Graph Processing

Acknowledgement: Arijit Khan, Sameh Elnikety
Big Graphs

Google: > 1 trillion indexed pages

Web Graph

Facebook: > 1.5 billion active users

Social Network

31 billion RDF triples in 2011

Information Network

De Bruijn: $4^k$ nodes ($k = 20, \ldots, 40$)

Biological Network

100M Ratings, 480K Users, 17K Movies

Graphs in Machine Learning
Big-Graph Scales

Acknowledgement: Y. Wu, WSU
Graph Data: Structure + Attributes

LinkedIn

Peter Norvig
Research Director at Google
San Francisco Bay Area | Computer Software

Peter Norvig's Overview
- Current: Engineering Director at Google
- Past: Division Chief, Computational Sciences at NASA
  Head, Computational Sciences Division at NASA Ames
  Chief Scientist at Jumglee
- Education: University of California, Berkeley
  Brown University
- Recommendations: 1 person has recommended Peter
- Connections: 500+ connections
- Websites: Personal Website
  Company Website
  RSS feed

Peter Norvig's Summary
- Programmer, designer, author, and manager in high tech R&D.
- Specialties: internet search, artificial intelligence, natural language processing, machine learning, programming, education
Graph Data: Structure + Attributes

Web Graph: 20 billion web pages + 20KB data-transfer rate = 4

30-35 MB/sec disk = 400 TB

LinkedIn
Page Rank Computation:
Offline Graph Analytics

Acknowledgement: I. Mele, Web Information Retrieval
Page Rank Computation: Offline Graph Analytics

\[ PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|} \]

- \( PR(u) \): Page Rank of node \( u \)
- \( F_u \): Out-neighbors of node \( u \)
- \( B_u \): In-neighbors of node \( u \)

Page Rank Computation:
Offline Graph Analytics

\[
PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}
\]

<table>
<thead>
<tr>
<th>V_1</th>
<th>V_3</th>
<th>V_2</th>
<th>V_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR(V_1)</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR(V_2)</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR(V_3)</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR(V_4)</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page Rank Computation: Offline Graph Analytics

\[ PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|} \]

<table>
<thead>
<tr>
<th></th>
<th>K=0</th>
<th>K=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR(V_1)</td>
<td>0.25</td>
<td>?</td>
</tr>
<tr>
<td>PR(V_2)</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>PR(V_3)</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>PR(V_4)</td>
<td>0.25</td>
<td></td>
</tr>
</tbody>
</table>

Page Rank Computation: Offline Graph Analytics

\[ PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|} \]

<table>
<thead>
<tr>
<th>V</th>
<th>K=0</th>
<th>K=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_1</td>
<td>0.25</td>
<td>?</td>
</tr>
<tr>
<td>V_2</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>V_3</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>V_4</td>
<td>0.25</td>
<td></td>
</tr>
</tbody>
</table>

Page Rank Computation: Offline Graph Analytics

\[ PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|} \]

<table>
<thead>
<tr>
<th></th>
<th>K=0</th>
<th>K=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR(V₁)</td>
<td>0.25</td>
<td>0.37</td>
</tr>
<tr>
<td>PR(V₂)</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>PR(V₃)</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>PR(V₄)</td>
<td>0.25</td>
<td></td>
</tr>
</tbody>
</table>

Page Rank Computation: Offline Graph Analytics

\[ PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|} \]

<table>
<thead>
<tr>
<th></th>
<th>K=0</th>
<th>K=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR(V₁)</td>
<td>0.25</td>
<td>0.37</td>
</tr>
<tr>
<td>PR(V₂)</td>
<td>0.25</td>
<td>0.08</td>
</tr>
<tr>
<td>PR(V₃)</td>
<td>0.25</td>
<td>0.33</td>
</tr>
<tr>
<td>PR(V₄)</td>
<td>0.25</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Page Rank Computation: Offline Graph Analytics

\[ PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|} \]

