Using Set Cover to Optimize a Large-Scale Low Latency Distributed Graph

Rui Wang
Staff Software Engineer
Outline

- LinkedIn’s Distributed Graph Infrastructure
- The Scaling Problem
- Set Cover by Example
- Evaluation
- Related Work
- Conclusions and Future Work
Outline

- LinkedIn’s Distributed Graph Infrastructure
- The Scaling Problem
- Set Cover by Example
- Evaluation
- Related Work
- Conclusions and Future Work
LinkedIn Products Backed By Social Graph

LinkedIn

Home Careers Products & Services Insights

400,842 followers

How You're Connected

LinkedIn

Recent Updates

LinkedIn Summertime here at LinkedIn is signaled by the arrival of Food Truck Friday at our Mountain View campus. We like to serve our lunch with a side of vitamin D and community! http://linkd.in/13L8LOD

Food Truck Friday Rolls Back Into LinkedIn for Summer

linkd.in - There are a number of things that signify the beginning of summer - warm weather, longer days, family vacations. At LinkedIn's headquarters in Mountain View, we associate the sights and sounds of summer with the rumble of trucks and the delicious...

Like (83) • Comment (5) • Share • 1 day ago

Careers

Interested in LinkedIn?
Learn about our company and culture.

418 jobs posted

Products And Services
LinkedIn’s Distributed Graph APIs

- **Graph APIs**
  - **Get Connections**
    - “Who does member X know?”
  - **Get Shared Connections**
    - “Who do I know in common with member Y”?
  - **Get Graph Distances**
    - “What’s the graph distances for each member returned from a search query?”
    - “Is member Z within my second degree network so that I can view her profile?”

- Over 50% queries is to get graph distances
## Distributed Graph Architecture Diagram

### Table:

<table>
<thead>
<tr>
<th>Member ID</th>
<th>Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partition N</th>
</tr>
</thead>
<tbody>
<tr>
<td>210,000,000</td>
</tr>
<tr>
<td>210,000,001</td>
</tr>
</tbody>
</table>
LinkedIn’s Distributed Graph Infrastructure

- **Graph Database (Graph DB)**
  - Partitioned and replicated graph database
  - Distributed adjacency list

- **Distributed Network Cache (NCS)**
  - LRU cache stores second degree network for active members
  - Graph traversals are converted to set intersections

- **Graph APIs**
  - Get Connections
  - Get Shared Connections
  - Get Graph Distances
Outline

- LinkedIn’s Distributed Graph Infrastructure
- The Scaling Problem
- Set Cover by Example
- Evaluation
- Related Work
- Conclusions and Future Work
Graph Distance Queries and Second Degree Creation

API
get graph distances

NCS
cache lookup

Graph DB
Partition IDs
K-Way merges
partial merges

[exists=true] set intersections
[exists=false] retrieve 1st degree connections

return
return
The Scaling Problem Illustrated

- Increased Query Distribution
- Merging and De-duping shift to single NCS node
Outline

- LinkedIn’s Distributed Graph Infrastructure
- The Scaling Problem
- Set Cover and Example
- Evaluation
- Related Work
- Conclusions and Future Work
Set Cover Problem

- **Definition**
  - Given a set of elements $U = \{1, 2, ..., m\}$ (called the universe) and a family $S$ of $n$ sets whose union equals the universe, the set cover problem is to identify the smallest subset of $S$ the union of which contains all elements in the universe.

- **Greedy Algorithm**
  - **Rule**
    - At each stage, choose the set that contains the largest number of uncovered elements.
  - **Upper bound**
    - $O(\log s)$, where $s$ is the size of the largest set from $S$.
Modified Set Cover Algorithm Example Cont.

- Build a map of partition ID -> Graph DB nodes for the input graph.
Randomly pick an element from $U = \{1, 2, 5, 6\}$
- $e = 5$

Retrieve from map
- $\text{nodes} = \{R_{21}, R_{13}\}$

Intersect
- $R_{21} = \{1, 5\}$ with $U$
- $R_{13} = \{5, 6\}$ with $U$

Select $R_{21}$
- $U = \{2, 6\}$, $C = \{R_{21}\}$
Randomly pick an element from $\mathcal{U} = \{2, 6\}$
- $e = 2$

Retrieve from map
- $\text{nodes} = \{R_{11}, R_{22}\}$

Intersect
- $R_{11} = \{1, 2\}$ with $\mathcal{U}$,
- $R_{22} = \{2, 4\}$ with $\mathcal{U}$

Select $R_{22}$
- $\mathcal{U} = \{6\}$, $C = \{R_{21}, R_{22}\}$
Modified Set Cover Algorithm Example Cont.

- Pick the final element from $U = \{ 6 \}$
  - $e = 6$

- Retrieve from map
  - $\text{nodes} = \{ R23, R13 \}$

- Intersect
  - $R23 = \{ 3, 6 \}$ with $U$,
  - $R13 = \{ 5, 6 \}$ with $U$

- Select $R23$
  - $U = \{ \}, C = \{ R21, R22, R23 \}$
Modified Set Cover Algorithm Example Solution

- Solution Compared to Optimal Result for $\mathcal{U}$, where $\mathcal{U} = \{1,2,5,6\}$
Outline

- LinkedIn’s Distributed Graph Infrastructure
- The Scaling Problem
- Set Cover by Example
- Evaluation
- Related Work
- Conclusions and Future Work
Evaluation

- Production Results
  - Second degree cache creation drops 38% on 99<sup>th</sup> percentile
  - Graph distance calculation drops 25% on 99<sup>th</sup> percentile
  - Outbound traffic drops 40%; Inbound traffic drops 10%
Outline

- LinkedIn’s Distributed Graph Infrastructure
- The Scaling Problem
- Set Cover by Example
- Evaluation
- Related Work
- Conclusions and Future Work
Related Work

- **Scaling through Replications**
  - Collocating neighbors [Pujol2010]
  - Replication based on read/write frequencies [Mondal2012]
  - Replication based on locality [Carrasco2011]

- **Multi-Core Implementations**
  - Parallel graph exploration [Hong2011]

- **Offline Graph Systems**
  - Google’s Pregel [Malewicz2010]
  - Distributed GraphLab [Low2012]
Outline

- LinkedIn’s Distributed Graph Infrastructure
- The Scaling Problem
- Set Cover by Example
- Evaluation
- Related Work
- Conclusions and Future Work
Conclusions and Future Work

- **Future Work**
  - Incremental cache updates
  - Replication on GraphDB nodes through LRU caching
  - New graph traversal algorithms

- **Conclusions**
  - Key challenges tackled
    - Work distribution balancing
    - Communication Bandwidth
  - Set cover optimized latency for distributed query by
    - Identifying a much smaller set of GraphDB nodes serving queries
    - Shifting workload to GraphDB to utilize parallel powers
    - Alleviating the K-way merging costs on NCS by reducing K
  - Available at