Algorithms for Distributed Stream Processing

Ashish Goel
Stanford University

Joint work with
I. Bahman Bahmani and Abdur Chowdhury; VLDB 2011
II. Bahman Bahmani and Rajendra Shinde
III. Michael Kapralov, Olga Kapralova, and Sanjeev Khanna

January 12, 2012

Active DHTs and Distirbuted Stream Processing

Incremental PageRank

A Diversion: Recommendation Systems

Fast Incremental PageRank via Monte Carlo

Locality Sensitive Hashing

Graph Sparsification in Active DHTs

An immensely successful idea which transformed offline analytics and bulk-data processing. Hadoop (initially from Yahoo!) is the most popular implementation.

- Map: Transforms a (key, value) pair into other (key, value) pairs using a UDF (User Defined Function) called **Map**. Many mappers can run in parallel on vast amounts of data in a distributed file system
- Shuffle: The infrastructure then transfers data from the mapper nodes to the "reducer" nodes so that all the (key, value) pairs with the same key go to the same reducer
- Reduce: A UDF that aggregates all the values corresponding to a key. Many reducers can run in parallel.

Active DHT

- Distributed Hash Table: Stores key-value pairs; supports insertion, lookup, and deletion
- **Active DHT**: Can supply arbitrary UDFs (User Defined Functions) to be executed on a key-value pair
- Examples: Twitter's Storm; Yahoo's S4 (both open source)
- Challenge: At high volume, small requests are not network efficient
- Challenge: Robustness
- Application: Distributed Stream Processing
- Application: Continuous Map-Reduce
- Active DHTs subsume bulk-synchronous graph processing systems such as Pregel

An Example Application of Continuous Map Reduce

- Problem: There is a stream of data arriving (eg. tweets) which needs to be farmed out to many users/feeds in real time
- A simple solution:

  MAP: $(\text{user } u, \text{string } \textit{tweet}, \text{time } t) \Rightarrow$
  $(v_1, (\textit{tweet}, t))$
  $(v_2, (\textit{tweet}, t))$
  $\ldots$
  $(v_K, (\textit{tweet}, t))$ where $v_1, v_2, \ldots, v_K$ follow u.

  REDUCE:
  $(\text{user } v, (\textit{tweet}_1, t_1), (\textit{tweet}_2, t_2), \ldots, (\textit{tweet}_J, t_J)) \Rightarrow$
  sort tweets in descending order of time or importance

- With Active DHTs, this and many other real-time web problems would become very simple to implement

- Number of network calls per update
- Size of network data transfer per update
- Maximum size of a key-value pair
- Total size of all key-value pairs
- Maximum number of requests that go to a particular key-value pair (akin to the curse of the last reducer)

Active DHTs and Distirbuted Stream Processing

Incremental PageRank

A Diversion: Recommendation Systems

Fast Incremental PageRank via Monte Carlo

Locality Sensitive Hashing

Graph Sparsification in Active DHTs

- An early and famous search ranking rule [Brin et al. 1998]
- Premise: Treats each hyperlink as an endorsement. You are highly reputed if other highly reputed nodes endorse you.
- Formula: N nodes, M edges, V is the set of nodes, E is the set of edges, ϵ is the "teleport" probability, $d(w)$ is the number of outgoing edges from node w, $\pi(w)$ is the PageRank. Now,

$$\pi(v) = \epsilon/N + (1-\epsilon) \sum_{(w,v)\in E} \pi(w)/d(w).$$

- Another interpretation: A random surfer traverses the web-graph, teleporting to a random node with probability ϵ at every step, and following a random hyperlink otherwise; π is the stationary distribution.

PageRank

- An early and famous search ranking rule [Brin et al. 1998]
- Premise: Treats each hyperlink as an endorsement. You are highly reputed if other highly reputed nodes endorse you.
- Formula: N nodes, M edges, V is the set of nodes, E is the set of edges, ϵ is the "teleport" probability, $d(w)$ is the number of outgoing edges from node w, $\pi(w)$ is the PageRank. Now,

$$\pi(v) = \epsilon/N + (1-\epsilon) \sum_{(w,v)\in E} \pi(w)/d(w).$$

- Another interpretation: A random surfer traverses the web-graph, teleporting to a random node with probability ϵ at every step, and following a random hyperlink otherwise; π is the stationary distribution.

