Graph Analytics in the Big Data Era

Yongming Luo, dr. George H.L. Fletcher

Web Engineering Group
What is really hot?
An old/new data model – graph data

- Model entities and relations between entities
- Trending application space
  - Social network analysis (facebook, linkedin, …)
  - Bioinformatics (protein networks)
  - Recommendation (web graph, netflix, …)
  - Semantic Web (RDF data, Google knowledge graph)
RDF as a data model for graph data

• RDF Model (Resource Description Framework)
  • Describe “things (resources)” on the web
  • Part of the linked data vision, connect information on the web
  • Distributed way of managing things
  • Schema-less feature
  • We will skip the strict format description for now
RDF Graph Model and Format

- Directed graph
- Triple format (subject, predicate, object)
- *Subjects* and *objects* are nodes/things in graph
- *Predicates* are edges
- (node, edge, node) format, nothing special

- No distinction between data and metadata
- Predicates can also be resources
Example of Triples

\[
\begin{align*}
\langle Alice \rangle & \langle isFriendOf \rangle \langle Bob \rangle \\
\langle Alice \rangle & \langle like \rangle \langle wine \rangle \\
\langle Alice \rangle & \langle graduateFrom \rangle \langle TU/e \rangle \\
\langle Bob \rangle & \langle like \rangle \langle beer \rangle \\
\langle Bob \rangle & \langle graduateFrom \rangle \langle TU/e \rangle
\end{align*}
\]
Example of Triples, expanded

- Alice isFriendOf Bob
- Alice like wine
- Alice graduateFrom TU/e
- Bob like beer
- Bob graduateFrom TU/e
- like isA feeling
- isFriendOf isA relation
Data Explosion in the (RDF) world

• How much data are we talking about?
  • Simply type filetype:rdf in google, 28M documents found

• Billion Triple Challenge at ISWC
• Easily reach a Trillion triples in commercial systems
• More data join the open data project to connect datasets, and the famous graph (next page)
Graph analytics

• What to do with graphs?
  • Pattern matching
    – graph query languages
  • Graph algorithms
    – Classical algorithms
    – PageRank-style algorithms
Pattern Matching - SPARQL Query for RDF

- The query language for RDF data
- Similar to SQL for relational database
- Similar grammar, select, where, filter clauses and more
- Example

```sparql
select ?a ?b where {
  ?a <isFriendOf> ?b .
  ?a <graduateFrom> ?c .
  ?b <graduateFrom> ?c .
}
```
SPARQL Example

```sparql
select ?a ?b where {
  ?a <isFriendOf> ?b .
  ?a <graduateFrom> ?c .
  ?b <graduateFrom> ?c .
}
```

Two people who are friends and graduate from the same university.

Alice

Bob

wine like

beer like

TU/e

graduateFrom

graduateFrom

isFriendOf
SPARQL Query can be Complex

```sparql
SELECT ?gn ?fn WHERE {
  ?fn <familyNameOf> ?p.
  ?p <type> "scientist";
  <bornInLocation> ?city;
  <hasDoctoralAdvisor> ?a.
  ?city <locatedIn> "Switzerland".
  ?city2 <locatedIn> "Germany".
}
```

rdf3x / Yago / A1.sparql
A few more words on query language

- Essentially ad-hoc graph algorithm execution
- Pattern matching as the backbone, with possibly many features added
  - E.g., regular expression, keyword search, aggregation
- Some special cases get special treatment
  - Triangle counting – community detection, graph measurement
Graph Algorithms

- Breadth First Search
- Single Source Shortest Path
- Bipartite Matching
- PageRank, SimRank and more
  - So called diffusion based techniques
Where are we

- Graph model
- Operations on graph
  - Pattern matching (queries) -- indexes
  - Algorithms -- platforms
Indexes to accelerate query processing

- Value-based indexes
- Relational approaches
- Document-oriented approaches
- Structural indexes
- Seeqr
- Frequent patterns, …

