An NSA Big Graph experiment

Paul Burkhardt, Chris Waring

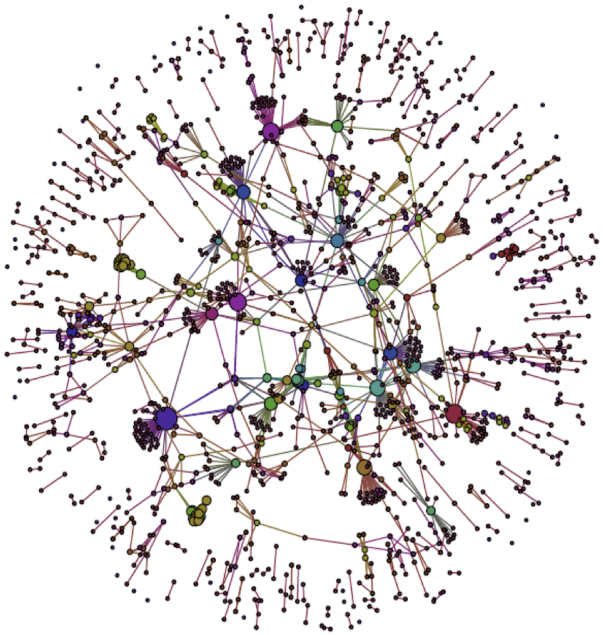
U.S. National Security Agency
Research Directorate - R6
Technical Report NSA-RD-2013-056002v1

May 20, 2013

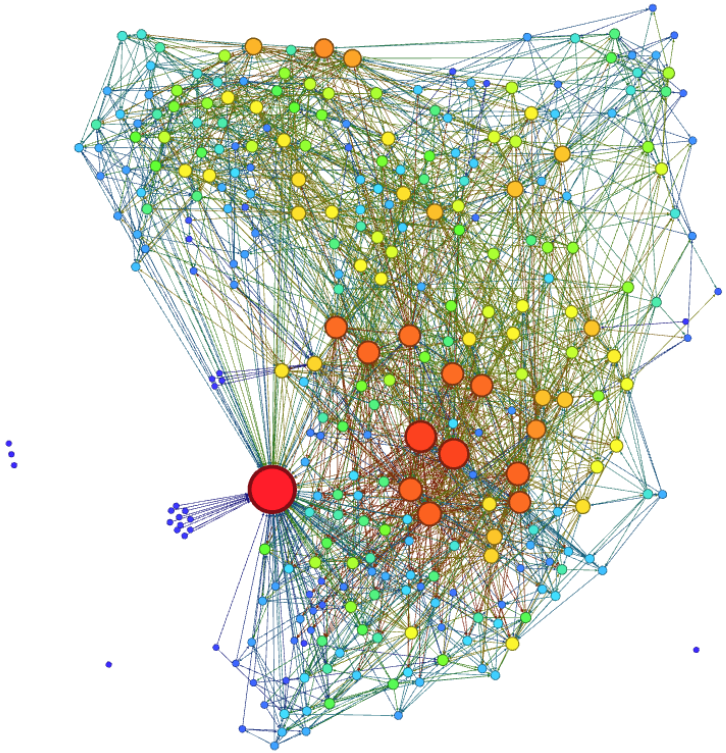
Graphs are everywhere!

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

- road networks
- utility grids
- internet
- protein interactomes



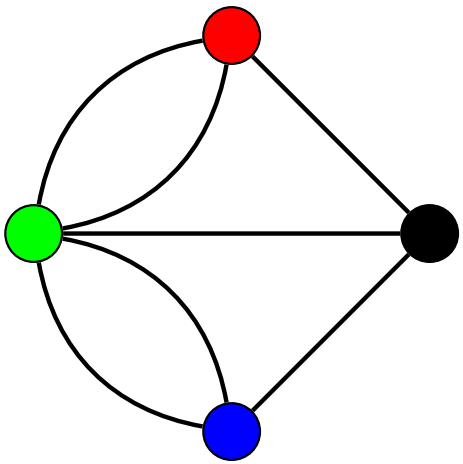
M. tuberculosis
Vashisht, *PLoS ONE* **7**(7), 2012



Brain network of C. elegans
Watts, Strogatz, *Nature* **393**(6684), 1998

Scale of the first graph

Nearly 300 years ago the first graph problem consisted of 4 vertices and 7 edges—*Seven Bridges of Königsberg* problem.



