Graph Data Processing

M. Tamer Özsu
Outline

Introduction

RDF Graph Querying

General Graph Processing
  Offline analytics
  Online querying
Graph Data are Very Common
Graph Data are Very Common

Social networks
Graph Data are Very Common

Trade volumes and connections
Graph Data are Very Common

Biological networks
Graph Data are Very Common

Linked data

Graph Types

RDF graph

- **mdb:film/2014**
  - release date: "1980-05-23"
  - title: "The Shining"
  - director: mdb:director/8476
  - actor: mdb:actor/29704
  - related book: bm:books/0743424425
  - rating: 4.7

- **mdb:director/8476**
  - name: "Stanley Kubrick"

- **mdb:actor/29704**
  - name: "Jack Nicholson"

- **mdb:film/2685**
  - title: "A Clockwork Orange"

- **mdb:film/424**
  - title: "Spartacus"

- **mdb:film/3418**
  - title: "The Passenger"

- **mdb:film/1267**
  - title: "The Last Tycoon"

- **bm:offers/0743424425amazonOffer**
  - rev:rating: 4.7
  - scam:hasOffer: bm:books/0743424425

- **geo:2635167**
  - name: "United Kingdom"
  - population: 62348447

- **mdb:actor/30013**

- **mdb:director/8476**
  - director: movie:director

- **mdb:film/1267**
  - director: mdb:director/8476
  - actor: mdb:actor/30013

- **mdb:film/2685**
  - director: mdb:director/8476
  - actor: mdb:actor/30013

- **mdb:actor/29704**
  - director: mdb:director/8476
  - actor: mdb:actor/30013

- **mdb:film/424**
  - director: mdb:director/8476
  - actor: mdb:actor/30013
Graph Types

Property graph
Graph Types

RDF graph

- Workload: SPARQL queries
- Query execution: subgraph matching by homomorphism

Property graph

- Workload: Online queries and analytic workloads
- Query execution: Much more varied
Outline

- Introduction

RDF Graph Querying

General Graph Processing
  - Offline analytics
  - Online querying
SPARQL Queries

```
SELECT ?name
WHERE {
    ?d movie:director_name "Stanley Kubrick".
    FILTER(?r > 4.0)
}
```

FILTER(?r > 4.0)
Graph-based Approach

- Answering SPARQL query $\equiv$ subgraph matching
- gStore, chameleon-db
gStore

General Approach:

▶ Work directly on the RDF graph and the SPARQL query graph
▶ Use a signature-based encoding of each entity and class vertex to speed up matching
▶ Filter-and-evaluate
  ▶ Use a false positive algorithm to prune nodes and obtain a set of candidates; then do more detailed evaluation on those
▶ Use an index (VS*-tree) over the data signature graph (has light maintenance load) for efficient pruning
1. Encode $Q$ and $G$ to Get Signature Graphs

**Query signature graph $Q^*$**

![Query signature graph](image)

**Data signature graph $G^*$**

![Data signature graph](image)
2. Filter-and-Evaluate

Query signature graph $Q^*$

Data signature graph $G^*$

Find matches of $Q^*$ over signature graph $G^*$
Verify each match in RDF graph $G$
How to Generate Candidate List

- Two step process:
  1. For each node of $Q^*$ get lists of nodes in $G^*$ that include that node.
  2. Do a multi-way join to get the candidate list
How to Generate Candidate List

- Two step process:
  1. For each node of $Q^*$ get lists of nodes in $G^*$ that *include* that node.
  2. Do a multi-way join to get the candidate list

- Alternatives:
How to Generate Candidate List

- Two step process:
  1. For each node of $Q^*$ get lists of nodes in $G^*$ that include that node.
  2. Do a multi-way join to get the candidate list

- Alternatives:
  - Sequential scan of $G^*$
    - Both steps are inefficient
How to Generate Candidate List

- Two step process:
  1. For each node of $Q^*$ get lists of nodes in $G^*$ that include that node.
  2. Do a multi-way join to get the candidate list

