Cloud-based approach to Big Graphs

HPEC 2015

Paul Burkhardt, Chris Waring

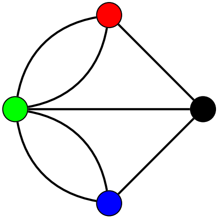
U.S. National Security Agency
Research Directorate - R6

September 17, 2015

Graph analysis begins as an abstraction of a problem

Crossing the River Pregel

Determining if each of the Seven Bridges of Königsberg can be crossed exactly once depends only upon the connectivity of the endpoints. . .



Network of nodes and links

- Concise representation of unstructured data —
- transform *hairballs* into regular, structured format of binary relationships
- Facilitates powerful algorithmic approach to network analysis. . .
  - shortest-paths
  - centrality
  - clustering

Scaling problem

Analyzing graphs is hard...

- topology is irregular thus...
- difficult to leverage computer memory hierarchy increasing...
- latency and bandwidth costs and...
- high degree vertices cause hot-spots

At *Big Data* scales it gets harder...

- challenges of graph analysis are exacerbated as size increases
- graph data can be too big to fit in total memory...
- too big to save global state information...
- often lack of random access to vertices and edges

Conventional memory-bound computational approaches

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

Physical limitations to memory-bounded approaches

SHARED-MEMORY

- Globally-shared memory limited by CPU addressing limit
- Top-end CPUs have 46-bit physical memory address space...
- therefore limited to 64 Terabytes of globally-shared memory

DISTRIBUTED-MEMORY

- per machine memory limited by...
- number of CPU pins, memory controller channels, DIMMS per channel...

Poor data locality affects memory throughput...

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 data locality affects memory throughput...

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

Poor data locality affects memory throughput...

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

Sequential block read on disk...

About 100 MB/s...

Choosing a *Big Graph* data structure

Undirected graph data structure space complexity

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

Need compact structure that can be easily distributed...

- edge list representation wins but...
- no direct access to neighbors of a specific vertex... unless edges are sorted and indexed by groups

Applying Cloud technologies to *Big Graphs*

Build a system with the following requirements

- *out-of-core* computational paradigm — not memory-bound
- scalable, distributed $\langle key, value \rangle$ repository to store edge list
- using only commodity hardware

Try the following Cloud technologies

- Accumulo
- MapReduce

NSA BigTable — Apache Accumulo

Extending BigTable

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*

Edge Table graph data structure

Using an Accumulo Table provides random access to edges!

- *Edge Table* is distributed into many *tablets*...
- each tablet has an index of its contents
- tablet contents are sorted by the *ROW ID*
- all values for a *ROW ID* coincide on the same tablet
- (v, u) edges are stored in the *ROW ID* to permit large adjacencies to split across multiple tablets

Accumulo *BatchScanner* to access vertex adjacencies

- multi-threaded Accumulo iterator
- scan a batch of row identifiers
- scan multiple tablets concurrently

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

KEY					VALUE
ROW ID	COLUMN			TIMESTAMP	
	FAMILY	QUALIFIER	VISIBILITY		

Using Apache MapReduce for *Big Graphs*

No longer memory-bound

Can process graphs that do not fit in memory!

Rethinking graph algorithms in MapReduce

- stateless – no history or globally-shared data
- scale-free memory – fixed working memory per task
- edge-centric – compute on pairwise information, e.g. endpoints of edges, paths. . .
- streaming – tasks do not re-read their input
- data-independent – no communication between tasks of the same phase

Implementation tips

- 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

Review of the conventional BFS algorithm

trivial BFS algorithm

Require: $s \longleftarrow source$
  allocate stack named *queue*
  allocate set named *visit*
  push(*queue*, s)
  while *queue* not empty **do**
    $v \longleftarrow$ pop(*queue*)
    for all $u \in N(v)$ **do**
      if $u \notin visit$ **then**
        visit add u
        push(*queue*, u)
      end if
    end for
  end while

Random Access Machine algorithm (RAM)

This algorithm uses a graph data structure such as the *adjacency list* with $O(1)$ access to the neighborhood $N(v)$ of a vertex v

For n vertices and m edges, BFS takes $O(n+m)$ time...

- $O(n)$ access to all $N(v)$...
- $O(m)$ tests for $u \in N(v)$ for all v to avoid duplicate visits

$$\sum_v d_v = 2m$$

where d_v is the degree of a vertex v

BFS is hard on *Big Graphs*

BFS memory latency cost

Linear, $O(n + m)$, in computer memory references...

But n and m can be trillions for *Big Graphs*!

Exploit rediscovery cycle in BFS on undirected graphs

Rediscovery cycle

A vertex will be rediscovered within a cycle of at most length two due to symmetry.

Definition

The $k+1$ step in BFS expands from the neighborhood of vertices at the *kth* step, then define the following:

- Let V_k be the set of frontier vertices at k step, and
- Let $N_{k+1} = \biguplus_{v \in V_k} N(v)$ be the multiset of neighbors of the frontier.

