RINGO: A System for Interactive Graph Analytics

Jure Leskovec (@jure)
Including joint work with Y. Perez, R. Sosič, A. Banarjee, M. Raison, R. Puttagunta, P. Shah
My research group at Stanford:

- Mining and modeling large social and information networks
- Problems motivated by the Web and on-line media, large scale data
- We work with networks from FB, Yahoo, Twitter, LinkedIn, Wikimedia, StackOverflow

This talk: Feedback appreciated!
Experts on StackOverflow

12/14/2014
Jure Leskovec (@jure), Stanford University
Observation

Graphs are almost never given but they have to be constructed from input data!
(graph constructions is a part of modeling/discovery process)

Examples:

- **Facebook graphs**: Friend, Communication, Poke, Co-Tag, Co-location, Co-Event
- **Cellphone/Email graphs**: How many calls?
- **Biology**: P2P, Gene interaction networks
Graph Analytics Workflow

Big data storage → Hadoop MapReduce → Relational tables → Graph construction operations → Graph analytics → Graphs and networks
Desiderata for Graph Analytics

Easy to use front-end
- Common high-level programming language

Fast execution times
- Interactive use (as opposed to batch use)

Ability to process large networks
- Tens of billions of edges

Support for several representations
- Transformations between tables and graphs

Large number of graph algorithms
- Straightforward to use

Workflow management and reproducibility
- Provenance
Two observations:

(1) Most graphs are not that large

(2) Big-memory machines are here:
4x Intel CPU, 64 cores, 1TB RAM, $35K

<table>
<thead>
<tr>
<th>Number of Edges</th>
<th>Number of Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0.1M</td>
<td>16</td>
</tr>
<tr>
<td>0.1M – 1M</td>
<td>25</td>
</tr>
<tr>
<td>1M – 10M</td>
<td>17</td>
</tr>
<tr>
<td>10M – 100M</td>
<td>7</td>
</tr>
<tr>
<td>100M – 1B</td>
<td>5</td>
</tr>
<tr>
<td>&gt; 1B</td>
<td>1</td>
</tr>
</tbody>
</table>

Stanford Large Network Collection
71 graphs
## Trade-offs

<table>
<thead>
<tr>
<th>Option 1</th>
<th>Option 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard SQL database</td>
<td>Custom representations</td>
</tr>
<tr>
<td>Separate systems for tables and graphs</td>
<td>Integrated system for tables and graphs</td>
</tr>
<tr>
<td>Single representation for tables and graphs</td>
<td>Separate table and graph representations</td>
</tr>
<tr>
<td>Distributed system</td>
<td>Single machine system</td>
</tr>
<tr>
<td>Disk based structures</td>
<td>In-memory structures</td>
</tr>
<tr>
<td>Options</td>
<td>Option 1</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Standard SQL database</td>
<td>Custom representations</td>
</tr>
<tr>
<td>Separate systems for tables and graphs</td>
<td>Integrated system for tables and graphs</td>
</tr>
<tr>
<td>Single representation for tables and graphs</td>
<td>Separate table and graph representations</td>
</tr>
<tr>
<td>Distributed system</td>
<td>Single machine system</td>
</tr>
<tr>
<td>Disk based structures</td>
<td>In-memory structures</td>
</tr>
</tbody>
</table>
Graph Analytics: Ringo

(a) Specify entities

(b) Specify relationships

(c) Optimize representation

(d) Perform analytic reasoning and inference

(e) Integrate the results

Unstructured data

Relational tables

Tabular networks

Network representation

Results

Ringo

Jure Leskovec (@jure), Stanford University
Ringo! (Python) code for executing finding the StackOverflow example

```python
P = ringo.LoadTable(schema,'posts.tsv')
JP = ringo.Select(P,'Tag=Java')
Q = ringo.Select(JP,'Type=question')
A = ringo.Select(JP,'Type=answer')
QA = ringoJoin(Q,A,'AnswerId','PostId')
G = ringo.ToGraph(QA,'UserId.1','UserId.2')
PR = ringo.GetPageRank(G)
S = ringo.ToTable(PR,'UserId','Score')
ringo.Save(S,'scores.bin')
```
Ringo Overview

High-Level Language User Front-End

- Interface with Graph Processing Engine
- Metadata (Provenance)
- Provenance Script

In-memory Graph Processing Engine

- Filters
- Graph Methods
- Graph Containers
- Graph, Table Conversions
- Table Objects
- Secondary Storage
Input data must be manipulated and transformed into graphs

Table data structure

<table>
<thead>
<tr>
<th>Src</th>
<th>Dst</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>v2</td>
<td></td>
</tr>
<tr>
<td>v2</td>
<td>v3</td>
<td></td>
</tr>
<tr>
<td>v3</td>
<td>v4</td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>v3</td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>v4</td>
<td></td>
</tr>
</tbody>
</table>