- Alternatives:
  - Sequential scan of $G^*$
    - Both steps are inefficient
  - Use S-trees
    - Height-balanced tree over signatures
    - Run an inclusion query for each node of $Q^*$ and get lists of nodes in $G^*$ that include that node.
      - Given query signature $q$ and a set of data signatures $S$, find all data signatures $s_i \in S$ where $q \& s_i = q$
      - Does not support second step – expensive
How to Generate Candidate List

- Two step process:
  1. For each node of $Q^*$ get lists of nodes in $G^*$ that include that node.
  2. Do a multi-way join to get the candidate list

- Alternatives:
  - Sequential scan of $G^*$
    - Both steps are inefficient
  - Use S-trees
    - Height-balanced tree over signatures
    - Run an inclusion query for each node of $Q^*$ and get lists of nodes in $G^*$ that include that node.
      - Given query signature $q$ and a set of data signatures $S$, find all data signatures $s_i \in S$ where $q \& s_i = q$
    - Does not support second step – expensive
  - VS-tree (and $VS^*$-tree)
    - Multi-resolution summary graph based on S-tree
    - Supports both steps efficiently
    - Grouping by vertices
S-tree Solution

\[
\begin{array}{c}
\text{0100 0000} \leftrightarrow 00010 \\
\text{1000 0000} \quad 10000 \\
\text{0000 0100}
\end{array}
\]

\[
\begin{array}{c}
G^1 \\
1111 1111
\end{array}
\]

\[
\begin{array}{c}
G^2 \\
0110 1111 \\
1101 1101
\end{array}
\]

\[
\begin{array}{c}
G^3 \\
0000 1110 \\
0110 1001 \\
1100 1001 \\
1001 1101
\end{array}
\]

\[
\begin{array}{c}
005 \\
0000 1000 \\
0000 0100 \\
0000 0010 \\
0000 0000
\end{array}
\]

\[
\begin{array}{c}
001 \\
0010 1000 \\
0001 0100 \\
0001 0010 \\
0000 1000
\end{array}
\]

\[
\begin{array}{c}
003 \\
1000 0001 \\
0100 0001 \\
0100 1000 \\
0000 0100
\end{array}
\]

\[
\begin{array}{c}
008 \\
1001 1000 \\
0001 0100 \\
0001 0001 \\
0000 0010
\end{array}
\]

\[
\begin{array}{c}
011 \\
0100 1000 \\
0010 1000 \\
0100 0001 \\
0001 0010
\end{array}
\]

Possibly large join space!
S-tree Solution

\[
\begin{array}{c}
0100 0000 \\
1000 0000 \\
0000 0100
\end{array}
\]

\[
\begin{array}{c}
G^1 \\
G^2 \\
G^3
\end{array}
\]

\[
\begin{array}{c}
0101 1111 \\
1101 1101 \\
1001 1101
\end{array}
\]

\[
\begin{array}{c}
0000 1000 \\
0010 1000 \\
1000 0001
\end{array}
\]

\[
\begin{array}{c}
0000 0100 \\
0000 0010 \\
0100 0001
\end{array}
\]

\[
\begin{array}{c}
0000 1001 \\
0100 1000 \\
0001 0100
\end{array}
\]

Possibly large join space!
S-tree Solution

Possibly large join space!

25 / 75
S-tree Solution

```
0100 0000 00010 10000 1111 1111 10000 0000 0100 002 011 008

G^1

1111 1111

G^2

1101 1101

G^3

1001 1101

Possibly large join space!
```
S-tree Solution

**G^1**

- **d_1^1**
  - 1111 1111

**G^2**

- **d_1^2**
  - 0110 1111

**G^3**

- **d_1^3**
  - 0000 1110
- **d_2^3**
  - 0110 1001
- **d_3^3**
  - 1100 1001

**Possibly large join space!**
S-tree Solution

```
0100 0000
  ↓
1000 0000
  ↓
0000 0100
  ↓
0000 0101
  ↓
0000 1000
  ↓
0000 1001
  ↓
0000 1010
  ↓
0000 1011
  ↓
0000 1100
  ↓
0000 1101
  ↓
0000 1110
  ↓
0000 1111
   ↓
1111 1111
   ↓
1101 1101
   ↓
1001 1000
   ↓
1001 1001
   ↓
1001 1100
   ↓
1001 1101
   ↓
1001 1110
   ↓
1001 1111
```

