along e
```

```
for each edge e                      Scatter
  If e.src has update
      scatter update along e
```

Vertex-Centric ⇨ **Edge-Centric**

*Roy, Amitabha, Ivo Mihailovic, and Willy Zwaenepoel. "X-stream: Edge-centric graph processing using streaming partitions." Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013.

7

Partition Edges and vertexs

- Shuffle edges with source vertex id(instead of sorting)
- Source vertex in memory

V1
1
2
3
4

SOURCE	DEST
1	5
4	7
2	7
4	3
4	8
3	8
2	4
1	3
3	2

Edges in disks/SSDs

V2
5
6
7
8

SOURCE	DEST
5	6
8	6
8	5
6	1

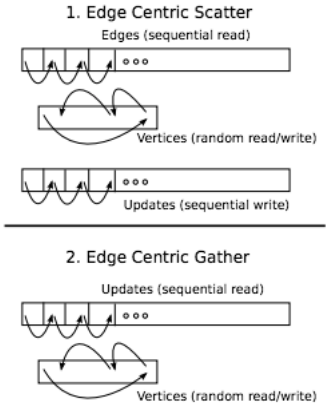
8

X-Stream: Edge Centric Computing with Partitions

```
scatter phase:
  for each streaming_partition p
    read in vertex set of p
    for each edge e in edge list of p
      edge_scatter(e): append update to Uout

shuffle phase:
  for each update u in Uout
    let p = partition containing target of u
    append u to Uin(p)
  destroy Uout

gather phase:
  for each streaming_partition p
    read in vertex set of p
    for each update u in Uin(p)
      edge_gather(u)
    destroy Uin(p)
```



The problems of X-Stream

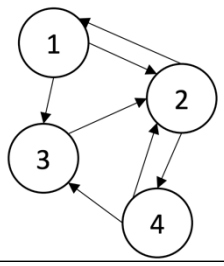
- X-Stream
  - Separate gather/scatter phase, with shuffle between them, introduce many extra I/O operations
  - Poor performance for BFS and WCC-like algorithms
    - Only a small portion of edges are touched for some iterations, but they would scan all edges

GridGraph

- A 2-level partitioning graph data structure
  - Improve locality
  - Enable more effective streaming on both edges and vertices
- A Streaming-Apply model
  - Combines the gather-apply-scatter operations
- A programming abstraction
  - To enable the streaming-apply model

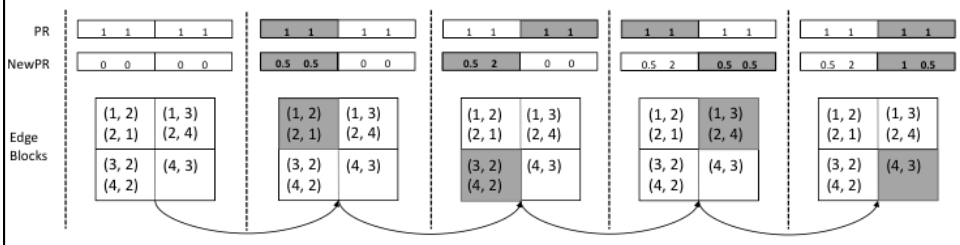
The Grid Representation

- 2-level hierarchical partitioning
  - 1st dimension
    - partition the vertices into chunks(1 2)(3,4)
    - partition the edges into shards by source vertex((1,2),(2,1),(1,3),(2,4))
  - 2nd dimension
    - partition the shards into blocks by destination vertex



(1, 2) (2, 1)	(1, 3) (2, 4)
(3, 2) (4, 2)	(4, 3)

The Streaming-Apply Model



Dual Sliding Windows:

- 1 read window + 1 write window
- Contents of active windows can be fit into RAM

Each iteration, just read edge list once, has opportunity to skip some blocks

Algorithm 1 Edge Block Streaming