<table>
<thead>
<tr>
<th></th>
<th>K=0</th>
<th>K=1</th>
<th>K=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR(V_1)</td>
<td>0.25</td>
<td>0.37</td>
<td>0.43</td>
</tr>
<tr>
<td>PR(V_2)</td>
<td>0.25</td>
<td>0.08</td>
<td>0.12</td>
</tr>
<tr>
<td>PR(V_3)</td>
<td>0.25</td>
<td>0.33</td>
<td>0.27</td>
</tr>
<tr>
<td>PR(V_4)</td>
<td>0.25</td>
<td>0.20</td>
<td>0.16</td>
</tr>
</tbody>
</table>

Page Rank Computation: Offline Graph Analytics

\[ PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|} \]

<table>
<thead>
<tr>
<th></th>
<th>K=0</th>
<th>K=1</th>
<th>K=2</th>
<th>K=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR(V₁)</td>
<td>0.25</td>
<td>0.37</td>
<td>0.43</td>
<td>0.35</td>
</tr>
<tr>
<td>PR(V₂)</td>
<td>0.25</td>
<td>0.08</td>
<td>0.12</td>
<td>0.14</td>
</tr>
<tr>
<td>PR(V₃)</td>
<td>0.25</td>
<td>0.33</td>
<td>0.27</td>
<td>0.29</td>
</tr>
<tr>
<td>PR(V₄)</td>
<td>0.25</td>
<td>0.20</td>
<td>0.16</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Page Rank Computation: Offline Graph Analytics

\[
PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}
\]

<table>
<thead>
<tr>
<th></th>
<th>K=0</th>
<th>K=1</th>
<th>K=2</th>
<th>K=3</th>
<th>K=4</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR(V₁)</td>
<td>0.25</td>
<td>0.37</td>
<td>0.43</td>
<td>0.35</td>
<td>0.39</td>
</tr>
<tr>
<td>PR(V₂)</td>
<td>0.25</td>
<td>0.08</td>
<td>0.12</td>
<td>0.14</td>
<td>0.11</td>
</tr>
<tr>
<td>PR(V₃)</td>
<td>0.25</td>
<td>0.33</td>
<td>0.27</td>
<td>0.29</td>
<td>0.29</td>
</tr>
<tr>
<td>PR(V₄)</td>
<td>0.25</td>
<td>0.20</td>
<td>0.16</td>
<td>0.20</td>
<td>0.19</td>
</tr>
</tbody>
</table>

Page Rank Computation: Offline Graph Analytics

\[
PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}
\]

<table>
<thead>
<tr>
<th></th>
<th>K=0</th>
<th>K=1</th>
<th>K=2</th>
<th>K=3</th>
<th>K=4</th>
<th>K=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR(V_1)</td>
<td>0.25</td>
<td>0.37</td>
<td>0.43</td>
<td>0.35</td>
<td>0.39</td>
<td>0.39</td>
</tr>
<tr>
<td>PR(V_2)</td>
<td>0.25</td>
<td>0.08</td>
<td>0.12</td>
<td>0.14</td>
<td>0.11</td>
<td>0.13</td>
</tr>
<tr>
<td>PR(V_3)</td>
<td>0.25</td>
<td>0.33</td>
<td>0.27</td>
<td>0.29</td>
<td>0.29</td>
<td>0.28</td>
</tr>
<tr>
<td>PR(V_4)</td>
<td>0.25</td>
<td>0.20</td>
<td>0.16</td>
<td>0.20</td>
<td>0.19</td>
<td>0.19</td>
</tr>
</tbody>
</table>

Page Rank Computation: Offline Graph Analytics

\[ PR_{k+1}(u) = \frac{\sum_{v \in B_u} PR_k(v)}{|F_v|} \]