PageRank in Social Networks

- A follows B or A is friends with B $\Rightarrow$ A endorses B
- Incremental: Update as soon as an edge arrives; needs to be efficient enough to also add "quasi-edges" eg. A clicks on something that B sent out, or A liked B, or retweeted B
- Personalized: Assume a teleport vector $\langle \epsilon_1, \epsilon_2, \ldots, \epsilon_N \rangle$ such that $\sum_i \epsilon_i = \epsilon$. Now, define

$$\pi(v) = \epsilon_v + (1 - \epsilon) \sum_{(w,v) \in E} \pi(w)/d(w).$$

  - Set $\epsilon_w = \epsilon$ and $\epsilon_i = 0$ for all other nodes $\Rightarrow$ Personalized PageRank for node w
- Goal: To maintain PageRank efficiently as edges arrive.

Two Approaches to Computing PageRank

- The power-iteration method: Set $\pi_0(w) = 1/N$ for all nodes, and run R iterations of

$$\pi_{r+1}(v) = \epsilon/N + (1-\epsilon) \sum_{(w,v)\in E} \pi_r(w)/d(w).$$

  Use π_R as an estimate of π.

- The Monte Carlo method: For each node v, simulate R PageRank random walks starting at v, where each random walk terminates upon teleportation. If node w is visited $\#(w)$ times, then use $\#(w) \cdot \frac{\epsilon}{RN}$ as an estimate of π
- $R = O(\log N)$ suffices for good estimates (the exact bounds differ).

Computing Incremental PageRank

Goal: Maintain an accurate estimate of PageRank of every node after each edge arrival.

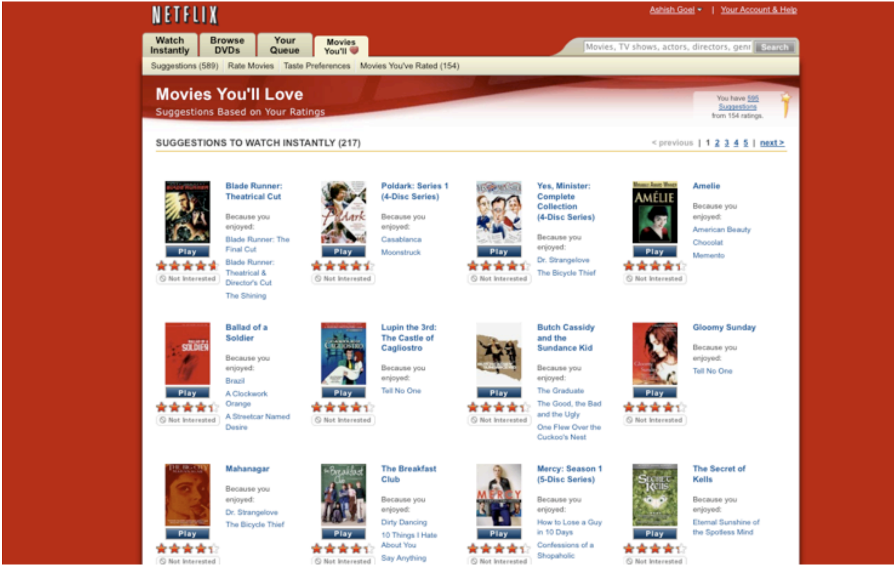
- Naive Approach 1: Run the power iteration method from scratch: Total time over M edge arrivals is $O(RM^2)$.
- Naive Approach 2: Run the Monte Carlo method from scratch: Total time over M edge arrivals is $O(RMN/\epsilon)$.
- Many heuristics known, but none is asymptotically a large improvement over the naive approaches.
- Our result: Implement Monte Carlo in total time $O^*(\frac{NR \log N}{\epsilon^2})$ under mild assumptions.