Crossing the River Pregel

Is it possible to cross each of the Seven Bridges of Königsberg exactly once?

Not too hard to fit in "memory".

Scale of real-world graphs

Graph scale in current Computer Science literature

On order of billions of edges, tens of gigabytes.

AS by Skitter (AS-Skitter) — internet topology in 2005 (n = router, m = traceroute)

LiveJournal (LJ) — social network (n = members, m = friendship)

U.S. Road Network (USRD) — road network (n = intersections, m = roads)

Billion Triple Challenge (BTC) — RDF dataset 2009 (n = object, m = relationship)

WWW of UK (WebUK) — Yahoo Web spam dataset (n = pages, m = hyperlinks)

Twitter graph (Twitter) — Twitter network[1] (n = users, m = tweets)

Yahoo! Web Graph (YahooWeb) — WWW pages in 2002 (n = pages, m = hyperlinks)

Popular graph datasets in current literature

	n (vertices in millions)	m (edges in millions)	size
AS-Skitter	1.7	11	142 MB
LJ	4.8	69	337.2 MB
USRD	24	58	586.7 MB
BTC	165	773	5.3 GB
WebUK	106	1877	8.6 GB
Twitter	42	1470	24 GB
YahooWeb	1413	6636	120 GB

[1] http://an.kaist.ac.kr/traces/WWW2010.html

Big Data begets *Big Graphs*

Increasing *volume*,*velocity*,*variety* of *Big Data* are significant challenges to scalable algorithms.

Big Data Graphs

How will graph applications adapt to *Big Data* at petabyte scale?

Ability to store and process *Big Graphs* impacts typical data structures.

Orders of magnitude

kilobyte (KB) $= 2^{10}$	terabyte (TB) $= 2^{40}$
megabyte (MB) $= 2^{20}$	petabyte (PB) $= 2^{50}$
gigabyte (GB) $= 2^{30}$	exabyte (EB) $= 2^{60}$

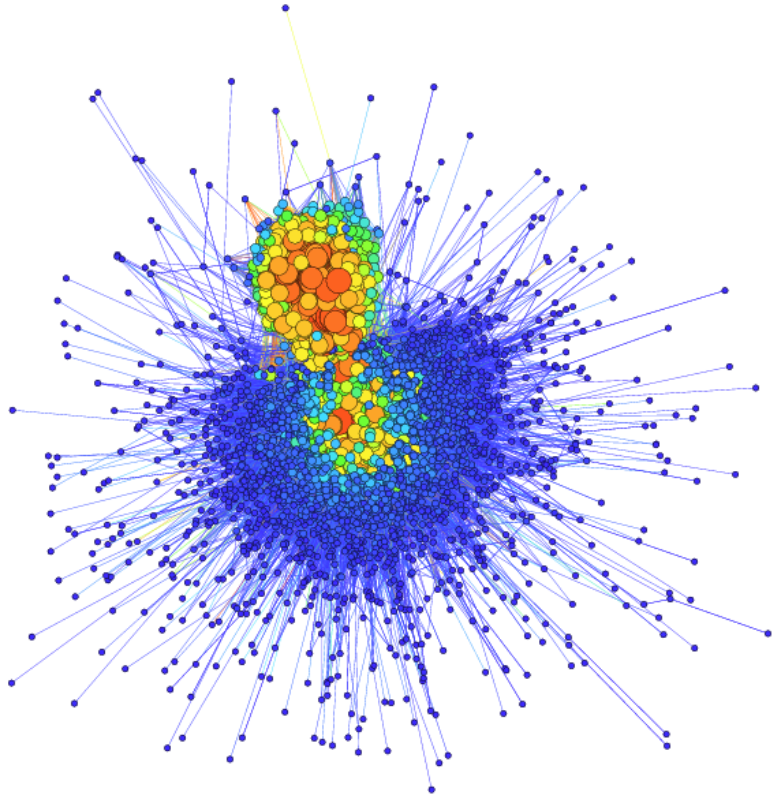
Undirected graph data structure space complexity

$$\Theta(\textit{bytes}) \times \begin{cases} \Theta(n^2) & \text{adjacency matrix} \\ \Theta(n+4m) & \text{adjacency list} \\ \Theta(4m) & \text{edge list} \end{cases}$$

Social scale. . .

