Big Graph Processing: Some Background

Bo Wu
Colorado School of Mines

Part of slides from: Paul Burkhardt (National Security Agency) and Carlos Guestrin (Washington University)
Graphs are everywhere!

- A graph is a collection of binary relationships, i.e. networks of pairwise interactions including social networks, digital networks…

part of Internet

brain network
Nearly 300 years ago the first graph problem consisted of 4 vertices and 7 edges — Seven Bridges of Konigsberg problem.

Is it possible to cross each of the seven bridges exactly once?

Not too hard to fit in memory.
Scale of real-world graphs

○ Graph scale in current CS literature
  – on order of billions of edges, tens of gigabytes

<table>
<thead>
<tr>
<th>Dataset</th>
<th>n (vertices in millions)</th>
<th>m (edges in millions)</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS-Skitter (AS-Skitter)</td>
<td>1.7</td>
<td>11</td>
<td>142 MB</td>
</tr>
<tr>
<td>LiveJournal (LJ)</td>
<td>4.8</td>
<td>69</td>
<td>337.2 MB</td>
</tr>
<tr>
<td>U.S. Road Network (USRD)</td>
<td>24</td>
<td>58</td>
<td>586.7 MB</td>
</tr>
<tr>
<td>Billion Triple Challenge (BTC)</td>
<td>165</td>
<td>773</td>
<td>5.3 GB</td>
</tr>
<tr>
<td>WWW of UK (WebUK)</td>
<td>106</td>
<td>1877</td>
<td>8.6 GB</td>
</tr>
<tr>
<td>Twitter (Twitter)</td>
<td>42</td>
<td>1470</td>
<td>24 GB</td>
</tr>
<tr>
<td>Yahoo! Web Graph (YahooWeb)</td>
<td>1413</td>
<td>6636</td>
<td>120 GB</td>
</tr>
</tbody>
</table>
Big Data begets Big Graphs

- Increasing volume, velocity, variety of Big Data are significant challenges to scalable algorithms
- How will graph applications adapt to Big Data at petabyte scale?
- Ability to store and process Big Graphs impacts typical data structures

<table>
<thead>
<tr>
<th>Orders of magnitude</th>
<th>Undirected graph data structure space complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>kilobyte (KB) = $2^{10}$</td>
<td>$\Theta(n^2)$  adjacency matrix</td>
</tr>
<tr>
<td>megabyte (MB) = $2^{20}$</td>
<td>$\Theta(n + 4m)$ adjacency list</td>
</tr>
<tr>
<td>gigabyte (GB) = $2^{30}$</td>
<td>$\Theta(4m)$     edge list</td>
</tr>
</tbody>
</table>
Social Scale

- 1 billion vertices, 100 billion edges
  - 111 PB adjacency matrix
  - 2.92 TB adjacency list
  - 2.92 TB edge list
Web scale…

- 50 billion vertices, 1 trillion edges
  - 271 EB adjacency matrix
  - 29.5 TB adjacency list
  - 29.1 TB edge list

Internet graph from the Opte Project (http://www.opte.org/maps)
Brain scale…

- 100 billion vertices, 100 trillion edge
  - 2.84 PB adjacency list
  - 2.84 PB edge list

Human connectome.
Gerhard et al., Frontiers in Neuroinformatics 5(3), 2011
Benchmarking scalability on Big Graphs

- Big Graph challenge our conventional thinking on both algorithms and computer architecture

- New Graph500.org benchmark provides a foundation for conducting experiments on graph datasets

**Graph500 benchmark**

Problem classes from 17 GB to 1 PB — many times larger than common datasets in literature.
Graph algorithms are challenging

- Difficult to parallelize
  - irregular data accesses increase latency
  - skewed data distribution creates bottlenecks
    - Celebrity nodes in social networks
- Increased size imposes greater...
  - storage overhead
  - IO burden

Algorithm complexity really matters!
Run-time of $O(n^2)$ on a trillion node graph is not practical!
Problem: How do we store and process Big Graphs?

- Conventional approach is to store and compute in-memory
- Shared memory
  - Parallel Random Access Machine (PRAM)
  - data in globally-shared memory
  - implicit communication by updating memory
  - fast-random access
- Distributed memory
  - Bulk Synchronous Parallel (BSP)
  - data distributed to local, private memory
  - explicit communication by sending messages
  - easier to scale by adding more machines
Memory is fast but…

- Algorithms must exploit computer memory hierarchy
  - designed for spatial and temporal locality
  - registers, L1,L2,L3 cache, TLB, pages, disk. . .
  - great for unit-stride access common in many scientific codes, e.g. linear algebra

- But common graph algorithm implementations have. . .
  - lots of random access to memory causing. . .
  - many cache and TLB misses
Question: What is the memory throughput if 90% TLB hit and 0.01% page fault on miss?

\[
T_n = p_n l_n + (1 - p_n) T_{n-1}
\]

**Example**

TLB = 20ns, RAM = 100ns, DISK = 10ms (10 × 10^6ns)

\[
T_2 = p_2 l_2 + (1 - p_2)(p_1 l_1 + (1 - p_1)T_0)
\]

\[
= .9(TLB+RAM) + .1(.9999(TLB+2RAM) + .0001(DISK))
\]

\[
= .9(120ns) + .1(.9999(220ns) + 1000ns) = 230ns
\]

