Graph-Parallel Querying of RDF with GraphX [1]

Alexander Schätzle, Martin Przyjaciel-Zablocki, Thorsten Berberich, Georg Lausen
University of Freiburg, Germany

schaetzle@informatik.uni-freiburg.de

Big-O(Q) 2015 – Hawaii, USA
Motivation

- **Semantic Web has arrived in real-world applications** (not only academia & research)
  - Web-scale semantic data makes single machine solutions infeasible

- **Hadoop ecosystem has become de-facto standard for Big Data applications**

- **Our Idea**: Use it for Semantic Web purposes as well
  - Main reasons:
    1) Web-scale semantic data requires solutions that scale out
    2) Industry has settled on Hadoop (or Hadoop-style) architectures
       ➜ superior cost-benefit ratio compared to specialized infrastructures
Motivation

- **RDF/SPARQL-on-Hadoop**
  - Existing solutions are based on *relational-style* data models implemented on *data-parallel* frameworks
  - Hence they do not exploit the inherent graph-like structure of RDF

- **Parallel Graph Processing Systems**
  - e.g. *Pregel, Giraph, GraphLab*
  - Tailored towards iterative graph algorithms
  - **But:** Not integrated with Hadoop or hard to combine with other frameworks
    - Data movement or duplication

- **S2X (SPARQL on Spark with GraphX)**
  - Spark GraphX allows to combine graph-parallel and data-parallel computation
  - GraphX used for graph pattern matching
  - Other SPARQL operators implemented in Spark
Apache Spark
- General-purpose in-memory cluster computing system
- Based on Resilient Distributed Datasets (RDD)
  - distributed, fault-tolerant collection of elements
  - kept in memory and partitioned across the cluster
- Jobs are modeled as Directed Acyclic Graphs (DAG) of tasks
  - each tasks runs on a horizontal partition of an RDD

GraphX
- Spark API for graphs and graph-parallel computation
- Uses a Property Graph Data Model (represented by two RDDs)
- Vertex-Centric view of graphs (similar to Pregel)
- Adopts Bulk-Synchronous Parallel (BSP) execution model
  - vertex programs run concurrently in a sequence of Supersteps
- Vertex-Cut partitioning
  - evenly assigns edges to machines, vertices can span multiple machines
RDF Property Graph Model

- **Property Graph (PG)**
  - $PG(P) = (V, E, P)$
  - $V =$ set of vertices, $E =$ set of directed edges
  - $P = (P_V, P_E) =$ collection of vertex and edge properties (key, value)

  ![Diagram](image)

- **RDF Graph (G)**
  - $G = \{t_1, ..., t_n\} =$ set of RDF triples
  - RDF triple $t = (\_subject, \_predicate, \_object)$

  ![Diagram](image)
RDF Property Graph Model

- **RDF to PG - Example**
  - \( G = \{(A, \text{knows}, B), (A, \text{likes}, B), (A, \text{likes}, C), (B, \text{knows}, C)\} \)

```
\begin{align*}
I_D(A) &= v_A \\
P_E(v_A, v_B).label &= \text{knows} \\
P_E(v_B, v_C).label &= \text{knows} \\
P_E(v_A, v_C).label &= \text{likes}
\end{align*}
```
### Standard Query Language for RDF

```sparql
SELECT * WHERE {
}
```

- **Triple Pattern**
  - Result of a **triple pattern** $tp$ is a bag of mappings
    
    $$\Omega_{tp} = \{\mu \mid \text{dom}(\mu) = \text{vars}(tp), \mu(tp) \in G\}$$

- **Basic Graph Pattern**
  - Result of a **basic graph pattern** $bgp = \{tp_1, ..., tp_m\}$ is the merge of compatible mappings (Join on common variables)
    
    $$\Omega_{bgp} = \Omega_{tp_1} \Join ... \Join \Omega_{tp_m}$$

### (Solution) Mapping

- $\mu : V \to RDF$ Terms
- Result of a **triple pattern** $tp$ is a bag of mappings
SPARQL - Example

```
SELECT * WHERE {
}
```

Diagram:

- Node A knows B
- Node A likes B
- Node B knows C

- Node A likes B
- Node B knows C
- Node B likes C
SPARQL - Example

```
SELECT * WHERE {
  ?B knows ?C
}
```