<table>
<thead>
<tr>
<th></th>
<th>K=0</th>
<th>K=1</th>
<th>K=2</th>
<th>K=3</th>
<th>K=4</th>
<th>K=5</th>
<th>K=6</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR(V₁)</td>
<td>0.25</td>
<td>0.37</td>
<td>0.43</td>
<td>0.35</td>
<td>0.39</td>
<td>0.39</td>
<td>0.38</td>
</tr>
<tr>
<td>PR(V₂)</td>
<td>0.25</td>
<td>0.08</td>
<td>0.12</td>
<td>0.14</td>
<td>0.11</td>
<td>0.13</td>
<td>0.13</td>
</tr>
<tr>
<td>PR(V₃)</td>
<td>0.25</td>
<td>0.33</td>
<td>0.27</td>
<td>0.29</td>
<td>0.29</td>
<td>0.28</td>
<td>0.28</td>
</tr>
<tr>
<td>PR(V₄)</td>
<td>0.25</td>
<td>0.20</td>
<td>0.16</td>
<td>0.20</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
</tr>
</tbody>
</table>

PageRank over MapReduce

Multiple MapReduce iterations

Each Page Rank Iteration:

- **Input:**
  - \((\text{id}_1, [\text{PR}_t(1), \text{out}_{11}, \text{out}_{12}, ...]),\)
  - \((\text{id}_2, [\text{PR}_t(2), \text{out}_{21}, \text{out}_{22}, ...]),\)
  ... 

- **Output:**
  - \((\text{id}_1, [\text{PR}_{t+1}(1), \text{out}_{11}, \text{out}_{12}, ...]),\)
  - \((\text{id}_2, [\text{PR}_{t+1}(2), \text{out}_{21}, \text{out}_{22}, ...]),\)
  ... 

Iterate until convergence → another MapReduce instance
PageRank over MapReduce (One Iteration)

Map

- **Input:** \((V_1, [0.25, V_2, V_3, V_4])\);
  \((V_2, [0.25, V_3, V_4])\);
  \((V_3, [0.25, V_1])\);
  \((V_4, [0.25, V_1, V_3])\)

- **Output:** \((V_2, 0.25/3), (V_3, 0.25/3), (V_4, 0.25/3), \ldots, (V_1, 0.25/2), (V_3, 0.25/2)\);
  \((V_1, [V_2, V_3, V_4]), (V_2, [V_3, V_4]), (V_3, [V_1]), (V_4, [V_1, V_3])\)
PageRank over MapReduce
(One Iteration)

Map

- Input: $(V_1, [0.25, V_2, V_3, V_4]);$
  $(V_2, [0.25, V_3, V_4]); (V_3, [0.25, V_1]);$
  $(V_4, [0.25, V_1, V_3])$

- Output: $(V_2, 0.25/3), (V_3, 0.25/3), (V_4, 0.25/3),$
  $\ldots, (V_1, 0.25/2), (V_3, 0.25/2);$
  $(V_1, [V_2, V_3, V_4]), (V_2, [V_3, V_4]), (V_3, [V_1]), (V_4, [V_1, V_3])$
PageRank over MapReduce (One Iteration)

Map

- Input: \((V_1, [0.25, V_2, V_3, V_4])\);
  \((V_2, [0.25, V_3, V_4])\); \((V_3, [0.25, V_1])\);
  \((V_4, [0.25, V_1, V_3])\)

- Output: \((V_2, 0.25/3), (V_3, 0.25/3), (V_4, 0.25/3), \ldots, (V_1, 0.25/2), (V_3, 0.25/2)\);
  \((V_1, [V_2, V_3, V_4]), (V_2, [V_3, V_4]), (V_3, [V_1]), (V_4, [V_1, V_3])\)

Shuffle

- Output: \((V_1, 0.25/1), (V_1, 0.25/2), (V_1, [V_2, V_3, V_4]); \ldots ; (V_4, 0.25/3), (V_4, 0.25/2), (V_4, [V_1, V_3])\)
PageRank over MapReduce (One Iteration)

Map

- **Input:** \((V_1, [0.25, V_2, V_3, V_4]); (V_2, [0.25, V_3, V_4]); (V_3, [0.25, V_1]); (V_4, [0.25, V_1, V_3])\)
- **Output:** \((V_2, 0.25/3), (V_3, 0.25/3), (V_4, 0.25/3), \ldots, (V_1, 0.25/2), (V_3, 0.25/2); (V_1, [V_2, V_3, V_4]), (V_2, [V_3, V_4]), (V_3, [V_1]), (V_4, [V_1, V_3])\)