Graph data structure
Creating a Graph in Ringo

Four ways to create a graph:

- The data already contains edges as source and destination pairs
- Nodes are connected based on their similarity
- Nodes are connected based on a temporal order
- Nodes are connected based on grouping and aggregation
Use case: In a forum, connect users that post to similar topics:

- Distance metrics
  - Euclidean, Haversine, Jaccard distance
- Connect similar nodes
  - SimJoin, connect if closer than the threshold
- Quadratic complexity
  - Locality sensitive hashing
Use case: In a Web log, connect pages in a temporal order as clicked by the users

- Connect a node with its successors
  - Events selected per user, ordered by timestamps
  - $NextK$, connect $K$ successors
Use case: In a Web log, measure the activity level of different user groups

- Edge creation
  - Partition users to groups
  - Identify interactions within each group
  - Compute a score for each group based on interactions
- Treat groups as super-nodes in a graph
Several graph containers are supported
Over 200 graph algorithms (provided via SNAP)
Graph Representation

Requirements:

- Fast processing
  - Efficient traversal of nodes and edges

- Dynamic structure
  - Quickly add/remove nodes and edges
    - Create subgraphs, dynamic graphs, ...

- How to achieve good balance?
Undirected Graph in Ringo

Nodes table

Sorted vector of neighbors

1
3
6
4

\[ \cdots \]

\[ \cdots \]

\[ \cdots \]
Directed Graph in Ringo

Nodes table

Sorted vectors of in and out neighbors

1
3
6
4

...
Multigraph in Ringo

Nodes table

Sorted vectors of in and out edges

Edges table

1
3
6
4

2
3
7
1
8
5
9
Tables in Ringo

- **Column based store**
  - Persistent row identifier
  - **Lots of tricks:** Implicit select, String pools

- **Standard table operations**
  - Select, join, project, group, aggregate

- **Graph construction operations**
  - Explicit pairs
  - Distance based edges, SimJoin
  - Time based edges, NextK
Convert Tables to Graphs

<table>
<thead>
<tr>
<th>Src</th>
<th>…</th>
<th>Dst</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>…</td>
<td>v2</td>
<td>…</td>
</tr>
<tr>
<td>v2</td>
<td>…</td>
<td>v3</td>
<td>…</td>
</tr>
<tr>
<td>v3</td>
<td>…</td>
<td>v4</td>
<td>…</td>
</tr>
<tr>
<td>v1</td>
<td>…</td>
<td>v3</td>
<td>…</td>
</tr>
<tr>
<td>v1</td>
<td>…</td>
<td>v4</td>
<td>…</td>
</tr>
</tbody>
</table>

- v1 → v2 → v3 → v4
- v2 → v3 → v4
- v1 → v3
- v1 → v4
Convert Tables to Graphs

- Make two copies of pairs of \((Src, Dst)\) columns (Parallel)
- Sort first copy by \(Src\), second copy by \(Dst\) (Parallel)
- Merge column \(Src\) from the first copy and \(Dst\) from the second copy (Sequential)
- Count the number of unique values in the merged column to get the number of nodes in the graph (S)
- Allocate the node hash table
- For each node, build vectors of neighbors (Parallel)
Ringo Implementation

- High-level front end
  - Python module
  - Based on Snap.py, uses SWIG for C++ interface

- High-performance graph engine
  - C++ based on SNAP

- Multi-core support
  - OpenMP to parallelize loops
  - Fast, concurrent hash table, vector operations
## Experiments: Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LiveJournal</th>
<th>Twitter2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>4.8M</td>
<td>42M</td>
</tr>
<tr>
<td>Edges</td>
<td>69M</td>
<td>1.5B</td>
</tr>
<tr>
<td>Text Size</td>
<td>1.1GB</td>
<td>26.2GB</td>
</tr>
<tr>
<td>Graph Size</td>
<td>0.7GB</td>
<td>13.2GB</td>
</tr>
<tr>
<td>Table Size</td>
<td>1.1GB</td>
<td>23.5GB</td>
</tr>
</tbody>
</table>
## Benchmarks, One Computer

