Social Search and Related Problems

Bahman Bahmani
Ashish Goel

Stanford University
Challenge

• Careful extension of existing algorithms to modern data models
• Large body of theory work
  – Distributed Computing
  – PRAM models
  – Streaming Algorithms
  – Sparsification, Spanners, Embeddings
  – LSH, MinHash, Clustering
  – Primal Dual
• Adapt the wheel, not reinvent it
Data Model #1: Map Reduce

- 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.
A Motivating Example: Continuous Map Reduce

• There is a stream of data arriving (e.g. tweets) which needs to be mapped to timelines

• Simple solution?
  – Map: \((\text{user } u, \text{ string } \text{tweet}, \text{ time } t) \rightarrow (v_1, (\text{tweet}, t)) (v_2, (\text{tweet}, t)) \ldots (v_K, (\text{tweet}, t))\) where \(v_1, v_2, \ldots, v_K\) follow \(u\).
  – Reduce: \((\text{user } v, (\text{tweet}_1, t_1), (\text{tweet}_2, t_2), \ldots (\text{tweet}_J, t_J)) \rightarrow \text{sort tweets in descending order of time}\)
Data Model #2: Active DHT

• DHT (Distributed Hash Table): Stores key-value pairs in main memory on a cluster such that machine $H(key)$ is responsible for storing the pair $(key, val)$

• Active DHT: In addition to lookups and insertions, the DHT also supports running user-specified code on the $(key, val)$ pair at node $H(key)$

• Like Continuous Map Reduce, but reducers can talk to each other
Problem #1: Incremental PageRank

• Assume social graph is stored in an Active DHT
• Estimate PageRank using Monte Carlo: Maintain a small number $R$ of random walks (RWs) starting from each node
• Store these random walks also into the Active DHT, with each node on the RW as a key
  — Number of RWs passing through a node $\sim$ PageRank
• New edge arrives: Change all the RWs that got affected
• Suited for Social Networks
Incremental PageRank

• Assume edges are chosen by an adversary, and arrive in random order
• Assume $N$ nodes
• Amount of work to update PageRank estimates of every node when the $M$-th edge arrives = $(RN/\varepsilon^2)/M$ which goes to 0 even for moderately dense graphs
• Total work: $O((RN \log M)/\varepsilon^2)$
• Consequence: Fast enough to handle changes in edge weights when social interactions occur (clicks, mentions, retweets etc)

[Joint work with Bahmani and Chowdhury]
Data Model #3: Batched + Stream

- Part of the problem is solved using Map-Reduce/some other offline system, and the rest solved in real-time
- Example: The incremental PageRank solution for the Batched + Stream model: Compute PageRank initially using a Batched system, and update in real-time
- Another Example: Social Search
Problem Statement: Real-Time Social Search

• Real-Time Social Search: Find a piece of content that is exciting to my extended network right now and matches my search criteria

• Hard technical problem: imagine building 100M real-time indexes over real-time content
Related Work: Social Search

- Social Search problem and its variants heavily studied in literature:
  - Name search on social networks: Vieira et al. '07
  - Social question and answering: Horowitz et al. '10
  - Personalization of web search results based on user’s social network: Carmel et al. '09, Yin et al. '10
  - Social network document ranking: Gou et al. '10
  - Search in collaborative tagging nets: Yahia et al '08

- Shortest paths proposed as the main proxy
Current Status: No Known Efficient, Systematic Solution...
... Even without the Real-Time Component
Related Work: Distance Oracles

• Approximate distance oracles: Bourgain, Dor et al '00, Thorup-Zwick '01, Das Sarma et al '10, ...

• Family of Approximating and Eliminating Search Algorithms (AESA) for metric space near neighbor search: Shapiro '77, Vidal '86, Micó et al. '94, etc.

• Family of "Distance-based indexing" methods for metric space similarity searching: surveyed by Chávez et al. '01, Hjaltason et al. '03
Formal Definition

• The Data Model
  – Static undirected social graph with $N$ nodes, $M$ edges
  – A dynamic stream of updates at every node
  – Every update is an addition or a deletion of a keyword
    • Corresponds to a user producing some content (tweet, blog post, wall status etc) or liking some content, or clicking on some content
    • Could have weights

• The Query Model
  – A user issues a single keyword query, and is returned the closest node which has that keyword
The Processing Model: Active DHT

- DHT (Distributed Hash Table): Stores key-value pairs in main memory on a cluster such that machine $H(\text{key})$ is responsible for storing the pair $(\text{key}, \text{val})$
- Active DHT: In addition to lookups and insertions, the DHT also supports running user-specified code on the $(\text{key}, \text{val})$ pair at node $H(\text{key})$
- Examples: Twitter’s Storm; LinkedIn’s Kafka; Yahoo’s S4 (all open source)
  - Largely subsumes “Streaming MapReduce” and Pregel
Our Contribution