Shuffle

- **Output:** \((V_1, 0.25/1), (V_1, 0.25/2), (V_1, [V_2, V_3, V_4]); \ldots; (V_4, 0.25/3), (V_4, 0.25/2), (V_4, [V_1, V_3])\)

Reduce

- **Output:** \((V_1, [0.37, V_2, V_3, V_4]); (V_2, [0.08, V_3, V_4]); (V_3, [0.33, V_1]); (V_4, [0.20, V_1, V_3])\)
Key Insight in Parallelization (Page Rank over MapReduce)

- The ‘future’ Page Rank values depend on ‘current’ Page Rank values, but not on any other ‘future’ Page Rank values.

- ‘Future’ Page Rank value of each node can be computed in parallel.
Problems with MapReduce for Graph Analytics

- MapReduce does not directly support iterative algorithms

  - Invariant graph-topology-data re-loaded and re-processed at each iteration \( \rightarrow \) wasting I/O, network bandwidth, and CPU

    Each Page Rank Iteration:
    
    **Input:**
    
    \((id_1, [PR_t(1), out_{11}, out_{12}, \ldots ]), (id_2, [PR_t(2), out_{21}, out_{22}, \ldots ]), \ldots\)
    
    **Output:**
    
    \((id_1, [PR_{t+1}(1), out_{11}, out_{12}, \ldots ]), (id_2, [PR_{t+1}(2), out_{21}, out_{22}, \ldots ]), \ldots\)

  - Materializations of intermediate results at every MapReduce iteration harm performance

  - Extra MapReduce job on each iteration for detecting if a fixpoint has been reached
Alternative to Simple MapReduce for Graph Analytics

- **HALOOP** [Y. Bu et. al., VLDB ‘10]
- **TWISTER** [J. Ekanayake et. al., HPDC ‘10]
- **Piccolo** [R. Power et. al., OSDI ‘10]
- **SPARK** [M. Zaharia et. al., HotCloud ‘10]
- **PREGEL** [G. Malewicz et. al., SIGMOD ‘10]
- **GBASE** [U. Kang et. al., KDD ‘11]

**Iterative Dataflow-based Solutions:**
- **Stratosphere** [Ewen et. al., VLDB ‘12]; **GraphX** [R. Xin et. al., GRADES ‘13]; **Naiad** [D. Murray et. al., SOSP’13]

**DataLog-based Solutions:**
- **SociaLite** [J. Seo et. al., VLDB ‘13]
Alternative to Simple MapReduce for Graph Analytics

- **HALOOP** [Y. Bu et. al., VLDB ‘10]
- **TWISTER** [J. Ekanayake et. al., HPDC ‘10]
- **Piccolo** [R. Power et. al., OSDI ‘10]
- **SPARK** [M. Zaharia et. al., HotCloud ‘10]
- **PREGEL** [G. Malewicz et. al., SIGMOD ‘10]
- **GBASE** [U. Kang et. al., KDD ‘11]

**Bulk Synchronous Parallel (BSP) Computation**

- Dataflow-based Solutions: **Stratosphere** [Ewen et. al., VLDB ‘12]; **GraphX** [R. Xin et. al., GRADES ‘13]; **Naiad** [D. Murray et. al., SOSP’13]

- DataLog-based Solutions: **SociaLite** [J. Seo et. al., VLDB ‘13]
PREGEL

- Inspired by Valiant’s Bulk Synchronous Parallel (BSP) model
- Communication through message passing (usually sent along the outgoing edges from each vertex) + Shared-Nothing
- Vertex centric computation

G. Malewicz et. al., “Pregel: A System for Large-Scale Graph Processing”, SIGMOD ‘10
PREGEL

- Inspired by Valiant’s Bulk Synchronous Parallel (BSP) model
- Communication through message passing (usually sent along the outgoing edges from each vertex) + Shared-Nothing
- Vertex centric computation