<table>
<thead>
<tr>
<th>Algorithm Graph</th>
<th>PageRank LiveJournal</th>
<th>PageRank Twitter2010</th>
<th>Triangles LiveJournal</th>
<th>Triangles Twitter2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Giraph</td>
<td>45.6s</td>
<td>439.3s</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>GraphX</td>
<td>56.0s</td>
<td>-</td>
<td>67.6s</td>
<td>-</td>
</tr>
<tr>
<td>GraphChi</td>
<td>54.0s</td>
<td>595.3s</td>
<td>66.5s</td>
<td>-</td>
</tr>
<tr>
<td>PowerGraph</td>
<td>27.5s</td>
<td>251.7s</td>
<td>5.4s</td>
<td>706.8s</td>
</tr>
<tr>
<td>Ringo</td>
<td>2.6s</td>
<td>72.0s</td>
<td>13.7s</td>
<td>284.1s</td>
</tr>
</tbody>
</table>

Hardware: 4x Intel CPU, 64 cores, 1TB RAM, $35K
## Benchmarks, Published

<table>
<thead>
<tr>
<th>System</th>
<th>Hosts</th>
<th>CPUs host</th>
<th>Host Configuration</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphChi</td>
<td>1</td>
<td>4</td>
<td>8x core AMD, 64GB RAM</td>
<td>158s</td>
</tr>
<tr>
<td>TurboGraph</td>
<td>1</td>
<td>1</td>
<td>6x core Intel, 12GB RAM</td>
<td>30s</td>
</tr>
<tr>
<td>Spark</td>
<td>50</td>
<td>2</td>
<td></td>
<td>97s</td>
</tr>
<tr>
<td>GraphX</td>
<td>16</td>
<td>1</td>
<td>8X core Intel, 68GB RAM</td>
<td>15s</td>
</tr>
<tr>
<td>PowerGraph</td>
<td>64</td>
<td>2</td>
<td>8x hyper Intel, 23GB RAM</td>
<td>3.6s</td>
</tr>
<tr>
<td>Ringo</td>
<td>1</td>
<td>4</td>
<td>20x hyper Intel, 1TB RAM</td>
<td>6.0s</td>
</tr>
</tbody>
</table>

Twitter2010, one iteration of PageRank
## Tables and Graphs

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LiveJournal</th>
<th>Twitter2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table to graph</td>
<td>8.5s, 13.0 MEdges/s</td>
<td>81.0s, 18.0 MEdges/s</td>
</tr>
<tr>
<td>Graph to table</td>
<td>1.5s, 46.0 MEdges/s</td>
<td>29.2s, 50.4 MEdges/s</td>
</tr>
</tbody>
</table>

Hardware: 4x Intel CPU, 80 cores, 1TB RAM, $35K
## Table Operations

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LiveJournal</th>
<th>Twitter2010</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Select 10K</strong></td>
<td>&lt;0.2s</td>
<td>1.6s</td>
</tr>
<tr>
<td></td>
<td>405.9 MRows/s</td>
<td>935.3 MRows/s</td>
</tr>
<tr>
<td><strong>Select All-10K</strong></td>
<td>&lt;0.1s</td>
<td>1.6s</td>
</tr>
<tr>
<td></td>
<td>575.0 MRows/s</td>
<td>917.7 MRows/s</td>
</tr>
<tr>
<td><strong>Join 10K</strong></td>
<td>0.6s</td>
<td>4.2s</td>
</tr>
<tr>
<td></td>
<td>109.5 MRows/s</td>
<td>348.8 MRows/s</td>
</tr>
<tr>
<td><strong>Join All-10K</strong></td>
<td>3.1s</td>
<td>29.7s</td>
</tr>
<tr>
<td></td>
<td>44.5 MRows/s</td>
<td>98.8 MRows/s</td>
</tr>
</tbody>
</table>
## Sequential Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-core</td>
<td>31.0s</td>
</tr>
<tr>
<td>Single source shortest path</td>
<td>7.4s</td>
</tr>
<tr>
<td>Strongly connected component</td>
<td>18.0s</td>
</tr>
</tbody>
</table>

LiveJournal, 1 core
## Load and Save Performance

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LiveJournal</th>
<th>Twitter2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load graph</td>
<td>5.2s</td>
<td>76.6s</td>
</tr>
<tr>
<td>Save graph</td>
<td>3.5s</td>
<td>69.0s</td>
</tr>
<tr>
<td>Load table</td>
<td>6.0s</td>
<td>114.3s</td>
</tr>
<tr>
<td>Save table</td>
<td>4.3s</td>
<td>100.2s</td>
</tr>
</tbody>
</table>
Big-memory machines are here:

- 1TB RAM, 100 Cores ≈ a small cluster
- No overheads of distributed systems
- Easy to program

Most “useful” datasets fit in memory

Big-memory machines present a viable solution for analysis of all-but-the-largest networks
Ringo: Network science & exploration

- In-memory graph analytics
- Processing of tables and graphs
- Fast and scalable
Get your own 1TB RAM server!

And download RINGO/SNAP
http://snap.stanford.edu/snap
THANKS!

@jure
http://snap.stanford.edu