<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

\[
s(i,j) = \frac{1}{|\text{out}(i)|} \sum_{j' \in \text{out}(i)} s(i, j') \quad i \neq j
\]

- **Similarity Propagation**
  - Input graph
  - Node-pair graph

- **Key observation:**
  - 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, \(O(n^2)\) 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
  - \(S = cP^TDP + cP^TD^2P + \ldots\)

- **Offline indexing**
  1. **Key observation:** self-similarity is 1.0
  2. **Indexing linear system** \(a_i, x_i = 1, i = 1, 2, \ldots, n\)

- **To compute \(a_i\), 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: \(O(TR)\)
- MCSS: Monte Carlo simulation for single-source query
  - constant time complexity: \(O(T^2R\log d)\)
- MCAP: Monte Carlo simulation for all-pair query
  - use MCSS repeatedly; time complexity: \(O(nTR^2\log d)\)

---

**Implementation on Spark**

<table>
<thead>
<tr>
<th>Why Spark?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. General-purpose in-memory cluster computing</td>
</tr>
<tr>
<td>2. Easy-to-use operations for distributed applications</td>
</tr>
</tbody>
</table>

**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>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>1.5M</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>5.5M</td>
<td>48.3GB</td>
<td>R</td>
<td>100</td>
<td># of walkers in simulating (a_i)</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>wiki-vote</td>
<td>43.4s</td>
<td>40.4s</td>
<td>34.5s</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>14376s</td>
</tr>
<tr>
<td>uk-union</td>
<td>-</td>
<td>-</td>
<td>3323s</td>
</tr>
<tr>
<td>clue-web</td>
<td>-</td>
<td>-</td>
<td>975s</td>
</tr>
</tbody>
</table>

---

**Challenges in SimRank computation**

1. How to compute \(D\) for big graph?
2. How to query efficiently given \(D\)?

---

**To compute \(a_i\), 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

---

**CloudWalker outperforms state of the art**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>wiki-vote</td>
<td>43.4s</td>
<td>40.4s</td>
<td>34.5s</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>14376s</td>
</tr>
<tr>
<td>uk-union</td>
<td>-</td>
<td>-</td>
<td>3323s</td>
</tr>
<tr>
<td>clue-web</td>
<td>-</td>
<td>-</td>
<td>975s</td>
</tr>
</tbody>
</table>

---

**To compute \(a_i\), 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: \(O(TR)\)
- MCSS: Monte Carlo simulation for single-source query
  - constant time complexity: \(O(T^2R\log d)\)
- MCAP: Monte Carlo simulation for all-pair query
  - use MCSS repeatedly; time complexity: \(O(nTR^2\log d)\)