Walking in the Cloud: Parallel SimRank at Scale
Zhengu 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

  \[
  s(i, j) = \sum_{t=0}^{\infty} \theta^t \cdot \sum_{(i, j)c \in \mathbf{A}(i, j)} s(i', j'), \quad i \neq j
  \]

  \[
  s(i, j): \text{similarity nodes } i \text{ and } j
  \]

  \[
  m(i, j): \text{in-neighbors of } i
  \]

  \[
  c: \text{decay factor}, 0 < c < 1
  \]

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

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

- Challenges in SimRank computation
  1. High complexity: \(O(n^4)\) 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
  \[\hat{S} = cP^tDP + cP^tDP + \ldots\]

  1. how to compute \(D\) for big graph?
  2. 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 \(x_i\) by Monte Carlo simulation, in parallel
  3. Solve the linear system via Jacobi method, in parallel

To compute \(a_0\), we obtain \(P^t\), 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: \(O(\mathcal{T}R)\)
- MCSS: Monte Carlo simulation for single-source query
  - constant time complexity: \(O(\mathcal{T}^2R \log d)\)
- MCAP: Monte Carlo simulation for all-pair query
  - use MCSS repeatedly; time complexity: \(O(n\mathcal{T}^2R \log d)\)

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

<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>10K</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 # of walk steps</td>
<td></td>
</tr>
<tr>
<td>twitter-2010</td>
<td>42M</td>
<td>1.5M</td>
<td>11.4GB</td>
<td>(L)</td>
<td>3 # of iterations in Jacobi method</td>
<td></td>
</tr>
<tr>
<td>uk-union</td>
<td>131M</td>
<td>5.5M</td>
<td>48.3GB</td>
<td>(R)</td>
<td>100 # of walkers in simulating (a_0)</td>
<td></td>
</tr>
<tr>
<td>clue-web</td>
<td>18</td>
<td>42.6H</td>
<td>401.1GB</td>
<td>(R')</td>
<td>10^30 # of walkers in MCSP and MCSS</td>
<td></td>
</tr>
</tbody>
</table>

Dataset

<table>
<thead>
<tr>
<th>Dataset</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>
</tr>
<tr>
<td>wiki-talk</td>
<td>2.7s</td>
<td>5.6s</td>
<td>2.5s</td>
</tr>
<tr>
<td>twitter-2010</td>
<td>11.8s</td>
<td>22.3s</td>
<td></td>
</tr>
<tr>
<td>uk-union</td>
<td>6.4g</td>
<td>13.1s</td>
<td>27.2s</td>
</tr>
<tr>
<td>clue-web</td>
<td>110.2h</td>
<td>64.0s</td>
<td>188.1s</td>
</tr>
</tbody>
</table>

Broadcasting

<table>
<thead>
<tr>
<th>Dataset</th>
<th>(D)</th>
<th>MCSP</th>
<th>MCSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>wiki-vote</td>
<td>50s</td>
<td>2.7s</td>
<td>2.5s</td>
</tr>
<tr>
<td>wiki-talk</td>
<td>630s</td>
<td>8.5s</td>
<td>13.9s</td>
</tr>
<tr>
<td>twitter-2010</td>
<td>8424s</td>
<td>22.3s</td>
<td></td>
</tr>
<tr>
<td>uk-union</td>
<td>6.4g</td>
<td>13.1s</td>
<td>27.2s</td>
</tr>
<tr>
<td>clue-web</td>
<td>110.2h</td>
<td>64.0s</td>
<td>188.1s</td>
</tr>
</tbody>
</table>

CloudWalker outperforms state of the art

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wiki-vote</td>
<td>43.4s</td>
<td>30.4ms</td>
<td>42.5s</td>
</tr>
<tr>
<td>wiki-talk</td>
<td>50s</td>
<td>8.5s</td>
<td>13.9s</td>
</tr>
<tr>
<td>twitter-2010</td>
<td>14376s</td>
<td>281ms</td>
<td></td>
</tr>
<tr>
<td>uk-union</td>
<td>-</td>
<td>8291s</td>
<td>291ms</td>
</tr>
<tr>
<td>clue-web</td>
<td>-</td>
<td>-</td>
<td>291ms</td>
</tr>
</tbody>
</table>