\[ G^1 \]

\[ G^2 \]

\[ G^3 \]

```
\begin{align*}
0100 0000 & \quad \text{00010} \quad 10000 \quad 0000 0100 \\
0110 1111 & \quad d_1 \quad 1111 1111 \\
0110 1001 & \quad d_2 \quad 1101 1101 \\
0000 1110 & \quad d_3 \quad 1000 1001 \\
0000 0100 & \quad 0000 0010 \\
0100 0001 & \quad 0100 0010 \\
0000 1001 & \quad 0000 1000 \\
1001 1001 & \quad 1001 1000 \\
0001 0001 & \quad 0001 0010 \\
0001 0100 & \quad 0001 0110 \\
0001 0100 & \quad 0001 0110 \\
0001 0100 & \quad 0001 0110 \\
0001 0100 & \quad 0001 0110 \\
0001 0100 & \quad 0001 0110 \\
0001 0100 & \quad 0001 0110 \\
```

Possibly large join space!
S-tree Solution

Possibly large join space!
Outline

Introduction

RDF Graph Querying

General Graph Processing
  Offline analytics
  Online querying
Classification

Graph Dynamism

Algorithm Types

Workload Types
Focus here is on the dynamism of the graphs in whether or not they change and how they change.
Classification

Graph Dynamism

Focus here is on the dynamism of the graphs in whether or not they change and how they change.

Algorithm Types

Focus here is on the how algorithms behave as their input changes.

Workload Types

Focus here is on the how algorithms behave as their input changes.
Focus here is on the dynamism of the graphs in whether or not they change and how they change.

Focus here is on the how algorithms behave as their input changes.

The types of workloads that the approaches are designed to handle.
Classification

Graph Dynamism:
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types:
- Offline
- Online
- Streaming
- Incremental
- Dynamic Batch
- Dynamic

Workload Types:
- Online Queries
- Analytics Workloads
Graphs do not change or we are not interested in their changes – only a snapshot is considered.
Classification

Graph Dynamism

- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types

- Offline
- Online
- Incremental
- Dynamic
- Batch

Workload Types

- Online Queries
- Analytics Workloads

Graphs do not change or we are not interested in their changes – only a snapshot is considered.

Graphs change and we are interested in their changes.
Graphs do not change or we are not interested in their changes – only a snapshot is considered.

Graphs change and we are interested in their changes.

Dynamic graphs with high velocity changes – not possible to see the entire graph at once.
Classification

Graph Dynamism

Static Graphs
Dynamic Graphs
Streaming Graphs
Evolving Graphs

Algorithm Types

Offline
Online
Incremental
Dynamic
Batch
Dynamic

Workload Types

Online Queries
Analytics Workloads

Graphs do not change or we are not interested in their changes – only a snapshot is considered.

Graphs change and we are interested in their changes.

Dynamic graphs with high velocity changes – not possible to see the entire graph at once.

Dynamic graphs with unknown changes – requires re-discovery of the graph (e.g., LOD).
Classification

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types
- Offline Algorithms
- Online Algorithms
- Streaming Algorithms
- Incremental Algorithms
- Dynamic Batch Algorithms
- Dynamic Incremental Algorithms

Workload Types
- Online Queries
- Analytics Workloads
Computation accesses a portion of the graph and the results are computed for a subset of vertices; e.g., point-to-point shortest path, subgraph matching, reachability, SPARQL.
Classification

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types
- Offline
- Online
- Streaming
- Incremental
- Dynamic
- Batch

Workload Types
- Online Queries
- Analytics Workloads

Computation accesses a portion of the graph and the results are computed for a subset of vertices; e.g., point-to-point shortest path, subgraph matching, reachability, SPARQL.

Computation accesses the entire graph and may require multiple iterations; e.g., PageRank, clustering, graph colouring, all pairs shortest path.
Classification

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types
- Offline
- Online
- Dynamic

Workload Types
- Online Queries
- Analytics Workloads

Sees the entire input in advance.
Classification

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types
- Offline
- Online
- Streaming
- Incremental

Workload Types
- Online Queries
- Analytics Workloads

Sees the entire input in advance.