Observation

Then it follows that V_k is the subset in N_k which have not been visited in the last two steps.

$$V_k = (N_k \setminus V_{k-1}) \cap (N_k \setminus V_{k-2}) = N_k \setminus (V_{k-1} \cup V_{k-2})$$

Reducing the k-level input to BFS

Minimizing state information

Only the last two frontier sets must be saved

$$V_k = N_k \setminus (V_{k-1} \cup V_{k-2})$$

Sliding Window for BFS on undirected graphs

Employ a *sliding window* to minimize input at each k step

Terms and Definitions

For a simple, undirected graph $G = (V, E)$

- with $n = |V|$ vertices and $m = |E|$ edges where ...
- $N(v) = \{u \in V \mid (v, u) \in E\}$ is the neighborhood of v
- $d_v = |N(v)|$ is the degree of v
- $d_{max} = max\{d_v \mid v \in V\}$ is the maximum degree
- $d(u, v)$ is the distance, i.e. shortest-path, from u to v
- D_G is the diameter of G, i.e. longest shortest-path in G
- V_k is the frontier set of vertices at distance k from the source
- $N_k = \biguplus_{v \in V_{k-1}} N(v)$ is the multiset of neighbors of V_{k-1}

Our Cloud-based BFS

MapReduce algorithm

For each v with all k values equal, thus is a $v \in V_k$, create $\langle v, k \rangle$ and $\{\langle u, k+1 \rangle \mid u \in N(v), v \in V_k\}$

After each k step, keep only $\langle v, \mathit{distance} \rangle$ pairs from V_k, V_{k-1}, and N_{k+1} for the following $k+1$ step.

Cloud-based BFS algorithm

Require: $K \leftarrow$ number of iterations
  for $k = 0$ to K **do**
    set input to V_{k-1}, V_{k-2} and N_k
    set output directory to V_k, N_{k+1}
    Map: Identity
    Reduce: input $\leftarrow \langle \mathit{key}, \{\mathit{values}\} \rangle$
    if every element in *values* is k **then**
      for all v in adjacency(*key*) **do**
        output $\langle v, k+1 \rangle$ to N_{k+1} directory
      end for
      output $\langle \mathit{key}, k \rangle$ to V_k directory {induced loop record}
    end if
  end for

Adjacency computation

This algorithm uses a *BatchScanner* to scan the *Edge Table* for adjacencies from multiple vertices simultaneously. . . not shown here

Task count vs concurrency

Balance number of reduce tasks

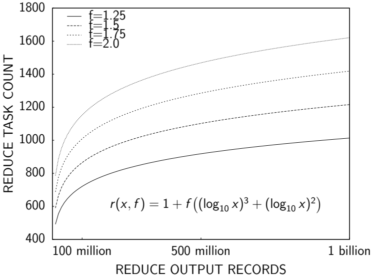
The number of map tasks is set by input size but the reduce task count is user-defined...

- each map task sends to every reduce task – Cartesian product in communication costs!
- balance concurrency needs with available resources...

Automatically scale reduce count for BFS

- when input for reduce phase at each k-step is not known *a priori*...
- and cluster resources fluctuate...
- use the following *quick-and-dirty* scaling function...

$$R(x, f) = 1 + f\left((\log_{10} x)^3 + (\log_{10} x)^2\right)$$



Custom Partitioner

Partition keys to align with tablets

- Sample distribution of content in the *Edge table*
    - Accumulo provides table splits
- Assign keys in round-robin to align with table splits...
- minimizing the overlap of scans across all tablets

Breadth-First Search (BFS) on an undirected R-MAT Graph

- count *Traversed Edges per Second* (TEPS)
- $n = 2^{scale}$, $m = 16 \times n$

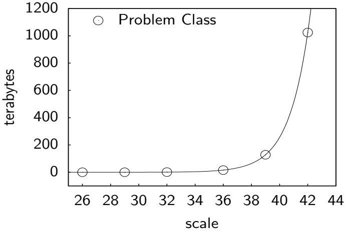


Figure: Graph500 Problem Sizes

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

BFS on 1 PB Graph with 70 trillion edges

- 150 million edges per second
- 19.5x more than cluster memory
- 16x larger problem than top competitor in Graph500 June 2012
- linear performance from 1 trillion to 70 trillion edges...

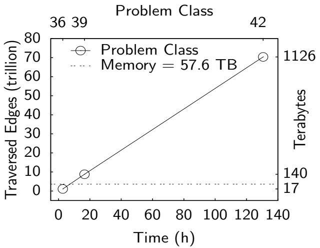


Figure: Graph500 Scalability Benchmark

Presented at the CMU 2013 SDI Seminar series

http://www.pdl.cmu.edu/SDI/2013/slides/big_graph_nsa_rd_2013_56002v1.pdf