A \( \rightarrow \) B \( \rightarrow \) C

\( \Omega_{tp_1} \)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>?A</td>
<td>?B</td>
</tr>
<tr>
<td>( \mu_{11} )</td>
<td>A</td>
</tr>
<tr>
<td>( \mu_{12} )</td>
<td>B</td>
</tr>
</tbody>
</table>
SPARQL - Example

```
SELECT * WHERE {
    ?B knows ?C
}
```

A knows B
B knows C
B likes A

```
Ω_{tp_1}

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>μ_{11}</td>
<td>A</td>
</tr>
<tr>
<td>μ_{12}</td>
<td>B</td>
</tr>
</tbody>
</table>
```

```
Ω_{tp_2}

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>μ_{21}</td>
<td>A</td>
</tr>
<tr>
<td>μ_{22}</td>
<td>A</td>
</tr>
</tbody>
</table>
```
SPARQL - Example

```
SELECT * WHERE {
}
```

The query involves relationships `knows` between entities and `likes` between other entities. The diagram illustrates these relationships with nodes labeled `A`, `B`, and `C`, and relationships marked `knows` and `likes`.

The tables `Ω_{tp_1}`, `Ω_{tp_2}`, and `Ω_{tp_3}` display the matching between entities, where:
- `Ω_{tp_1}` relates `A` and `B` through `μ_11` and `μ_12`.
- `Ω_{tp_2}` relates `A` and `B` through `μ_21`, `μ_22`.
- `Ω_{tp_3}` relates `B` and `C` through `μ_31`, `μ_32`.

This setup facilitates the querying process in a graph-parallel manner with GraphX, enabling efficient processing of large RDF datasets.
SPARQL - Example

```
SELECT * WHERE {
}

Ω_{tp_1} = \begin{array}{cc}
\mu_{11} & A \\
\mu_{12} & B
\end{array}
\begin{array}{cc}
\mu_{21} & A \\
\mu_{22} & C
\end{array}

Ω_{tp_2} = \begin{array}{cc}
\mu_{11} & A \\
\mu_{12} & B
\end{array}
\begin{array}{cc}
\mu_{21} & A \\
\mu_{22} & C
\end{array}

Ω_{tp_3} = \begin{array}{cc}
\mu_{31} & A \\
\mu_{32} & B
\end{array}
\begin{array}{cc}
\mu_{31} & B \\
\mu_{32} & C
\end{array}

Ω_{bgp} = \{ \mu : (?A \rightarrow A, ?B \rightarrow B, ?C \rightarrow C) \}
Basic Idea:
- Every vertex stores the variables of a query where it is a candidate for
- Vertices exchange their candidate sets for validation

Match Candidate:
- \( \text{matchC} = (v, \text{?var}, \mu, tp) \)
- \( v \) is a candidate for \( \text{?var} \) with mapping \( \mu \) derived from triple pattern \( tp \)

\[
\begin{align*}
\nu_A &\quad \text{label} = A \\
\nu_B &\quad \text{label} = B \\
\end{align*}
\]

\[
\begin{align*}
\text{label} &= \text{knows} \\
\end{align*}
\]

\[
\begin{align*}
\nu_A, \text{?A,}\mu: (?A \rightarrow A, ?B \rightarrow B), tp) &\quad (\nu_B, \text{?B,}\mu: (?A \rightarrow A, ?B \rightarrow B), tp) \\
\end{align*}
\]

Match Set:
- Set of all match candidates of a vertex
- Match set is stored as a property of the vertex \( P_V(v).\text{matchS} \)
BGP Matching in a Nutshell

- **Superstep 1:**
  - Determine all possible match candidates
    - Iterate over all edges and match with all triple patterns at once
  - Yields an overestimation of the BGP result

- **Following Supersteps:**
  - Validate *Local* Match Sets
    - Does a vertex have all required match candidates for a variable?
  - Validate *Remote* Match Sets
    - Are the neighbors of vertex still the required candidates?
  - Exchange Match Sets with Neighbor Vertices
  - Repeat until no changes occur

- **Result Generation (solution mappings):**
  - Collect and merge (join) all match sets to produce the final mappings
BGP Matching - Example

SELECT * WHERE {
    ?B knows ?C
}

Superstep 1

\[
\begin{align*}
\nu_A &\quad label = A \\
\nu_B &\quad label = B \\
\nu_C &\quad label = C
\end{align*}
\]

\[
\begin{align*}
?A &\quad \mu \quad tp_1 \\
?B &\quad \mu \quad tp_3
\end{align*}
\]

\[
\begin{align*}
?B &\quad \mu \quad tp_1 \\
?C &\quad \mu \quad tp_3
\end{align*}
\]
BGP Matching - Example

SELECT * WHERE {

}  

\[ \mu \]

Superstep 1

\[ \nu_A \quad \nu_B \quad \nu_C \]

\[ \text{label} = \text{likes} \]

\[ \text{label} = \text{knows} \]

\[ \text{label} = \text{likes} \]

<table>
<thead>
<tr>
<th>( ? \text{var} )</th>
<th>( \mu )</th>
<th>( \text{tp} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>?A</td>
<td>(?A ( \rightarrow ) A, ?B ( \rightarrow ) B)</td>
<td>( \text{tp}_1 )</td>
</tr>
<tr>
<td>?B</td>
<td>(?B ( \rightarrow ) A, ?C ( \rightarrow ) B)</td>
<td>( \text{tp}_3 )</td>
</tr>
<tr>
<td>?A</td>
<td>(?A ( \rightarrow ) A, ?B ( \rightarrow ) B)</td>
<td>( \text{tp}_2 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( ? \text{var} )</th>
<th>( \mu )</th>
<th>( \text{tp} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>?B</td>
<td>(?A ( \rightarrow ) A, ?B ( \rightarrow ) B)</td>
<td>( \text{tp}_1 )</td>
</tr>
<tr>
<td>?C</td>
<td>(?B ( \rightarrow ) A, ?C ( \rightarrow ) B)</td>
<td>( \text{tp}_3 )</td>
</tr>
<tr>
<td>?B</td>
<td>(?A ( \rightarrow ) A, ?B ( \rightarrow ) B)</td>
<td>( \text{tp}_2 )</td>
</tr>
</tbody>
</table>
BGP Matching - Example