Recommendation Systems

Goal: Make personalized recommendations of goods that a consumer may like

Three integral parts:

- Collect data about users' preferred goods; Explicit (Netflix ratings) or Implicit (Amazon purchases)
- Identify similar users to a given client, or similar goods to a given good
- Use this similarity to find other goods that the client may want to consume
- The "good" could be another user, if we are doing friend suggestion in a social network


NETFLIX
Ashish Goel | Your Account & Help
Watch Instantly
Browse DVDs
Your Queue
Movies You'll
Search
Suggestions (589) Rate Movies Taste Preferences Movies You've Rated (154)
Movies You'll Love
Suggestions Based on Your Ratings
You have 595 Suggestions from 154 ratings.
SUGGESTIONS TO WATCH INSTANTLY (217)
< previous | 1 2 3 4 5 | next >
Blade Runner: Theatrical Cut
Because you enjoyed: Blade Runner: The Final Cut Blade Runner: Theatrical & Director's Cut The Shining
Poldark: Series 1 (4-Disc Series)
Because you enjoyed: Casablanca Moonstruck
Yes, Minister: Complete Collection (4-Disc Series)
Because you enjoyed: Dr. Strangelove The Bicycle Thief
Amelie
Because you enjoyed: American Beauty Chocolat Memento
Ballad of a Soldier
Because you enjoyed: Brazil A Clockwork Orange A Streetcar Named Desire
Lupin the 3rd: The Castle of Cagliostro
Because you enjoyed: Tell No One
Butch Cassidy and the Sundance Kid
Because you enjoyed: The Graduate The Good, the Bad and the Ugly One Flew Over the Cuckoo's Nest
Gloomy Sunday
Because you enjoyed: Tell No One
Mahanagar
Because you enjoyed: Dr. Strangelove The Bicycle Thief
The Breakfast Club
Because you enjoyed: Dirty Dancing 10 Things I Hate About You Say Anything
Mercy: Season 1 (5-Disc Series)
Because you enjoyed: How to Lose a Guy in 10 Days Confessions of a Shopaholic
The Secret of Kells
Because you enjoyed: Eternal Sunshine of the Spotless Mind
Play
Not Interested

amazon.com

Hello, Ashish Goel. We have recommendations for you. (Not Ashish?)

Ashish's Amazon.com | Today's Deals | Gifts & Wish Lists | Gift Cards

Shop All Departments | Search | All Departments

Your Amazon.com | Your Browsing History | Recommended For You | Rate These Items | Improve Your Recommendations

shish's Amazon.com › **Recommended for You**

f you're not Ashish Goel, click here.)

Just For Today

Browse Recommended

Recommendations

- All Electronics
- Baby
- Beauty
- Books
- Books on Kindle
- Camera & Photo
- Clothing & Accessories
- Computer & Accessories
- Grocery & Gourmet Food
- Health & Personal Care
- Home Improvement
- Industrial & Scientific
- Jewelry
- Kitchen & Dining
- MP3 Downloads
- Magazine Subscriptions
- Movies & TV
- Music
- Musical Instruments
- Office Products
- Patio, Lawn & Garden
- Shoes

These recommendations are based on items you own and more.

view: **All** | New Releases | Coming Soon

1. 

Shuffled Row
by Amazon Digital Services (August 2, 2010)
Average Customer Review: (76)
Auto-delivered wirelessly

Price: $0.00

Offered by Amazon Digital S

Add to Cart | Add

I own it | Not interested | Rate this item

Recommended because you purchased **U.S. Personal Document Service** and more (Fix this)

2. 

Lost Hero, The
by Rick Riordan (October 12, 2010)
Average Customer Review: (61)
Auto-delivered wirelessly

Kindle Price: $9.74

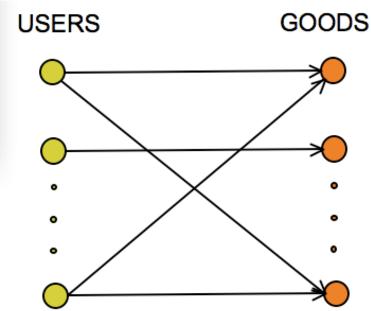
I own it | Not interested | Rate this item

Recommended because you purchased **The Last Olympian (Percy Jackson and the Olympians, Book 5)** and more (Fix this)

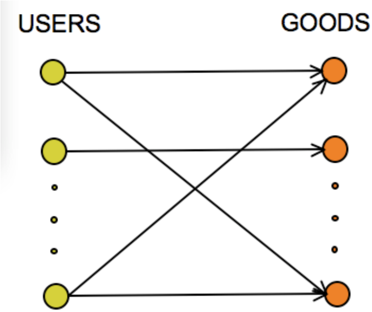
3. 