**ANSWER:** 33 MB/s
Graph problems that fit in memory can leverage excellent advances in architecture and libraries. 

- Cray XMT2 designed for latency-hiding
- SGI UV2 designed for large, cache-coherent shared-memory
- Body of literature and libraries
  - Parallel Boost Graph Library (PBGL) — Indiana University
  - Multithreaded Graph Library (MTGL) — Sandia National Labs
  - GraphCT/STINGER — Georgia Tech
  - GraphLab — Carnegie Mellon University
  - Giraph — Apache Software Foundation

But some graphs do not fit in memory...
We can add more memory, but …

- Memory capacity is limited by. . .
  - number of CPU pins, memory controller channels, DIMMs per channel
  - memory bus width

- Globally-shared memory limited by. . .
  - CPU address space
  - cache-coherence
Larger systems, greater latency …

- Increasing memory can increase latency
  - traverse more memory addresses
  - larger system with greater physical distance between machines

Fundamental limitation: speed of light

- Latency causes significant inefficiency in new CPU architectures

IBM PowerPC A2
A single 1.6 GHz PowerPC A2 can perform 204.8 operations per nanosecond!
Easier to increase capacity using disks

- Current Intel Xeon E5 architectures:
  - 384 GB max. per CPU (4 channels x 3 DIMMS x 32 GB)
  - 64 TB max. globally-shared memory (46-bit address space)
  - 3881 dual Xeon E5 motherboards to store Brain Graph — 98 racks

- Disk capacity not unlimited but higher than memory
  - Larget disk on market: 8TB
  - needs 364 to store Brain Graph which can fit in 5 racks

- Disk is not enough—applications will still require memory for processing
Big Graph Processing Frameworks
Why not just map reduce?

- Developed by Google
- Excellent for embarrassingly massively parallel computations
  - No communication needed
  - Many machine learning algorithms fall into this category
- Not efficient for iterative algorithms that have dependences
  - Unnecessary IO traffic
What’s the natural way to program graph computation?

“Think like a Vertex.”

-Malewicz et al. [SIGMOD’10]
Most famous parallel graph processing framework

The **GraphLab Goals**

- Know how to solve ML problem on 1 machine
- Efficient parallel predictions
GASP

Gather (Reduce)
Accumulate information about neighborhood

User Defined:
- Gather(y) → Σ
- Σ₁ + Σ₂ → Σ₃

Apply
Apply the accumulated value to center vertex

User Defined:
- Apply(y, Σ) → y'̂

Scatter
Update adjacent edges and vertices.

User Defined:
- Scatter(y) → y'̂

Update Edge Data & Activate Neighbors
We still need parallelism
Graph partition: not easy at all at scale

Regular Mesh

Natural Graph
Power-law distribution

More than $10^8$ vertices have one neighbor.

High-Degree Vertices

Top 1% of vertices are adjacent to 50% of the edges!
Power-law degree distribution

“Star Like” Motif

President Obama

Followers
Random graph partitioning

- Graph parallel abstractions rely on partitioning:
  - Minimize communication
  - Balance computation and storage

Diagram:
- 10 Machines → 90% of edges cut
- 100 Machines → 99% of edges cut!
Challenges of high-degree vertices

Machine 1

Data transmitted across network $O(\text{# cut edges})$

Machine 2
Idea of vertex cut
GAS decomposition

Gather

Apply

Scatter
Random Edge-Placement

• Randomly assign edges to machines

Balanced Vertex-Cut

Y Spans 3 Machines
Z Spans 2 Machines
Not cut!
Greedy Vertex-Cuts

- Place edges on machines which already have the vertices in that edge.
What’s the popularity of this user?
PargeRank Algorithm

\[ R[i] = 0.15 + \sum_{j \in \text{Nbrs}(i)} w_{ji} R[j] \]

- Update ranks in parallel
- Iterate until convergence
PageRank in Graphlab

GraphLab_PageRank(i)

// Compute sum over neighbors
total = 0
foreach (j in in_neighbors(i)):
  total = total + R[j] * w_{ji}

// Update the PageRank
R[i] = 0.1 + total

// Trigger neighbors to run again
if R[i] not converged then
  foreach (j in out_neighbors(i))
    signal vertex-program on j
Triangle counting on Twitter

40M Users, 1.4 Billion Links

Counted: 34.8 Billion Triangles

Hadoop
1536 Machines
423 Minutes

GraphLab
64 Machines
15 Seconds

1000 x Faster
What if I don’t have a cluster?

GraphChi: Going small with GraphLab

Solve huge problems on small or embedded devices?
GraphChi — disk-based GraphLab

- Challenge
  - Random disk accesses

- Naive solutions
  - Graph clustering
  - Prefetching

- Solution
  - Novel graph representation in disk
  - Parallel sliding window
  - Minimizes random accesses
Parallel sliding window layout

A shard is easy to fit in memory
Parallel sliding window execution

Load subgraph for vertices 1..100

O(P^2) random accesses per pass on entire graph
Triangle counting on Twitter graph

![Diagram showing triangle counting results]

- **40M Users**
- **1.2B Edges**
- **Total: 34.8 Billion Triangles**

- **Hadoop**
  - 1636 Machines
  - 423 Minutes

- **GraphChi**
  - 59 Minutes, 1 Mac Mini!

- **GraphLab2**
  - 64 Machines, 1024 Cores
  - 1.5 Minutes