1 billion vertices, 100 billion edges

- 111 PB adjacency matrix
- 2.92 TB adjacency list
- 2.92 TB edge list

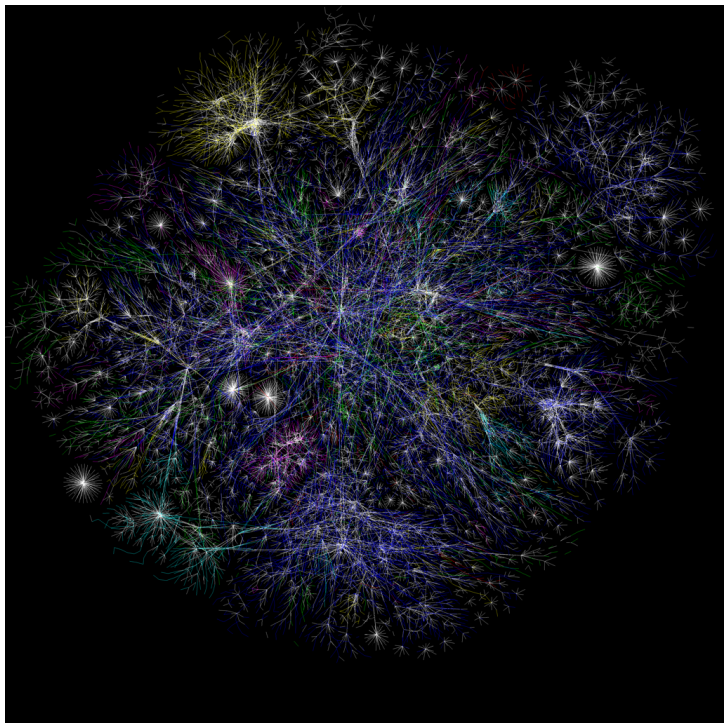


Twitter graph from Gephi dataset
(http://www.gephi.org)

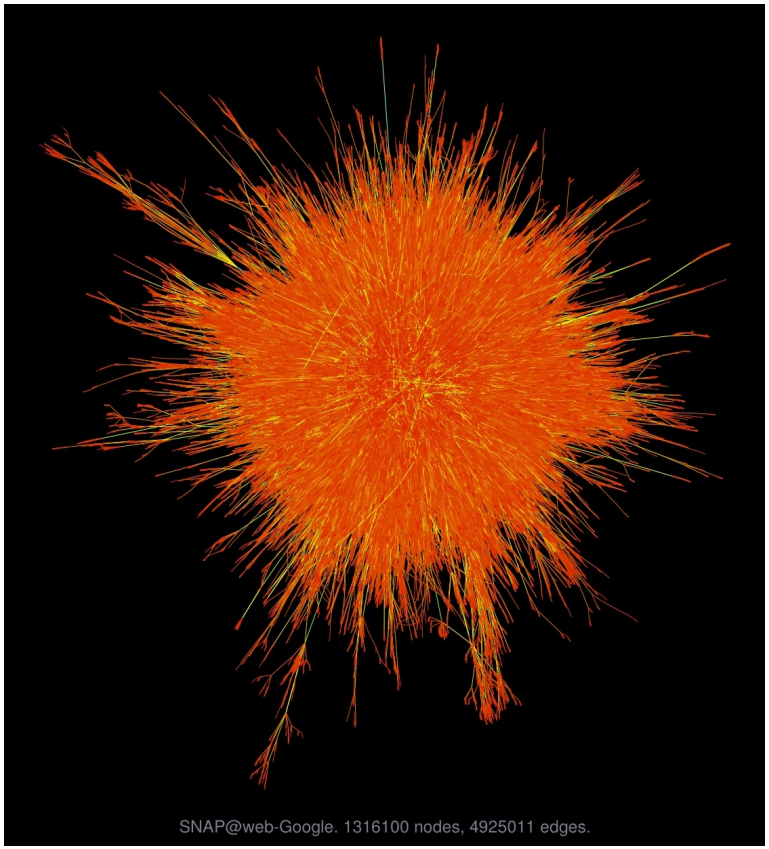
Web scale. . .