• Bridge the gap between distance oracles and social search. Propose a Scheme that
  – Takes $O^\sim(M)$ time for offline graph processing (uses Das Sarma et al’s oracle)
  – Takes $O^\sim(1)$ time for index operations (i.e. query and update)
  – Can be efficiently implemented on an Active DHT with $O^\sim(C)$ total memory where $C$ is the corpus size, and with $O^\sim(1)$ DHT calls per index operation in the worst case, and two DHT calls in a common case

• Empirical validation
Partitioned Multi-Indexing: Overview

• Maintain a small number (e.g., 100) indexes of real-time content, and a corresponding small number of distance sketches [Hence, ”multi”]

• Each index is partitioned into up to \( \frac{N}{2} \) smaller indexes [Hence, “partitioned”]

• Content indexes can be updated in real-time; Distance sketches are batched

• Real-time efficient querying on Active DHT
Distance Sketch: Overview

• Sample sets $S_i$ of size $N/2^i$ from the set of all nodes $V$, where $i$ ranges from 1 to $\log N$

• For each $S_i$, for each node $v$, compute:
  — The “landmark node” $L_i(v)$ in $S_i$ closest to $v$
  — The distance $D_i(v)$ of $v$ to $L(v)$

• Intuition: if $u$ and $v$ have the same landmark in set $S_i$ then this set witnesses that the distance between $u$ and $v$ is at most $D_i(u) + D_i(v)$, else $S_i$ is useless for the pair $(u,v)$
Distance Sketch: Overview

• Sample sets $S_i$ of size $N/2^i$ from the set of all nodes $V$, where $i$ ranges from 1 to $\log N$

• For each $S_i$, for each node $v$, compute:
  – The “landmark” $L_i(v)$ in $S_i$ closest to $v$
  – The distance $D_i(v)$ of $v$ to $L(v)$

• Intuition: if $u$ and $v$ have the same landmark in set $S_i$ then this set witnesses that the distance between $u$ and $v$ is at most $D_i(u) + D_i(v)$, else $S_i$ is useless for the pair $(u,v)$
Distance Sketch: Overview

• Sample sets $S_i$ of size $N/2^i$ from the set of all nodes $V$, where $i$ ranges from 1 to $\log N$
• For each $S_i$, for each node $v$, compute:
  – The “landmark” $L_i(v)$ in $S_i$ closest to $v$
  – The distance $D_i(v)$ of $v$ to $L(v)$
• Intuition: if $u$ and $v$ have the same landmark in set $S_i$ then this set witnesses that the distance between $u$ and $v$ is at most $D_i(u) + D_i(v)$, else $S_i$ is useless for the pair $(u,v)$
Distance Sketch: Overview

- Sample sets $S_i$ of size $N/2^i$ from the set of all nodes $V$, where $i$ ranges from 1 to $\log N$
- For each $S_i$, for each node $v$, compute:
  - The “landmark” $L_i(v)$ in $S_i$ closest to $v$
  - The distance $D_i(v)$ of $v$ to $L(v)$
- Intuition: if $u$ and $v$ have the same landmark in set $S_i$ then this set witnesses that the distance between $u$ and $v$ is at most $D_i(u) + D_i(v)$, else $S_i$ is useless for the pair $(u,v)$
- Repeat the entire process $O(\log N)$ times for getting good results
Algorithm 1 Distance Sketching Algorithm.

1: Input: Undirected graph $G$, $k \geq 1$, $0 \leq r \leq \log_2 n$

2: Let $h = k(r + 1)$

3: for $i = 0$ to $h - 1$ do

4: Sample, uniformly at random, a subset $S_i \subseteq V$ of size $|S_i| = 2^i \mod (r+1)$.

5: Do a BFS from $S_i$, and compute, for all $u \in V$, $L_i[u] = \arg\min_{x \in S_i} \{d(u, x)\}$, and $D_i[u] = d(u, L_i[u])$.