Relational approaches, RDF as an example

- Treat triples as a three-column table
- Relational DB, row store, column store

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Alice&gt;</td>
<td>&lt;isFriendOf&gt;</td>
<td>&lt;Bob&gt;</td>
</tr>
<tr>
<td>&lt;Alice&gt;</td>
<td>&lt;like&gt;</td>
<td>&lt;wine&gt;</td>
</tr>
<tr>
<td>&lt;Alice&gt;</td>
<td>&lt;graduateFrom&gt;</td>
<td>&lt;TU/e&gt;</td>
</tr>
<tr>
<td>&lt;Bob&gt;</td>
<td>&lt;like&gt;</td>
<td>&lt;beer&gt;</td>
</tr>
<tr>
<td>&lt;Bob&gt;</td>
<td>&lt;graduateFrom&gt;</td>
<td>&lt;TU/e&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- One entity one row
Relational Approaches – Query Processing

- For relational DB
- SPARQL -> SQL

```sql
select ?a ?b where {
  ?a <isFriendOf> ?b .
  ?a <graduateFrom> ?c .
  ?b <graduateFrom> ?c .
}
```

```sql
select T1.subject, T2.subject
from example T1
  inner join example T2
  inner join example T3
  on T1.object = T2.object
  AND T1.subject = T3.subject
  AND T2.subject = T3.object
where T1.predicate = 'graduateFrom'
  AND T2.predicate = 'graduateFrom'
  AND T3.predicate = 'isFriendOf';
```
select T1.subject, T2.subject
from example T1
inner join example T2
inner join example T3
on T1.object = T2.object
AND T1.subject = T3.subject
AND T2.subject = T3.object
where T1.predicate = 'graduateFrom'
AND T2.predicate = 'graduateFrom'
AND T3.predicate = 'isFriendOf'
Relational DB is not enough

- **Problems**
  - Reduce data size -> ID mapping

<table>
<thead>
<tr>
<th>String</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Alice&gt;</td>
<td>1</td>
</tr>
<tr>
<td>&lt;Bob&gt;</td>
<td>2</td>
</tr>
<tr>
<td>&lt;like&gt;</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- How to build indexes?
- Too many self-joins
- Several alternative SQLs, which one to choose?
Native Approaches - RDF-3X

- **B-Tree index**
- **Index several (if not all) permutations of triples**
  - \((s, p, o) \quad (p, s, o) \quad (o, s, p) \quad (s, o, p) \ldots\)
- **prefer merge join rather than hash join**

\[
\text{select } ?a \ ?b \text{ where } \\
\quad ?a \ <\text{isFriendOf}> \ ?b . \\
\quad ?a \ <\text{graduateFrom}> \ ?c . \\
\quad ?b \ <\text{graduateFrom}> \ ?c . \\
\]
One possible plan from RDF-3X

- **HashJoin**: $(v1=v5)$
  - **MergeJoin**: $(v2=v3)$
    - **IndexScan**: $(v1, \text{isFriendOf}, v2)$
    - **IndexScan**: $(v3, \text{graduateFrom}, v4)$
  - **IndexScan**: $(v5, \text{graduateFrom}, v6)$
  - **Filter**: $(v4=v6)$
More techniques

• Query optimization
  • Statistics, query history, cache, all information helps
• Compression
  • Standard compression techniques
  • Works better for column store
• Update
  • Save changes first, merge them later
One Entity One Row

- In the spirit of E-R model, or adjacency list of graphs
- Reduce self-joins

<table>
<thead>
<tr>
<th>Entity</th>
<th>isFriendOf</th>
<th>like</th>
<th>graduateFrom</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Alice&gt;</td>
<td>&lt;Bob&gt;</td>
<td>&lt;wine&gt;</td>
<td>&lt;TU/e&gt;</td>
</tr>
<tr>
<td>&lt;Bob&gt;</td>
<td></td>
<td>&lt;beer&gt;</td>
<td>&lt;TU/e&gt;</td>
</tr>
<tr>
<td>&lt;wine&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;beer&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;TU/e&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
One Entity One Row - Cons

