<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>PASCO: Parallel SimRank Computation at Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Li, Z; Fang, Y; Liu, Q; Cheng, J; Cheng, RCK; Lui, JCS</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>The 2015 ACM Symposium on Cloud Computing (SoCC 2015), Kohala Coast, HI., 27-29 August 2015.</td>
</tr>
<tr>
<td><strong>Issued Date</strong></td>
<td>2015</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10722/214756">http://hdl.handle.net/10722/214756</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.</td>
</tr>
</tbody>
</table>
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

  \[ s(i, j) = \left\{ \begin{array}{ll}
  c \cdot \sum_{n=1}^{\infty} \left( \frac{d_{ij}}{n} \right)^n & \text{if } i = j, \ i \neq j
  \end{array} \right. \]

  \[ s(i, j): \text{similarity of nodes } i \text{ and } j \]
  \[ m(i): \text{in-neighbors of } i \]
  \[ c: \text{decay factor}, 0 < c < 1 \]

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

- Three fundamental queries
  1. Single-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: \( O(n^3) \) 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 \( S = cP^T D P + D \)
  \( P \): the transition matrix on graph
  \( D \): the diagonal correction matrix to be estimated

  \( S = D + cP^T D P + cP^TD^2P + \ldots \)

  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

  Indexing linear system \( a_i x = \sum_{j=1}^{n} c_{ij} a_j x_j \)

  - Generate \( a_i \) by Monte Carlo simulation, in parallel
  - Solve the linear system via Jacobi method, in parallel

To compute \( a_i \), we obtain \( P_c \) 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: \( O(TR) \)
  - MCSS: Monte Carlo simulation for single-source query
    - constant time complexity: \( O(T^3R \log d) \)
  - MCAP: Monte Carlo simulation for all-pair query
    - use MCSS repeatedly; time complexity: \( O(nT^3R \log d) \)

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

Effectiveness: CloudWalker converges quickly

- Broadcasting is more efficient, but RDD is more scalable

CloudWalker outperforms state of the art