Sees the input piece-meal as it executes.
Classification

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types
- Offline
- Online
- Streaming
- Incremental

Workload Types
- Online Queries
- Analytics Workloads

- Offline
  - Sees the entire input in advance.

- Online
  - Sees the input piece-meal as it executes.

- Streaming
- Incremental
  - One-pass online algorithm with limited memory.

- Dynamic
  - Batch Dynamic
Classification

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types
- Offline
- Online
- Dynamic
- Streaming
- Incremental
- Batch

Workload Types
- Online Queries
- Analytics Workloads

Sees the entire input in advance.
Sees the input piece-meal as it executes.
One-pass online algorithm with limited memory.
Online algorithm with some info about forthcoming input.
Classification

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types
- Offline
- Online
- Streaming
- Incremental

Workload Types
- Dynamic
- Batch
- Online Queries
- Analytics
  - Sees the entire input in advance, which may change; answers computed as change occurs.
  - Sees the entire input in advance.
  - One-pass online algorithm with limited memory.
  - Online algorithm with some info about forthcoming input.
Classification

Graph Dynamism
- Static Graphs
- Dynamic Graphs
- Streaming Graphs
- Evolving Graphs

Algorithm Types
- Offline
- Online
- Streaming
- Incremental
- Dynamic
- Batch
- Dynamic

Workload Types
- Online Queries
- Analytics

- Sees the entire input in advance.
- Sees the input piece-meal as it executes.
- One-pass online algorithm with limited memory.
- Online algorithm with some info about forthcoming input.
- Similar to dynamic, but computation happens in batches of changes.
- Sees the entire input in advance, which may change; answers computed as change occurs.
Dynamic (or batch-dynamic) algorithms do not make sense for static graphs.
Graph Workloads

**Offline graph analytics**

- PageRank
- Clustering
- Strongly connected components
- Diameter finding
- Graph colouring
- All pairs shortest path
- Graph pattern mining
- Machine learning algorithms (Belief propagation, Gaussian non-negative matrix factorization)

**Online graph querying**

- Reachability
- Single source shortest-path
- Subgraph matching
- SPARQL queries
PageRank Computation

A web page is important if it is pointed to by other important pages.

\[ r(P_i) = \sum_{P_j \in B_{P_i}} \frac{r(P_j)}{|F_{P_j}|} \]

\[ r(P_2) = \frac{r(P_1)}{2} + \frac{r(P_3)}{3} \]

\[ r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|F_{P_j}|} \]

- \( B_{P_i} \): in-neighbours of \( P_i \)
- \( F_{P_i} \): out-neighbours of \( P_i \)
PageRank Computation

A web page is important if it is pointed to by other important pages.

\[ r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|F_{P_j}|} \]

<table>
<thead>
<tr>
<th>Iteration 0</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Rank at Iter. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_0(P_1) = 1/6 )</td>
<td>( r_1(P_1) = 1/18 )</td>
<td>( r_2(P_1) = 1/36 )</td>
<td>5</td>
</tr>
<tr>
<td>( r_0(P_2) = 1/6 )</td>
<td>( r_1(P_2) = 5/36 )</td>
<td>( r_2(P_2) = 1/18 )</td>
<td>4</td>
</tr>
<tr>
<td>( r_0(P_3) = 1/6 )</td>
<td>( r_1(P_3) = 1/12 )</td>
<td>( r_2(P_3) = 1/36 )</td>
<td>5</td>
</tr>
<tr>
<td>( r_0(P_4) = 1/6 )</td>
<td>( r_1(P_4) = 1/4 )</td>
<td>( r_2(P_4) = 17/72 )</td>
<td>1</td>
</tr>
<tr>
<td>( r_0(P_5) = 1/6 )</td>
<td>( r_1(P_5) = 5/36 )</td>
<td>( r_2(P_5) = 11/72 )</td>
<td>3</td>
</tr>
<tr>
<td>( r_0(P_6) = 1/6 )</td>
<td>( r_1(P_6) = 1/6 )</td>
<td>( r_2(P_6) = 14/72 )</td>
<td>2</td>
</tr>
</tbody>
</table>