- Null values
- Multi-value property
- Too many properties
- Schema required
- Schema update

<table>
<thead>
<tr>
<th>Entity</th>
<th>isFriendOf</th>
<th>like</th>
<th>graduateFrom</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Alice&gt;</td>
<td>&lt;Bob&gt;</td>
<td>&lt;wine&gt;</td>
<td>&lt;TU/e&gt;</td>
</tr>
<tr>
<td>&lt;Bob&gt;</td>
<td></td>
<td>&lt;beer&gt;</td>
<td>&lt;TU/e&gt;</td>
</tr>
<tr>
<td>&lt;wine&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;beer&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;TU/e&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Structural Index - Motivation

- Value indexes are good, but there is more regularity/structure we can get from data
- Query/index mismatch
Structural Index

- Preserve structures in index
- Give the queries what they are looking for, not less, not more
It will be better if we have something like
select ?a ?b where {
  ?a <isFriendOf>   ?b .
  ?a <graduateFrom> ?c .
  ?b <graduateFrom> ?c .
}

If two triples share a subject
Draw an SS edge between them
Same for SO, OS, SP, PS, …
Ideal case

?a <isFriendOf> ?b

SS

?a <graduateFrom> ?c

OS

?b <graduateFrom> ?c

PP,OO

partition1

SS

partition2

OS

PP,OO

partition3
How to build partition blocks (structural index)?

- E.g., according to k-bisimulation, denoted $\approx^k$

$x \approx^k y$ when the following holds:
- $x \approx^0 y$ if the node labels of $x$ and $y$ are the same
- If $x \rightarrow x'$, then there is some $y \rightarrow y'$, such that $x' \approx^{k-1} y'$
- If $y \rightarrow y'$, then there is some $x \rightarrow x'$, such that $x' \approx^{k-1} y'$
Algorithm for k-bisimulation computation

- Create a signature for each node in the graph
- Nodes are partitioned by their signature values
Where are we

• Graph model
• Operations on graph
  • Pattern matching (queries) -- indexes
  • Algorithms -- platforms
Platforms for (graph) algorithms

- Single machine, out-of-core
  - I/O-efficient algorithms
  - GraphChi
- Shared-nothing architecture
  - MapReduce (Hadoop)
  - Pregel/GraphLab/Giraph
- More to come and play
Project Ideas

1. Graph Generator

2. Graph Consumer
Project Idea: Bisimulation-friendly Big Graph Generator

• In recent research, we see that power-law distribution in bisimulation results are not preserved in current synthetic graph generators.
• We want to change that.
• The task includes:
  0. Pick a distributed programming framework, map-reduce (or other distributed framework as you like, spark, hyracks), get comfortable with the programming environment.
  1. Design an algorithm that generates big graphs (billions of edges) that are
     1.1. power-law
     1.2. bisimulation friendly
     *1.3 other properties, such as small diameter
  2. Test, compare with other approaches, both in efficiency and in quality (e.g., socialbench, graph500)
Project Idea: External-Memory Giraph

- Giraph is an Apache copy of Pregel, a BSP-like computation framework for distributed environment. A very new and hot platform to play with.
- It is proved that BSP algorithms can be simulated in an external memory environment in an efficient way.
- In this research, we want to use external memory environment as a backend for Giraph, enabling its efficiency on single machine.
- The task includes:
  1. try out Giraph
  2. write a few classical algorithms in Giraph
  3. write the external memory backend, API-compatible
  4. compare the result on medium to large graph datasets (~1 billion edges)
Other related topics

• If you have some related ideas in mind, just come and talk to me.
Thank you!

Q&A

y.luo@tue.nl