Mech
by B. V. Larson (June 10, 2010)
Average Customer Review: (26)
Auto-delivered wirelessly

Kindle Price: $0.00



The arrow could denote LIKES or CONSUMES or FOLLOWS



Compute similarity score on the left, propagate it to relevance score on the right, and then vice-versa; repeat a few times

Starting point: A client C is most similar to herself

- How do we do this propagation? Two extremes:
  - LOVE: All the similarity score of a user X gets transferred to each good that X likes, and the same in the reverse direction. (Same as HITS)
  - MONEY: If X likes K goods, then a (1/K) fraction of the similarity score of X gets transferred to each good that X likes (Same as SALSA)
- Empirical finding: MONEY does far better than LOVE
- Observation: Computing MONEY is the same as doing PageRank in a graph with all the edges converted to being bidirectional

Dark Test: Run various algorithms to recommend friends, but don't display the results. Instead, just observe how many recommendations get followed *organically*.

	HITS	COSINE	Personalized PageRank	SALSA
Top 100	0.25	4.93	5.07	6.29
Top 1000	0.86	11.69	12.71	13.58

Table: Link Prediction Effectiveness


Lists
uccione has died of cancer
ttp://on.cnn.com/94hiXv
by toptweets
O'Higgins, Chile. Oct 20
WNW of Talca, depth...
something I am proud of.
tomorrow.
til you refuse to correct it...
Your Tweets 94
4 Oct: @lloydoftheflies Yes. Eg: Find all points within distance/hitting time less than k from node v; do a SVD; return top 3 eigenvectors, etc.
Following 115
Followers 1,416
Listed 100
Trends
Worldwide · change
#Nike600K Promoted
#themwasthedays
#yeaisaidit
#mickyyoochun
Guccione
Mac App
iLife
Apple Killed
Slits
Blu-ray
Who to follow
Suggestions for you · refresh
sara · Follow
sara
stop · Follow
Doug Bowman
dannysullivan · Follow
Danny Sullivan
bgurley · Follow
Bill Gurley
View all suggestions
Browse interests · Find friends

The Random Permutation Model

- Assume edges of a network are chosen by an adversary, but then these edges arrive in random order.
- At time $t = 1, 2, \ldots M$:
  Arriving edge = $\langle u_t, v_t \rangle$
  Out degree of node $w = d_t(w)$
  PageRank of node $w = \pi_t(w)$
- Technical consequence: $\mathbb{E}[\pi_{t-1}(u_t)/d_t(u_t)] = 1/t$
- Impossible to verify assumption given a single network, but we empirically validated the above technical consequence for the twitter network

Algorithm for Incremental PageRank

- Initialize: Store R random walks starting at each node
- At time t, for every random walk passing through node u_t, shift it to use the new edge $\langle u_t, v_t \rangle$ with probability $1/d_t(u_t)$
- Time for each re-routing: $O(1/\epsilon)$.
- Time to decide whether any walk will get rerouted: $O(1)$
- Claim: This faithfully maintains R random walks after arbitrary edge arrivals.

Observe that we need the graph and the stored random walks to be available in an Active DHT; this is a reasonable assumption for social networks, though not necessarily for the web-graph.

Algorithm for Incremental PageRank

- Initialize: Store R random walks starting at each node
- At time t, for every random walk passing through node u_t, shift it to use the new edge $\langle u_t, v_t \rangle$ with probability $1/d_t(u_t)$
- Time for each re-routing: $O(1/\epsilon)$.
- Time to decide whether any walk will get rerouted: $O(1)$
- Claim: This faithfully maintains R random walks after arbitrary edge arrivals.

Observe that we need the graph and the stored random walks to be available in an Active DHT; this is a reasonable assumption for social networks, though not necessarily for the web-graph.

Remember the technical consequence of the random permutation model: $\mathbb{E}[\pi_{t-1}(u_t)/d_t(u_t)] = 1/t$.