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)

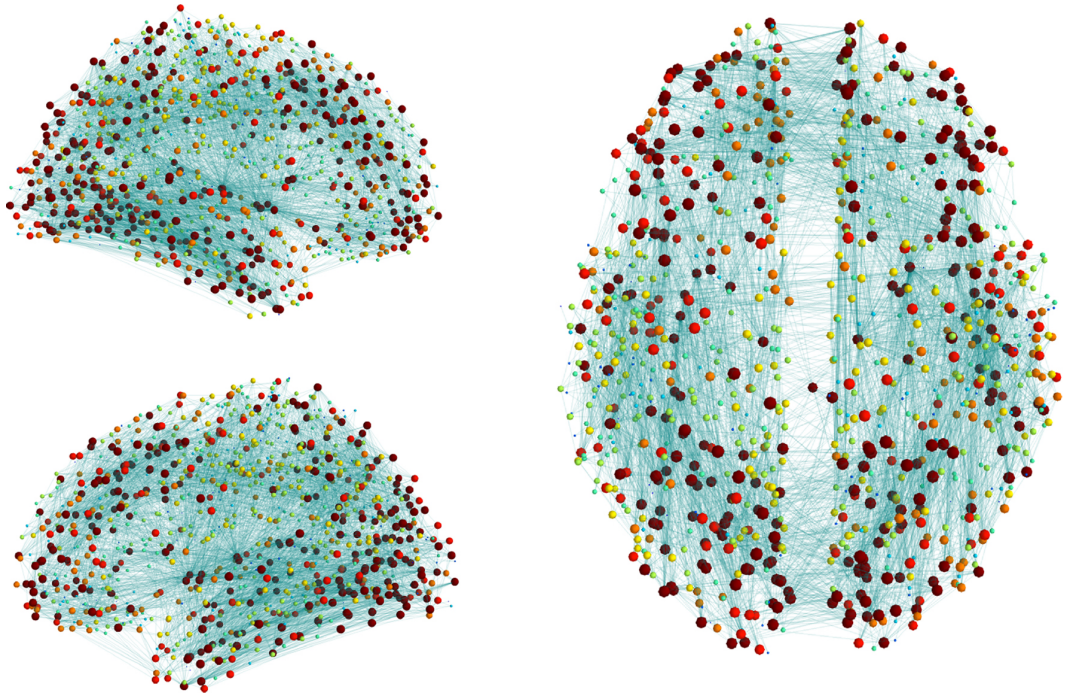


Web graph from the SNAP database
(http://snap.stanford.edu/data)

Brain scale. . .

100 billion vertices, 100 trillion edges

- 2.08 $mN_A \cdot bytes^2$ (molar bytes) adjacency matrix
- 2.84 PB adjacency list
- 2.84 PB edge list



Human connectome.
Gerhard et al., *Frontiers in Neuroinformatics* **5**(3), 2011

[2] $N_A = 6.022 \times 10^{23} mol^{-1}$

Benchmarking scalability on *Big Graphs*

Big Graphs 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 access increases latency
- skewed data distribution creates bottlenecks
  - giant component
  - high degree vertices

Increased size imposes greater...

- latency
- resource contention (i.e. hot-spotting)

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

Poor locality increases latency...

QUESTION: What is the memory throughput if 90% TLB hit and 0.01% page fault on miss?

Effective memory access time

$$T_n = p_n l_n + (1 - p_n) T_{n-1}$$

Example

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

$$\begin{aligned} T_2 &= p_2 l_2 + (1 - p_2)(p_1 l_1 + (1 - p_1) T_0) \\ &= .9(\text{TLB+RAM}) + .1(.9999(\text{TLB+2RAM}) + .0001(\text{DISK})) \\ &= .9(120ns) + .1(.9999(220ns) + 1000ns) = 230ns \end{aligned}$$

ANSWER:

Poor locality increases latency...