6: end for

7: $\forall u \in V$, let $E[u] = \langle (L_0[u], D_0[u]), \ldots, (L_{h-1}[u], D_{h-1}[u]) \rangle$. 
Partitioned Multi-Indexing: Overview

- Maintain a priority queue $PMI(i, x, w)$ for every sampled set $S_i$, every node $x$ in $S_i$, and every keyword $w$
- When a keyword $w$ arrives at node $v$, add node $v$ to the queue $PMI(i, L_i(v), w)$ for all sampled sets $S_i$
  - Use $D_i(v)$ as the priority
  - The inserted tuple is $(v, D_i(v))$
- Perform analogous steps for keyword deletion
- Intuition: Maintain a separate index for every $S_i$, partitioned among nodes in $S_i$
Querying: Overview

• If node $u$ queries for keyword $w$, then look for the best result among the top results in exactly one partition of each index $S_i$
  – Look at $PMI(i, L_i(u), w)$
  – If non-empty, look at the top tuple $<v, D_i(v)>$, and return the result $<i, v, D_i(u) + D_i(v)>$

• Choose the tuple $<i, v, D>$ with smallest $D$
Intuition

• Suppose node $u$ queries for keyword $w$, which is present at a node $v$ very close to $u$

  – It is likely that $u$ and $v$ will have the same landmark in a large sampled set $S_i$ and that landmark will be very close to both $u$ and $v$. 
Distributed Implementation

• Sketching easily done on MapReduce
• Indexing operations (updates and search queries) can be implemented on an Active DHT
  • Basic Idea: Shard sketches based on node, and indexes based on word
  • Only 2 network accesses per query/update (assumes that the total index size for the keyword is small enough to fit on one machine; else $O^\sim(1)$)
  • Total network communication almost constant per update or per search result.
Theorems

1. Efficiency: Already specified
Our Contribution (revisited)

- Bridge the gap between distance oracles and social search
  - Propose an algorithm that take $O^\sim(M)$ time for offline graph processing (uses Das Sarma et al.’s oracle)
  - Takes $O^\sim(1)$ time for index operations (i.e. query and update)
  - Can be efficiently implemented on an Active DHT with $O^\sim(C)$ total memory where $C$ is the corpus size, and with $O^\sim(1)$ DHT calls per index operation in the worst case, and two DHT calls per in a common case

- Empirical validation
Theorems

2. Correctness: Suppose
   - Node $v$ issues a query for word $w$
   - There exists a node $x$ with the word $w$

   Then we find a node $y$ which contains $w$ such that, with high probability,
   $$d(v,y) = O(\log N)d(v,x)$$

Builds on Das Sarma et al; much better in practice (typically, $1 + \varepsilon$ rather than $O(\log N)$)
Extensions: Combining with Other Relevance Measures

• Examples of important relevance measures: PageRank, tf-idf, and recency (for real-time search results).

• Rank based on combined score function:

\[ s_{u,\omega}(v) = \lambda d(u, v) + (1 - \lambda)\alpha_v(\omega) \]

• Approximate variant decomposes:

\[ \tilde{s}_{u,\omega}(v) = \lambda \tilde{d}(u, v) + (1 - \lambda)\alpha_v(\omega) \]

and leads to same guarantees.
Extensions: Multiple Results

• Can easily extend scheme to get $J$ results per query
Extensions: Other Distance Measures

- Distance measure only used for computing the distance sketch

- Hence, our scheme extends to any distance measure where $O(1)$ “bfs-like” computations can be performed efficiently offline (eg. on MapReduce)
Experimental Setup

- The twitter network is a sub-sample (~4M nodes); the corpus was bios of users; queries synthesized using random walks to model a user's behavior

- Measures:
  - Number of failed queries, i.e. where none of the top J results is as good as the known target node
  - Average depth of successful queries

<table>
<thead>
<tr>
<th></th>
<th>Undirected</th>
<th>Directed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>Grid</td>
<td>ForestFire</td>
</tr>
<tr>
<td>Real-world</td>
<td>Undirected Twitter</td>
<td>Directed Twitter</td>
</tr>
</tbody>
</table>

Table 1: Networks used in the experiments.
Experimental Results

Fraction of failed queries for undirected networks

Fraction of failed queries for directed networks
Experimental Results

Average depth of the first good result
Experimental Results

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Our Scheme</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Network</td>
<td>58</td>
<td>18</td>
</tr>
<tr>
<td>Undirected Twitter Network</td>
<td>930</td>
<td>71</td>
</tr>
<tr>
<td>ForestFire Network</td>
<td>74</td>
<td>5</td>
</tr>
<tr>
<td>Directed Twitter Network</td>
<td>1384</td>
<td>163</td>
</tr>
</tbody>
</table>

Table 2: Total preprocessing time (sec).

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Our Scheme</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid Network</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>Undirected Twitter Network</td>
<td>1</td>
<td>61</td>
</tr>
<tr>
<td>ForestFire Network</td>
<td>2</td>
<td>44</td>
</tr>
<tr>
<td>Directed Twitter Network</td>
<td>2</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 3: Total query time (sec) over 20000 queries.
Future Work

• Formal analysis over generative models

• Multi-keyword queries

• Other measures of social closeness (eg. PageRank/hitting times)

• Implementation over active DHT (eg. Storm)
  – Partial progress