Iterative processing.
Some Alternative Computational Models for Offline Analytics

- MapReduce
  - map and reduce functions
  - Not suitable for iterative processing due to data movement at each stage
  - Need to save in storage system intermediate results of each iteration

- Vertex-centric paradigm
  - Specify (a) the computation to be performed at each vertex, and (b) its communication with neighbour vertices
  - Designed specifically for interactive graph processing
    - Synchronous (e.g., Pregel, Giraph)
    - Asynchronous (e.g., GraphLab)
Some Alternative Computational Models for Offline Analytics

- **MapReduce**
  - Map and reduce functions
  - Not suitable for iterative processing due to data movement at each stage
  - Need to save in storage system intermediate results of each iteration

- **Vertex-centric paradigm**
  - Specify (a) the computation to be performed at each vertex, and (b) its communication with neighbour vertices
  - Designed specifically for interactive graph processing
  - Synchronous (e.g., Pregel, Giraph)
  - Asynchronous (e.g., GraphLab)
Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, **vertex-centric** programs.
Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, **vertex-centric** programs.

Computation
Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, **vertex-centric** programs.

Superstep 1  \[\rightarrow\]  Superstep 2  \[\rightarrow\]  Superstep 3
Pregel-like systems are **BSP**, *vertex-centric* programs.
Pregel-like systems are **BSP**, **vertex-centric** programs.
Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, **vertex-centric** programs.
Pregel-like Graph Processing Systems

Pregel-like systems are **BSP**, *vertex-centric* programs.

- “Think like a vertex”: 

![Diagram of a central vertex connected to four other vertices]
GraphLab (Asynchronous)

GraphLab features *asynchronous* execution:

- No communication barriers. ✔️
- Uses the *most recent* vertex values. ✔️
GraphLab (Asynchronous)

Implemented via distributed locking:
GraphLab (Asynchronous)

Implemented via distributed locking:

\[ v_0 \rightarrow v_1 \rightarrow v_3 \rightarrow v_0 \]
\[ v_0 \rightarrow v_2 \rightarrow v_4 \rightarrow v_0 \]
GraphLab (Asynchronous)

Implemented via distributed locking:
GraphLab (Asynchronous)

Implemented via distributed locking:

\[
\begin{align*}
&v_0 \\
&v_1 \quad v_0 \\
&v_2 \quad v_0 \\
&v_3 \quad v_0 \\
&v_4 \quad v_0
\end{align*}
\]
GraphLab (Asynchronous)

Implemented via distributed locking:
Reachability Queries
Can you reach film_1267 from film_2014?
Is there a book whose rating is $> 4.0$ associated with a film that
was directed by Stanley Kubrick?
Reachability Queries

Think of Facebook graph and finding friends of friends.
Subgraph Matching

FILTER(?r > 4.0)

?name rdfs:label "Stanley Kubrick"

?b movie:relatedBook "Stanley Kubrick"

?d movie:director "Stanley Kubrick"

?m movie:director "Stanley Kubrick"

FILTER(?r > 4.0)

bm:offers/0743424425amazonOffer scam:hasOffer

bm:books/0743424425

rev:rating 4.7

"United Kingdom" geo:2635167

62348447

db:film/3418

"The Passenger" refs:label

mdb:actor/29704

movie:actor "Jack Nicholson"

mdb:film/1267

"The Last Tycoon" refs:label

mdb:director/8476

movie:director "Stanley Kubrick"

mdb:director/8476

movie:director "Stanley Kubrick"

mdb:film/2685

refs:label "A Clockwork Orange"

mdb:film/424

refs:label "Spartacus"

mdb:film/1267

"The Last Tycoon" refs:label

mdb:film/3418

"The Passenger" refs:label

mdb:actor/29704

movie:actor "Jack Nicholson"

mdb:film/1267

"The Last Tycoon" refs:label

mdb:director/8476

movie:director "Stanley Kubrick"

mdb:film/2685

refs:label "A Clockwork Orange"

mdb:film/424

refs:label "Spartacus"
Graph-based SPARQL processing


For more information II

More on RDF graph management


- More organized slides from my talks are available at http://www.slideshare.net/MTamerOzsu/web-data-management-with-rdf

General graph processing