QUESTION: What is the memory throughput if 90% TLB hit and 0.01% page fault on miss?

Effective memory access time

$$T_n = p_n l_n + (1 - p_n) T_{n-1}$$

Example

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

$$\begin{aligned} T_2 &= p_2 l_2 + (1 - p_2)(p_1 l_1 + (1 - p_1) T_0) \\ &= .9(\text{TLB+RAM}) + .1(.9999(\text{TLB+2RAM}) + .0001(\text{DISK})) \\ &= .9(120ns) + .1(.9999(220ns) + 1000ns) = 230ns \end{aligned}$$

ANSWER: 33 MB/s

If it fits. . .

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 — parallel traces of same length, one line per bit

Globally-shared memory limited by...

- CPU address space
- cache-coherency

Larger systems, greater latency...

Increasing memory can increase latency.

- traverse more memory addresses
- larger system with greater physical distance between machines

Light is only so fast (but still faster than neutrinos!)

It takes approximately 1 nanosecond for light to travel 0.30 meters

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.

Largest capacity HDD on market

4 TB HDD — need 728 to store Brain Graph which can fit in 5 racks (4 drives per chassis)

Disk is not enough—applications will still require memory for processing

Top supercomputer installations

Largest supercomputer installations do not have enough memory to process the Brain Graph (3 PB)!

- Titan Cray XK7 at ORNL — #1 Top500 in 2012
  - 0.5 million cores
  - 710 TB memory
  - 8.2 Megawatts[3]
  - 4300 sq.ft. (NBA basketball court is 4700 sq.ft.)
- Sequoia IBM Blue Gene/Q at LLNL — #1 Graph500 in 2012
  - 1.5 million cores
  - 1 PB memory
  - 7.9 Megawatts[3]
  - 3000 sq.ft.

Electrical power cost

At 10 cents per kilowatt-hour — $7 million per year to keep the lights on!

3

[3] Power specs from http://www.top500.org/list/2012/11

Cloud architectures can cope with graphs at *Big Data* scales

Cloud technologies designed for *Big Data* problems.

- scalable to massive sizes
- fault-tolerant; restarting algorithms on *Big Graphs* is expensive
- simpler programming model; MapReduce

Big Graphs from real data are not static. . .

Distributed $\langle key, value \rangle$ repositories

Much better for storing a distributed, dynamic graph than a distributed filesystem like HDFS.

- create index on edges for fast random-access and ...
- sort edges for efficient block sequential access
- updates can be fast but ...
- not transactional; eventually consistent

NSA BigTable — Apache Accumulo

Google BigTable paper inspired a small group of NSA researchers to develop an implementation with cell-level security.

Apache Accumulo

Open-sourced in 2011 under the Apache Software Foundation.

Using Apache Accumulo for *Big Graphs*

Accumulo records are flexible and can be used to store graphs with typed edges and weights.

- store graph as distributed edge list in an *Edge Table*
  - edges are natural $\langle key, value \rangle$ pairs, i.e. vertex pairs
- updates to edges happen in memory and are immediately available to queries and . . .
- updates are written to write-ahead logs for fault-tolerance

Apache Accumulo $\langle key, value \rangle$ record

KEY					VALUE
ROW ID	COLUMN			TIMESTAMP	
	FAMILY	QUALIFIER	VISIBILITY		

But for bulk-processing, use MapReduce

Big Graph processing suitable for MapReduce. . .

- large, bulk writes to HDFS are faster (no need to sort)
- temporary scratch space (local writes)
- greater control over parallelization of algorithms

Quick overview of MapReduce

MapReduce processes data as $\langle key, value \rangle$ pairs in three steps:

1. MAP
   - map tasks independently process blocks of data in parallel and output $\langle key, value \rangle$ pairs
   - map tasks execute user-defined "map" function
2. SHUFFLE
   - sorts $\langle key, value \rangle$ pairs and distributes to reduce tasks
   - ensures value set for a key is processed by one reduce task
   - framework executes a user-defined "partitioner" function
3. REDUCE
   - reduce tasks process assigned (key, value set) in parallel
   - reduce tasks execute user-defined "reduce" function

MapReduce is not great for iterative algorithms

- Iterative algorithms are implemented as a sequence of MapReduce jobs, i.e. rounds.
- But iteration in MapReduce is expensive...
  - temporary data written to disk
  - sort/shuffle may be unnecessary for each round
  - framework overhead for scheduling tasks

Good principles for MapReduce algorithms

Be prepared to *Think in MapReduce*...

