Tackling Large Graphs with Secondary Storage

Amitabha Roy
EPFL
Graphs

Social networks

Document networks

Biological networks

Humans, phones, bank accounts
Graph are Difficult

- Graph mining is challenging problem
- Traversal leads to data-dependent accesses
  - Little predictability
- Hard to parallelize efficiently
Tackling Large Graphs

• Normal approach
• Throw resources at the problem
• What does it take to process a trillion edges?
Big Iron

HPC/Graph500 benchmarks (June 2014)

<table>
<thead>
<tr>
<th>Graph Edges</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 trillion</td>
<td>Tsubame</td>
</tr>
<tr>
<td>1 trillion</td>
<td>Cray</td>
</tr>
<tr>
<td>1 trillion</td>
<td>Blue Gene</td>
</tr>
<tr>
<td>1 trillion</td>
<td>NEC</td>
</tr>
</tbody>
</table>
Large Clusters

A billion edges isn’t cool. You know what’s cool? A TRILLION edges.

Yes, using 3940 machines

Avery Ching, Facebook @Strata, 2/13/2014
Big Data

- Data is growing exponentially
  - 40 Zettabytes by 2020
- Unlikely you can put it all in DRAM
  - Need PM, SSD, Magnetic disks
  - Secondary storage != DRAM
- Also applicable to graphs
Motivation

If I can store the graph then why can’t I process it?

- 32 machines x 2TB magnetic disk = 64 TB storage
- 1 trillion edges x 16 bytes per edge = 16 TB storage
Problem #1

• Irregular access patterns
Problem #1

- Random access penalties

2ms seeks on a graph with a trillion edges ~ 1 year!
Problem #2

- Partitioning graphs across machines is hard
- Random partitions very poor for real-world graphs

Twitter graph: 20X difference with 32 machines!
Outline

- X-Stream (address problem #1)
- SlipStream (address problem #2)
X-Stream

• Single machine graph processing system [SOSP’13]

• Turns graph processing into sequential access
  • Change computation model
  • Partitioning of graph
Scatter-Gather

Existing computational model
Scatter-Gather

Activate vertex
Scatter-Gather

Scatter Updates
Scatter-Gather

Gather Updates
Storage

Edges

1 → 5
1 → 6

Vertices

1 2 3 4 5 6

1 2 3 4 5 6
Edge File

Edges:
1 → 5
1 → 6
6 → 2
6 → 4

Vertices:
1
2
3
4
5
6
Edge File

1 → 5
1 → 6
6 → 2
6 → 4
Edge-centric Scatter-Gather

Scan entire edge list

SCAN

1 → 5
1 → 6

6 → 2
6 → 4
Edge-centric Scatter-Gather

Use only necessary edges

SCAN

1 → 5
1 → 6
6 → 2
6 → 4
Tradeoff

✔ Achieve sequential bandwidth

✖ Need to scan entire edge list

Winning Tradeoff !
Winning Tradeoff

- Real-world graphs have small diameter
- Traversals in just a few iterations of scatter-gather
- Large number of active vertices in most iterations
Benefit

Order oblivious

SCAN

1 → 5
6 → 4
1 → 6
6 → 2
What about the vertices?
What about the vertices?

Seeking in RAM is free!
How can we fit vertices in RAM?

SCAN
1 → 5
1 → 6
6 → 2
6 → 4

SEEK
1
2
3
4
5
6
Streaming Partitions

Fits in RAM

1 → 5
1 → 6
2 → 3
3 → 5
6 → 2
6 → 4
Streaming Partitions

SCAN

1 → 5
1 → 6
2 → 3

3 → 5

6 → 2
6 → 4

Load in RAM

1
2

3
4

5
6
Producing Partitions

• No requirement on quality (# of cross edges)
  • Need only fit into RAM
  • Random partitions are great
• Random partitions work great
Algorithms Supported

- Supports traversal algorithms
  - BFS, WCC, MIS, SCC, K-Cores, SSSP, BC
- Supports algebraic operations on the graph
  - BP, ALS, SpMV, Pagerank
- Good testbed for newer streaming algorithms
  - HyperANF, Semi-streaming Triangle Counting
Competition

• Graphchi
  • Another on-disk graph processing system (OSDI’12)
  • Special on-disk data structure: shards
  • Makes accesses look sequential
  • Producing shards requires sorting edges
More Competition

• Applies to any two level memory
• Includes CPU cache and DRAM
• Main memory graph processing ?
• Looked at Ligra (PPoPP 2012)
BFS

Time (seconds)

Ligra

X-Stream

CPUs

100.0

10.0

1.0

0.1

1 2 4 8 16

CPUs

1.0 10.0 100.0

Time (seconds)
Where we stand

10 billion

Powergraph
OSDI’12

100 billion

Pregel
SIGMOD’10
300 machines

1 trillion

Ligra
PPoPP’12

X-Stream
SOSP’13
1 machine

How do we get further? Scale out
SlipStream

• Aggregate bandwidth and storage of a cluster

• Solves the graph partitioning problem
  • Rethinking storage access
  • Rethinking streaming partition execution