```
for j ← 1, P do
    for i ← 1, P do
        if ChunkIsActive(i) then
            StreamEdgeBlock(i, j);
        end if
    end for
end for
```

Streaming-Apply Processing Model

- Vertex accesses are aggregated
  - Good locality
    - Only 2 partitions of vertices are accessed within each edge block
- On-the-fly updates onto vertices
  - Reduces I/O
  - Enables asynchronous implementation of algorithms (like WCC, etc.) which converges faster

Evaluation

- Datasets
  - Real world datasets
    - Social graph: LiveJournal, Twitter
    - Web graph: UK, Yahoo

Dataset	|V|	|E|	Data size
LiveJournal	4.85 million	69.0 million	527MB
Twitter	61.6 million	1.47 billion	11GB
UK	106 million	3.74 billion	28GB
Yahoo	1.41 billion	6.64 billion	50GB

- Benchmarks
  - BFS, WCC, SpMV, PageRank

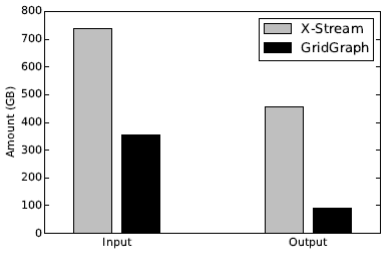
Performance vs. GraphChi, X-Stream

- Test environment
  - 1 AWS i2.xlarge instance
    - 4 (hypert.) vCPU cores
    - 30.5 GB memory
      - Memory limited to 8 GB
    - 800GB SSD

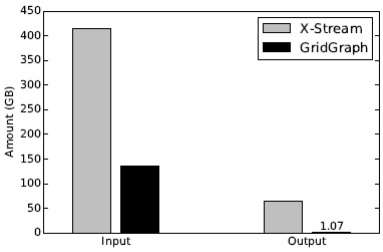
Runtime

	BFS	WCC	SpMV	PageR.
LiveJournal				
GraphChi	22.05	17.28	10.12	52.08
X-Stream	6.54	14.65	6.63	18.22
GridGraph	2.11	2.53	1.96	10.54
Twitter				
GraphChi	411.3	439.6	254.0	1225
X-Stream	435.9	1199	143.9	1779
GridGraph	51.34	190.3	43.78	461.4
UK				
GraphChi	3776	2527	407.2	3307
X-Stream	8081	12057	383.7	4374
GridGraph	979.0	1264	106.3	1285
Yahoo				
GraphChi	-	-	1540	13416
X-Stream	-	-	1076	9957
GridGraph	11935	3694	379.0	3923

I/O Amount vs. X-Stream



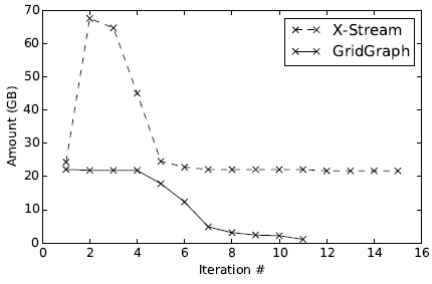
(a) PageRank on Yahoo graph with 5 iterations



(b) WCC on Twitter graph

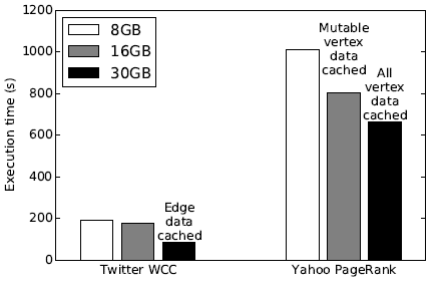
- Effects of Streaming-Apply and selective scheduling
- Less I/O (especially write) not only good for performance, but also good for life time of SSDs

I/O Amount vs. X-Stream



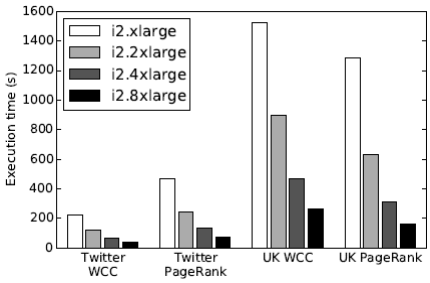
Faster convergence due to “asynchronous” label propagation
WCC on Twitter Graph

Scalability



← Scalability with available memory size

Scalability with I/O throughput →

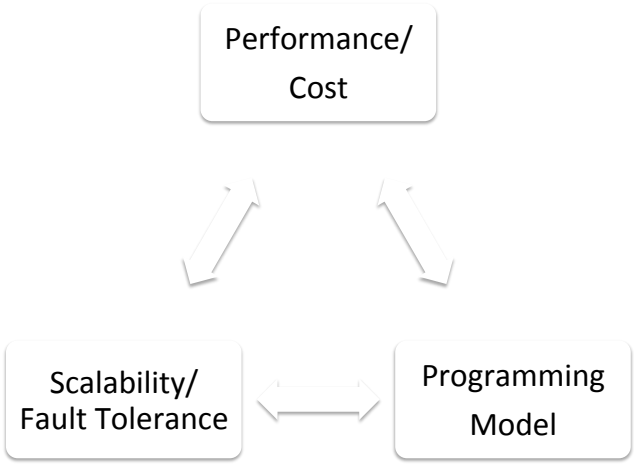


Performance vs. PowerGraph, GraphX

- Test environment
  - PowerGraph, GraphX
    - 16 AWS m2.4xlarge instances
  - GridGraph
    - 1 AWS i2.4xlarge instance(4 SSDs)

System	Twitter WCC	Twitter PR	UK WCC	UK PR	Cost per hr.
PowerGraph	244	249	714	833	15.68
GraphX	251	419	647	462	15.68
GridGrpah	64	132	471	314	3.41

Discussion on Big Data Systems

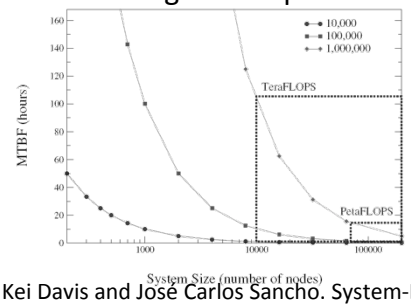


Fault-tolerance or Performance

- MapReduce, Spark
  - Use immutable data objects to help fault-tolerance
  - Poor performance, especially for BFS-like algorithms
- GraphLab,MPI, Pregel
  - Use mutable data objects to favor performance
  - Poor fault-tolerance features

The importance of performance

- Why fault-tolerance is important?
  - Many nodes, long execution time
- With significantly improvement on performance
  - A few nodes(maybe 1), shorter execution time
  - Fault tolerance is no longer so important



* Fabrizio Petrini and Kei Davis and José Carlos Sancho. System-Level Fault-Tolerance in Large-Scale Parallel Machines with Buffered Coscheduling. FTPDS04, 2004.

Many big data problems has its upper bound

- Number of human beings
  - 10G, the size of the social network is around 10TB
- Number of products
  - ~1M
- Moore's law is driving the compute power, memory size and I/O bandwidth keep improving
  - Today 36 core, 2TB memory and 8 SSD is quite accessible
- Today's big data problem with bounded size will be tomorrow's small data problem

GridGraph

- Performance as the first design choice instead of fault-tolerance
- 10 times more cost-effective than current distributed graph systems
- A good base for future extension
  - Compression to reduce I/O to further improve performance
  - More efficient processing for BFS-like algorithms
  - Port to systems with NVM
  - Distributed systems
- Plan to Open source
  - Welcome collaborations!

Q & A

THANKS