- avoid iteration and minimize number of rounds
- limit child JVM memory
  - number of concurrent tasks is limited by per machine RAM
- set IO buffers carefully to avoid spills (requires memory!)
- pick a good partitioner
- write raw comparators
- leverage compound keys
  - minimizes hot-spots by distributing on key
  - secondary-sort on compound keys is almost free

Round-Memory tradeoff

Constant, O(1), in memory and rounds.

Best of both worlds – MapReduce and Accumulo

Store the *Big Graph* in an Accumulo Edge Table . . .

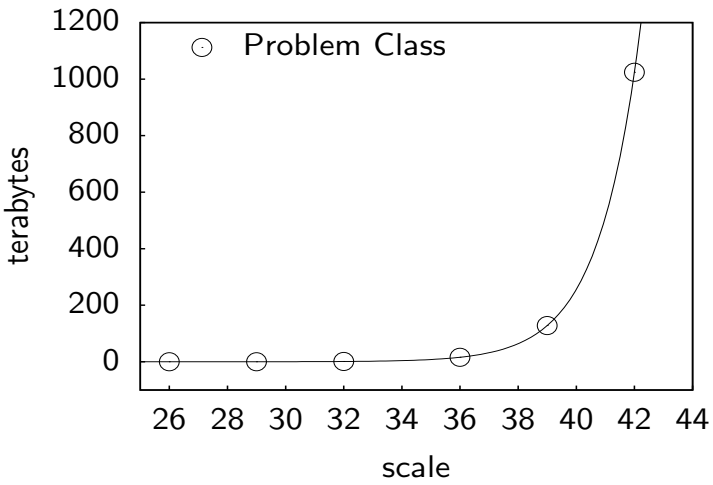
- great for updating a big, typed, multi-graph
- more timely (lower latency) access to graph

Then extract a sub-graph from the Edge Table and use MapReduce for graph analysis.

How well does this really work?

Prove on the industry benchmark Graph500.org!

- Breadth-First Search (BFS) on an undirected R-MAT Graph
- count *Traversed Edges per Second* (TEPS)
- $n = 2^{scale}$, $m = 16 \times n$



Graph500 Problem Sizes

Class	Scale	Storage
Toy	26	17 GB
Mini	29	140 GB
Small	32	1 TB
Medium	36	17 TB
Large	39	140 TB
Huge	42	1.1 PB

Walking a ridiculously *Big Graph*...

Everything is harder at scale!

- performance is dominated by ability to acquire adjacencies
- requires specialized MapReduce partitioner to load-balance queries from MapReduce to Accumulo Edge Table

Space-efficient Breadth-First Search is critical!

- create $\langle vertex, distance \rangle$ records
- sliding window over distance records to minimize input at each k iteration
- reduce tasks get adjacencies from Edge Table but only if...
- all distance values for a vertex, v, are equal to k