Each vertex:
- Receives messages sent in the previous superstep
- Executes the same user-defined function
- Modifies its value
- If active, sends messages to other vertices (received in the next superstep)
- Votes to halt if it has no further work to do \( \rightarrow \) becomes inactive

Terminate when all vertices are inactive and no messages in transmission
PREGEL

Input

Output

PREGEL Computation Model

Active

Inactive

Votes to Halt

Message Received

Computation

Communication

Superstep

Synchronization

State Machine for a Vertex in PREGEL
PREGEL System Architecture

- Master-Slave architecture

Acknowledgement: G. Malewicz, Google
PageRank in Giraph (Pregel)

Suppose: PageRank = 0.15/NUM_VERTICES + 0.85 * SUM

```java
public void compute(Iterator<DoubleWritable> msgIterator) {
    double sum = 0;
    while (msgIterator.hasNext())
        sum += msgIterator.next().get();
    DoubleWritable vertexValue =
        new DoubleWritable(0.15/NUM_VERTICES + 0.85 * sum);
    setVertexValue(vertexValue);
    if (getSuperstep() < MAX_STEPS) {
        long edges = getOutEdgeMap().size();
        sentMsgToAllEdges(
            new DoubleWritable(getVertexValue().get() / edges));
    } else voteToHalt();
}
```

http://incubator.apache.org/giraph/
Page Rank with PREGEL

\[ PR = \frac{0.15}{5} + 0.85 \times \text{SUM} \]

Superstep = 0
Page Rank with PREGEL

PR = 0.15/5 + 0.85 * SUM

Superstep = 1
Page Rank with PREGEL

\[ PR = \frac{0.15}{5} + 0.85 \times \text{SUM} \]
Page Rank with PREGEL

PR = 0.15/5 + 0.85 * SUM

Computation converged

Superstep = 3
Page Rank with PREGEL

PR = \(0.15/5 + 0.85 \times \text{SUM}\)

Superstep = 4
Page Rank with PREGEL

\[ PR = \frac{0.15}{5} + 0.85 \times \text{SUM} \]

Superstep = 5
Benefits of PREGEL over MapReduce

MapReduce

- Requires passing of entire graph topology from one iteration to the next
- Intermediate results after every iteration is stored at disk and then read again from the disk
- Programmer needs to write a driver program to support iterations; another MapReduce program to check for fixpoint

PREGEL

- Each node sends its state only to its neighbors.
- Graph topology information is not passed across iterations
- Main memory based (20X faster for k-core decomposition problem; B. Elser et. al., IEEE BigData ’13)
- Usage of supersteps and master-client architecture makes programming easy
Graph Algorithms Implemented with PREGEL (and PREGEL-Like-Systems)

- Page Rank
- Triangle Counting
- Connected Components
- Shortest Distance
- Random Walk
- Graph Coarsening
- Graph Coloring
- Minimum Spanning Forest
- Community Detection
- Collaborative Filtering
- Belief Propagation
- Named Entity Recognition

Not an Exclusive List
Disadvantages of PREGEL

- In Bulk Synchronous Parallel (BSP) model, performance is limited by the slowest machine
  - Real-world graphs have power-law degree distribution, which may lead to a few highly-loaded servers
BSP Programming Model and its Variants: Offline Graph Analytics

- **PREGEL** [G. Malewicz et. al., SIGMOD ‘10]
- **GPS** [S. Salihoglu et. al., SSDBM ‘13]
- **X-Stream** [A. Roy et. al., SOSP ‘13]
- **GraphLab/ PowerGraph** [Y. Low et. al., VLDB ‘12]
- **Grace** [G. Wang et. al., CIDR ‘13]
- **SIGNAL/COLLECT** [P. Stutz et. al., ISWC ‘10]
- **Giraph++** [Tian et. al., VLDB ‘13]
- **GraphChi** [A. Kyrola et. al., OSDI ‘12]
- **Asynchronous Accumulative Update** [Y. Zhang et. al., ScienceCloud ‘12], **PrIter** [Y. Zhang et. al., SOCC ‘11]

- Synchronous
- Asynchronous