• We know how to do it right for one machine
Scaling Out

- Assign different streaming partitions to machines

Graph partitioning is hard to get right
Load Imbalance

Red

Blue
Load Imbalance

Red

Blue

IDLE
IDLE
Flat Storage

Stripe data across all disks
Allow any machine to access any disk

Red

✔ Balance Capacity
✔ Balance BW

Blue

SP
SP

SP
SP
Flat Storage

Stripe data across all disks
Allow any machine to access any disk

Flat Storage Box
Flat Storage

- Assumes full bisection bandwidth network
- Can be done at data-center scales
- Nightingale et. al. OSDI 2012 using CLOS switches
- Already true at rack scale
  - Like in our cluster
Flat Storage

Red

Blue

Flat Storage Box

SP

SP

SP

SP
Flat Storage

Using only half the available bandwidth

Flat Storage Box

Red

IDLE

IDLE
Extracting Parallelism

- Edge-centric loop
  - Stream in edges/updates
  - Access vertices
- What if…
- Independent copies of vertices on machines
Extracting Parallelism

Scan → Scatter/Gather → Vertices
Scatter Step

Scan Edges

Scatter

Vertices
Scatter Step

Scan Edges

Flat Storage Box

Scatter

Vertices

machine 1

Scatter

Vertices

machine 2
Gather Step

Scan Updates

Flat Storage Box

Gather

Vertices

machine 1

Gather

Vertices

machine 2
Merge Step

Application of updates is commutative

machine 1

machine 2

No need to go to disk
X-Stream to SlipStream

SlipStream graph algorithms

= 

X-Stream graph algorithms

+ 

Merge function

• Easy to write merge function (looks like gather)
Putting it Together

Red

Flat Storage Box

SP
SP
Putting it Together

Red → Copy

Flat Storage Box:

SP  SP
Putting it Together

✅ Back to Full Bandwidth

Red

Flat Storage Box

Red

SP

SP
Automatic Load Balancing

Compute Box

Flat Storage Box
Recap

- Graph Partitioning across machines is hard
  - Drop locality using flat storage
    - Make it one disk
  - Same streaming partition on multiple nodes
    - Extract full bandwidth from the aggregated disk
- Systems approach to solving algorithms problem
Flat Storage

- Distributed Storage layer for SlipStream
- Looked at other designs
  - FDS (OSDI 2012)
  - GFS (SOSP 2003)
  - ...
- Implementing distributed storage is hard 😞
The Hard Bit

Store Block X
The Hard Bit

Where is block X?

Need a location service
f: file, block → machine, offset
Block Location

Store block of updates
Block Location is Irrelevant

Give me any block of updates

Streaming is order oblivious!
Random Schedule

- Centralized metadata service $\Rightarrow$ randomization
- Connect to a random machine for load/store
- Extremely simple implementation
Downside?

- Can lead to collisions
- Collisions reduce utilization
No Downside

- Utilization lower bound at \((1 - \frac{1}{e})\) \(\sim 62\%\)
Recap

• Building distributed storage is hard

• Algorithms approach to solving systems problem
  • Streaming algorithms are order oblivious
  • Randomized schedule
Evaluation Results

- 32 cores
- 32 GB RAM
- 200 GB SSD
- 2 TB 5200 RPM

Rack: 10 GigE full bisection
Scalability

- Solve larger problems using more machines
- Used synthetic scale-free graphs
  - Double problem size (vertices and edges)
  - Double machine count
- Till 32 machines, 4 billion vertices, 64 billion edges
Scaling RMAT (SSD)

32X problem size at 2.7X cost
Scaling RMAT (SSD)

32X problem size at 2.7X cost

Engineering
Loss of sequentaility
Collisions

Normalized Wall Time

Machines

PR  BFS  SCC  WCC  BP  MCST  Cond.  MIS  SPMV  SSSP
Capacity

• Largest graph we can fit in our cluster
  • 32 billion vertices, 1 trillion edges
  • Magnetic disks
  • BFS
• Projected seeks were 1 year
## Terascale

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Time</td>
<td>2d 9h</td>
</tr>
<tr>
<td>MTEPS</td>
<td>5</td>
</tr>
<tr>
<td>I/O</td>
<td>282 TB</td>
</tr>
<tr>
<td>BW</td>
<td>1.53 GB/s</td>
</tr>
</tbody>
</table>

Don’t need supercomputers or very large clusters
```
<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Time</td>
<td>2d 9h</td>
</tr>
<tr>
<td>MTEPS</td>
<td>5</td>
</tr>
<tr>
<td>I/O</td>
<td>282 TB</td>
</tr>
<tr>
<td>BW</td>
<td>1.53 GB/s</td>
</tr>
</tbody>
</table>

Direct results from unordered edge list
```
## SlipStream vs. Competition

<table>
<thead>
<tr>
<th>System</th>
<th>RAM</th>
<th>Pre-process</th>
<th>Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>Powergraph</td>
<td>128 GB</td>
<td>1271s</td>
<td>103s</td>
</tr>
<tr>
<td>SlipStream</td>
<td>32 GB</td>
<td>X</td>
<td>1854s</td>
</tr>
</tbody>
</table>

WCC/RMAT/128M vertices 2B edges/2 machines

Preprocessing your data for locality can take a lot of time!
Where we stand

- Powergraph (OSDI'12)
- Pregel (SIGMOD'10, 300 machines)
- Ligra (PPoPP'12)

10 billion

100 billion

1 trillion

- X-Stream (SOSP'13, 1 machine)
- SlipStream (32 machines)

How do we get further? Buy more disks :)
Conclusion

• Process large graphs using secondary storage
  • Match algorithm to systems: streaming
  • Match system to algorithms: order obliviousness
• If you can store it, you can process it