Graph500 Experiment

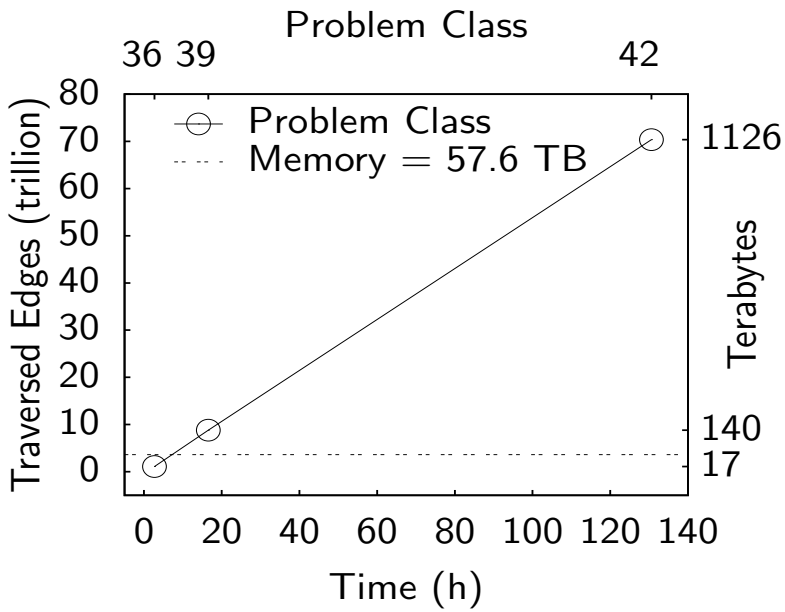
Graph500 **Huge** class — scale 42

2^{42} (4.40 trillion) vertices
2^{46} (70.4 trillion) edges
1 Petabyte

Cluster specs

1200 nodes
2 Intel quad-core per node
48 GB RAM per node
7200 RPM sata drives

- **Huge** problem is 19.5x more than cluster memory



Graph500 Scalability Benchmark

Graph500 Experiment

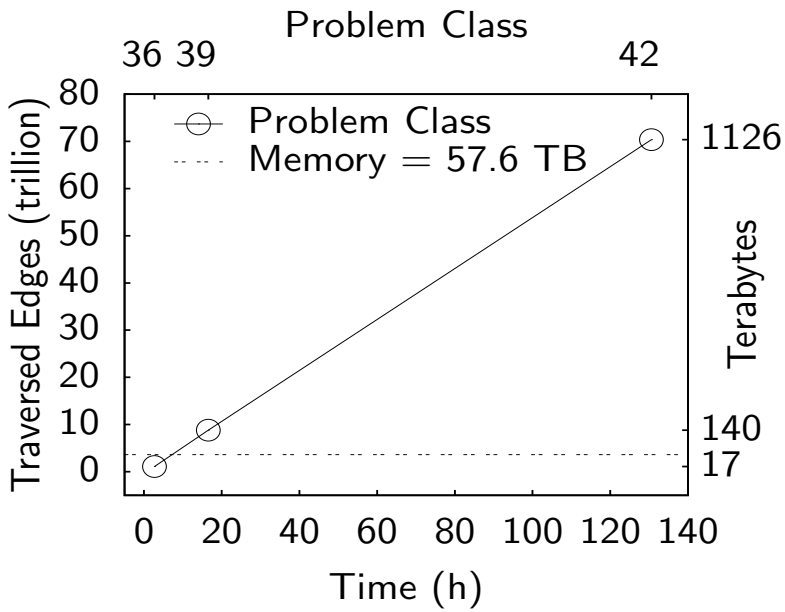
Graph500 **Huge** class — scale 42

2^{42} (4.40 trillion) vertices
2^{46} (70.4 trillion) edges
1 Petabyte

Cluster specs

1200 nodes
2 Intel quad-core per node
48 GB RAM per node
7200 RPM sata drives

- **Huge** problem is 19.5x more than cluster memory
- linear performance from 1 trillion to 70 trillion edges...