- Expected running time at time t
  $= \mathbb{E}[(\text{Number of random walks rerouted})]/\epsilon$
  $= \mathbb{E}[(\text{Number of random walks via } u_t)/d_t(u_t)]/\epsilon$
  $= \mathbb{E}[(RN/\epsilon)\pi_{t-1}(u_t)/d_t(u_t)]/\epsilon$
  $= (RN/\epsilon^2)/t$ [From technical assumption].
- Total running time =
  $$O((RN/\epsilon^2)\sum_{t=1}^{M} 1/t) = O((RN\log M)/\epsilon^2)$$
  (ignoring time taken to actually make the decision whether to reroute a random walk)

Verifying $\mathbb{E}[\pi_{t-1}(u_t)/d_t(u_t)] = 1/t$

In the random permutation model, any of the t edges present at the end of time t is equally likely to have been the last to arrive, i.e. $\mathbb{P}[u_t = x] = d_t(x)/t$. Hence,

$$\begin{aligned}
\mathbb{E}[\pi_{t-1}(u_t)/d_t(u_t)] &= \sum_{x \in V} \mathbb{P}[u_t = x]\pi_{t-1}(x)/d_t(x) \\
&= \sum_{x \in V} \pi_{t-1}(x)/t \\
&= 1/t
\end{aligned}$$

Also, empirically verified on Twitter's network.

Directions

- Extend running time result to adversarial arrival (lower bound by [Lofgren 2012])
- Efficient personalized search: combine inverted indexes with personalized reputation systems: recent progress by Bahmani and Goel
- Speed up incremental computation of other graph and IR measures, assuming random permutation model

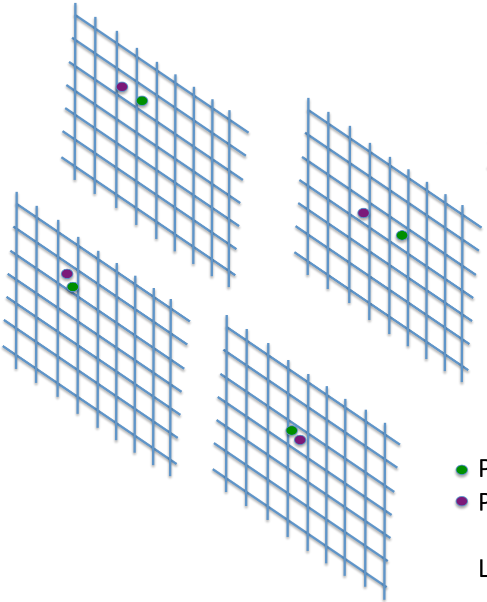
- A Hash Family H is said to be a (l, u, p_l, p_u)-LSH if
  1. For any two points x, y such that $||x - y||_2 \leq l$, $\mathbb{P}[h(x) = h(y)] \geq p_l$, and
  2. For any two points x, y such that $||x - y||_2 \geq u$, $\mathbb{P}[h(x) = h(y)] \leq p_u$,

  where h is a hash function chosen uniformly from the family H
- Given a LSH family, one can design an algorithm for the (l, u) Near Neighbor problem that uses $O(n^\rho)$ hash functions, where n is the number of points, and $\rho = \frac{\log p_l}{\log p_u}$
- We can obtain $\rho = l/u$ using a simple LSH family
- The idea extends to metrics other than ℓ_2

[Indyk-Motwani 2004, Andoni-Indyk 2006]

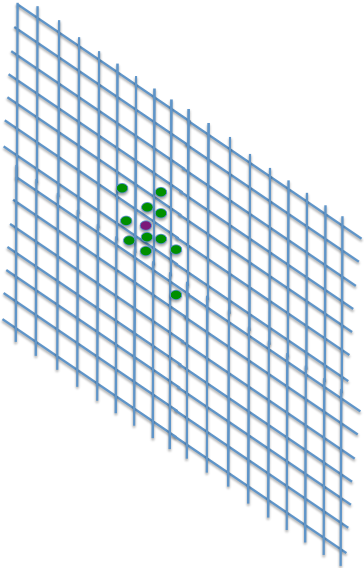
- Project every point to a set of K randomly chosen lines; the position of the point on the K lines defines a hash function f.
- Impose a random grid on this K dimensional space; the identifier for the grid cell in which a point x falls is $h(x)$
- For each database point x and each query point q, we would generate $L = n^\rho$ key-value pairs in the map stage
- Data points: $\text{Map}(x) \rightarrow \{(h_1(x), x, 0), \ldots, (h_L(x), x, 0)\}$
- Query points: $\text{Map}(q) \rightarrow \{(h_1(q), q, 1), \ldots, (h_L(q), q, 1)\}$
- Reduce : For any hash cell, see if any of the query points is close to any of the data points
- Problem: Shuffle size will be too large for Map-Reduce/Active DHTs
- Problem: Total space used will be very large for Active DHTs

