Walking in the Cloud: Parallel SimRank at Scale
Zhenguo Li¹, Yixiang Fang², Qin Liu³, Jiefeng Cheng¹, Reynold Cheng², John C.S. Lui³
¹Huawei Noah’s Ark Lab, ²HKU, ³CUHK
To appear in PVLDB’16

SimRank [1]

- Graph data grows rapidly
  1. Internet of Things
  2. World Wide Web
- Similarity is fundamental
  1. Information retrieval
  2. Recommender system
  3. churn prediction
- SimRank - two objects are similar if referenced by similar objects
  \[
  s(i, j) = \sum_{c} \frac{1}{\text{in}(i) \cap \text{in}(j)} \sum_{a \in \text{in}(i)} s(a, j) - s(i, a) \quad \text{if } j \neq i
  \]
  \[
  s(i, j) = \text{similarity of nodes } i \text{ and } j
  \]
  \[
  m(i) = \text{in-neighbors of } i
  \]
  \[
  c = \text{decay factor, } 0 < c < 1
  \]

- It captures human perception of similarity
- It outperforms other similarity measures, such as co-citation

Three fundamental queries
1. Single-query pair – return similarity of two nodes
2. Single-source query – return similarity of every node to a node
3. All-query pair – return similarity between every two nodes

Challenges in SimRank computation
1. High complexity: \(O(n^2)\) time, \(O(n^3)\) space
2. Heavy computational dependency (hard to be parallelized)
3. Not allow querying similarities individually

CloudWalker – Big SimRank, instant response

- Contribution
  1. Enable parallel SimRank computation
  2. Test on the largest graph, clue-web (\(|V| = 1B, |E| = 43B\))

- Problem
  SimRank Decomposition \(S = cP^TDP + D\)
  \(P:\) the transition matrix on graph
  \(D:\) the diagonal correction matrix to be estimated
  \(S = D + cP^TDP + cP^TP^TD + \ldots\)

  - how to compute \(D\) for big graph?
  - how to query efficiently given \(D\)?

- Offline indexing \(x = [D_{11}, D_{22}, \ldots, D_{nn}]^T\)
  1. Key observation: self-similarity is 1.0
  2. Generate \(a_i^x\) by Monte Carlo simulation, in parallel
  3. Solve the linear system via Jacobi method, in parallel

To compute \(a_w\), we obtain \(P_e^x\), using Monte Carlo Simulation
1. Place \(R\) random walkers on node \(i\)
2. Each walker walks \(t\) steps along in-links
3. Count the distribution of walkers

Online queries

- MCSP: Monte Carlo simulation for single-pair query
  - constant time complexity: \(O(\text{TR})\)
- MCSS: Monte Carlo simulation for single-source query
  - constant time complexity: \(O(T^{1.4}R \log R)\)
- MCAP: Monte Carlo simulation for all-pair query
  - use MCSS repeatedly; time complexity: \(O(nT^{1.4}R \log R)\)

Implementation on Spark

Why Spark?
- General-purpose in-memory cluster computing
- Easy-to-use operations for distributed applications

Two implementation models
- Broadcasting: Graph stored in each machine
- RDD (Resilient Distributed Dataset): Graph stored in an RDD

Experiments

- Setup: cluster, datasets, and default parameters
  - 10 nodes (each with 16 cores, 377GB RAM, 20TB disk)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Size</th>
<th>Parameter</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>wiki-vote</td>
<td>7.1K</td>
<td>103K</td>
<td>476.8KB</td>
<td>c</td>
<td>0.6</td>
<td>decay factor of SimRank</td>
</tr>
<tr>
<td>wiki-talk</td>
<td>2.4M</td>
<td>5M</td>
<td>45.6MB</td>
<td>T</td>
<td>10</td>
<td># of walk steps</td>
</tr>
<tr>
<td>twitter-2010</td>
<td>42M</td>
<td>1.5B</td>
<td>11.4GB</td>
<td>L</td>
<td>3</td>
<td># of iterations in Jacobi method</td>
</tr>
<tr>
<td>uk-union</td>
<td>131M</td>
<td>5.5B</td>
<td>48.3GB</td>
<td>R</td>
<td>100</td>
<td># of walkers in simulating (a_w)</td>
</tr>
<tr>
<td>clue-web</td>
<td>118</td>
<td>42.6B</td>
<td>401.1GB</td>
<td>P</td>
<td>10,000</td>
<td># of walkers in MCSP and MCSS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>D</th>
<th>MCSP</th>
<th>MCSS</th>
<th>RDD</th>
<th>D</th>
<th>MCSP</th>
<th>MCSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>wiki-vote</td>
<td>7s</td>
<td>0.046s</td>
<td>0.042s</td>
<td>50s</td>
<td>2.7s</td>
<td>2.5s</td>
<td></td>
</tr>
<tr>
<td>wiki-talk</td>
<td>59s</td>
<td>0.048s</td>
<td>0.179s</td>
<td>430s</td>
<td>8.5s</td>
<td>13.9s</td>
<td></td>
</tr>
<tr>
<td>twitter-2010</td>
<td>975s</td>
<td>0.049s</td>
<td>0.281s</td>
<td>8424s</td>
<td>11.8s</td>
<td>22.3s</td>
<td></td>
</tr>
<tr>
<td>uk-union</td>
<td>3323s</td>
<td>0.052s</td>
<td>0.292s</td>
<td>6.4h</td>
<td>13.1s</td>
<td>27.2s</td>
<td></td>
</tr>
<tr>
<td>clue-web</td>
<td>3323s</td>
<td>0.052s</td>
<td>0.292s</td>
<td>110.2h</td>
<td>64.0s</td>
<td>188.1s</td>
<td></td>
</tr>
</tbody>
</table>

Effectiveness: CloudWalker converges quickly

Broadcasting is more efficient, but RDD is more scalable

CloudWalker outperforms state of the art

Preprocessing, single-pair and single-source queries

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Prep. SP.</td>
<td>S S</td>
<td>Prep. SP.</td>
<td>S S</td>
</tr>
<tr>
<td>wiki-vote</td>
<td>43.4s</td>
<td>42.5s</td>
<td>187ms</td>
</tr>
<tr>
<td>wiki-talk</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>twitter-2010</td>
<td>-</td>
<td>-</td>
<td>14379s</td>
</tr>
<tr>
<td>uk-union</td>
<td>-</td>
<td>-</td>
<td>8291s</td>
</tr>
<tr>
<td>clue-web</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>