Graph500 Scalability Benchmark

Graph500 Experiment

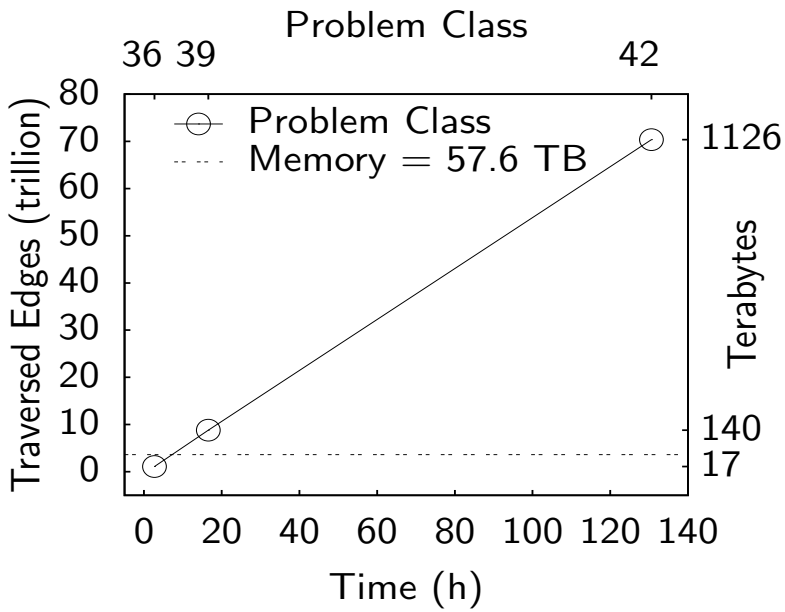
Graph500 **Huge** class — scale 42

2^{42} (4.40 trillion) vertices
2^{46} (70.4 trillion) edges
1 Petabyte

Cluster specs

1200 nodes
2 Intel quad-core per node
48 GB RAM per node
7200 RPM sata drives

- **Huge** problem is 19.5x more than cluster memory
- linear performance from 1 trillion to 70 trillion edges...
- despite multiple hardware failures!

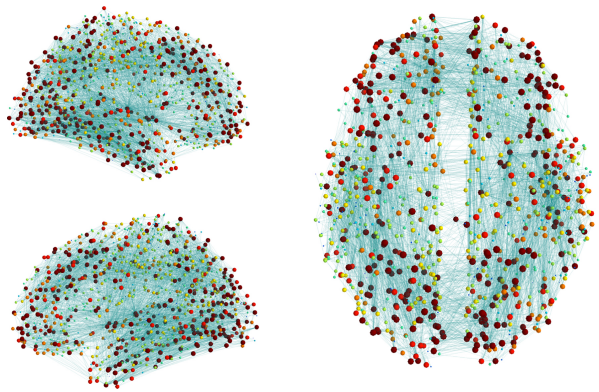


Graph500 Scalability Benchmark

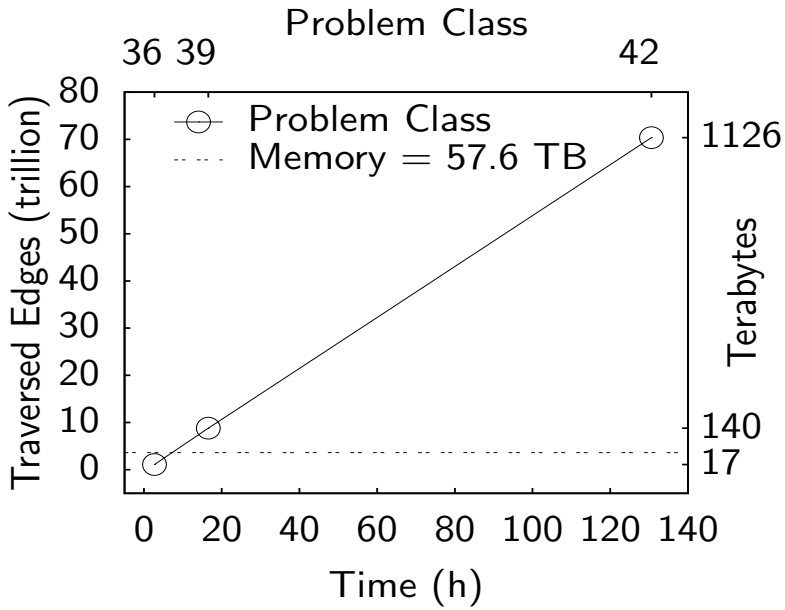
Graph500 Experiment

Graph500 **Huge** class — scale 42

2^{42} (4.40 trillion) vertices
2^{46} (70.4 trillion) edges
1 Petabyte



- **Huge** problem is 19.5x more than cluster memory
- linear performance from 1 trillion to 70 trillion edges...
- despite multiple hardware failures!



Graph500 Scalability Benchmark

Future work

What are the tradeoffs between disk- and memory-based *Big Graph* solutions?

- power–space–cooling efficiency
- software development and maintenance

Can a hybrid approach be viable?

- memory-based processing for subgraphs or incremental updates
- disk-based processing for global analysis

Advances in memory and disk technologies will blur the lines. Will solid-state disks be another level of memory?