- Instead of hashing each point using $L = n^{\rho}$ different hash functions, hash $L = n^{2\rho}$ perturbations of the query point using the same hash function [Panigrahi 2006].
- $\text{Map(q)} \rightarrow \{(h(q + \delta_1), q, 1), \ldots, (h(q + \delta_L), q, 1)\}$
- Reduces space in centralized system, but still has a large shuffle size in Map-Reduce and too many network calls over Active DHTs



- Projection of query point
- Projection of data point

L = N^{ρ} Hash functions

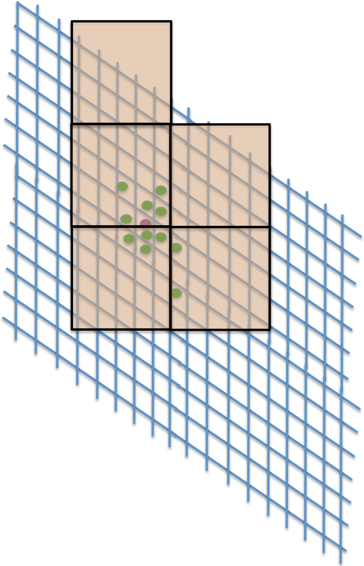


Hopefully, one of the query offsets maps to the same cell as the close by data point

- Projection of query offset
- Projection of data point

$L = N^{2\rho}$ query offsets

Reapplying LSH to Entropy LSH

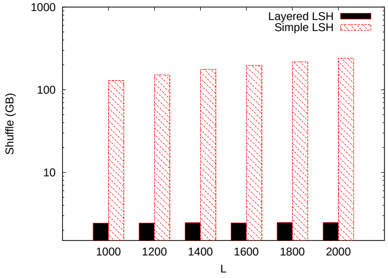


Apply another LSH to the grid cells, and use the "meta-cell" as the key.

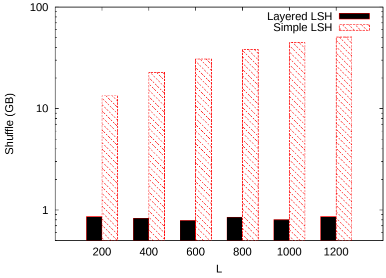
Intuition: All the query offsets get mapped to a small number of meta-cells

- Projection of query offset
- Projection of data point

$L = N^{2\rho}$ query offsets



(a) Random data



(b) An image database

- Number of network calls/shuffle-size/space per data point: $O(1)$
- Number of network calls/shuffle-size/space per query point: $O(\sqrt{\log n})$
- Maximum size of a key-value pair: Not analyzed. But we can show that for some small constant c, if $||x - y||_2 > cl$ then $\mathbb{P}[g(x) = g(y)] < 1/2$ where g is the meta-cell.
- Maximum number of requests that go to a particular key-value pair: Same analysis as above
- **Open Problems:** Optimum tradeoff? Extend to dense point sets?

- Number of network calls/shuffle-size/space per data point: $O(1)$
- Number of network calls/shuffle-size/space per query point: $O(\sqrt{\log n})$
- Maximum size of a key-value pair: Not analyzed. But we can show that for some small constant c, if $||x - y||_2 > cl$ then $\mathbb{P}[g(x) = g(y)] < 1/2$ where g is the meta-cell.
- Maximum number of requests that go to a particular key-value pair: Same analysis as above
- **Open Problems:** Optimum tradeoff? Extend to dense point sets?

Graph Sparsification via Union-Find

- Typical Approach to Graph Sparsification: For every edge e, assign a weight w_e
  - Sample the edge with probability $1/w_e$ and assign it weight w_e if sampled.
  - Weight w_e typically measures the "connectivity strength" of the endpoints of the edge in the graph [Benczur-Karger 1996, Spielman-Teng 2004]
- Our observation: We can use a series of nested Union-Find data structures to estimate this weight [details omitted]
  - Stream Processing: Since Union-Find is an easy structure to update, we get an efficient algorithm for streaming sparsification
  - Other approaches to streaming sparsification exist [Ahn-Guha 2009, Fung et al. 2011], but Union-Find will be easy to "distribute"

A connectivity data structure. Every node u maintains a parent pointer $p(u)$, and a node u is a root if $p(u) = u$. The structure is acyclic, so every node has a root that can be found by following parent pointers.

Find(u) Keep following parent pointers from u till we get to a root r

Path compression: set $p(v) = r$ for every node v on the path from u to r.

Union(u, v) Compute $a = \text{Find}(u)$; $b = \text{Find}(v)$. Assume a has smaller "rank". Set $p(a) = b$.

Amortized time: $O(\log^* n)$ per call.

Union-Find in an Active DHT

- Treat the parent array p as a set of key-value pairs $(u, p(u), \mathrm{rank}(u))$.
- Number of network calls per update: $O(\log^* n)$ amortized
- Maximum size of a key-value pair: $O(1)$
- Total number of key-value pairs: $O(n)$
- Problem: Maximum number of queries to a key-value pair is $O(m)$.
  - Once a graph gets connected, every Find query hits the root, and there are $O(m)$ Union queries, each triggering two Find queries.
  - Fix: Zig-zag Find. In Union(u, v), first compare whether $p(u) = p(v)$ and trigger a full Find only when they are not equal
  - Maximum load on a key-value pair: $O(n \log^* n)$. Other performance measures unaffected

- Treat the parent array p as a set of key-value pairs $(u, p(u), \text{rank}(u))$.
- Number of network calls per update: $O(\log^* n)$ amortized
- Maximum size of a key-value pair: $O(1)$
- Total number of key-value pairs: $O(n)$
- Problem: Maximum number of queries to a key-value pair is $O(m)$.
  - Once a graph gets connected, every Find query hits the root, and there are $O(m)$ Union queries, each triggering two Find queries.
  - Fix: Zig-zag Find. In Union(u, v), first compare whether $p(u) = p(v)$ and trigger a full Find only when they are not equal
  - Maximum load on a key-value pair: $O(n \log^* n)$. Other performance measures unaffected

- A Distributed Stream Processing Algorithm for Sparsification
- Total space used: $\tilde{O}(n)$
- Size of key-value pair: $O(1)$
- Amortized update complexity:
  - Number of network calls: $\tilde{O}(1)$
  - Amount of data transfer: $\tilde{O}(1)$
  - Total amount of computation: $\tilde{O}(1)$
- Total number of calls to a specific key-value pair: $O(n \log^* n)$.

Conclusion

- Active DHTs can do to real-time computation what Map-Reduce did to Bulk processing
- Many algorithmic issues, some discussed here
    - Graph algorithms (eg. sparsification)
    - Search/social search (eg. PageRank)
    - Mining large data sets (eg. LSH)
- Directions: Optimization; Robustness; Other basic graph, search, and data-processing measures

THANK YOU

K. Ahn and S. Guha.
On graph problems in a semi-streaming model.
Automata, languages and programming: Algorithms and complexity, pages 207 – 216, 2009.

A. Andoni and P. Indyk.
Near optimal hashing algorithms for approximate nearest neighbor in high dimensions.
FOCS '06.

András A. Benczúr and David R. Karger.
Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time.
Proceedings of the 28th annual ACM symposium on Theory of computing, pages 47–55, 1996.

S. Brin, L. Page, R. Motwani, and T. Winograd
What can you do with a Web in your Pocket?, 1998

Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi.
A general framework for graph sparsification.
STOC, pages 71–80, 2011.

M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni.
Locality sensitive hashing scheme based on p-stable distributions.
SoCG '04.

P. Lofgren.
A lower bound on amortized complexity of the Monte Carlo method for incremental PageRank.
Personal communication.

R. Panigrahi.
Entropy based nearest neighbor search in high dimensions.
SODA '06.

Daniel A. Spielman and Shang-Hua Teng.
Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems.

STOC '04, 2004.