```
SELECT * WHERE {
} 

\[ \text{tp}_1 \]  \[ \text{tp}_2 \]  \[ \text{tp}_3 \] 

- Superstep 1

\[ v_A \quad \text{label} = A \quad \text{label} = \text{likes} \quad \text{label} = \text{knows} \]
\[ v_B \quad \text{label} = B \quad \text{label} = \text{knows} \quad \text{label} = \text{likes} \]
\[ v_C \quad \text{label} = C \]

<table>
<thead>
<tr>
<th>?var</th>
<th>( \mu )</th>
<th>tp</th>
</tr>
</thead>
<tbody>
<tr>
<td>?A</td>
<td>(( ?A \rightarrow A, ?B \rightarrow B ))</td>
<td>\text{tp}_1</td>
</tr>
<tr>
<td>?B</td>
<td>(( ?B \rightarrow A, ?C \rightarrow B ))</td>
<td>\text{tp}_3</td>
</tr>
<tr>
<td>?A</td>
<td>(( ?A \rightarrow A, ?B \rightarrow B ))</td>
<td>\text{tp}_2</td>
</tr>
<tr>
<td>?A</td>
<td>(( ?A \rightarrow A, ?B \rightarrow C ))</td>
<td>\text{tp}_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?var</th>
<th>( \mu )</th>
<th>tp</th>
</tr>
</thead>
<tbody>
<tr>
<td>?B</td>
<td>(( ?A \rightarrow A, ?B \rightarrow B ))</td>
<td>\text{tp}_1</td>
</tr>
<tr>
<td>?C</td>
<td>(( ?B \rightarrow A, ?C \rightarrow B ))</td>
<td>\text{tp}_3</td>
</tr>
<tr>
<td>?B</td>
<td>(( ?A \rightarrow A, ?B \rightarrow B ))</td>
<td>\text{tp}_2</td>
</tr>
<tr>
<td>?B</td>
<td>(( ?A \rightarrow A, ?B \rightarrow C ))</td>
<td>\text{tp}_2</td>
</tr>
</tbody>
</table>

```
BGP Matching - Example

\[\text{SELECT } * \text{ WHERE } \{ \begin{array}{c}
\text{?A knows ?B . } \\
\text{?A likes ?B . } \\
\text{?B knows ?C}
\end{array} \} \]

- **Superstep 1**

\[\begin{array}{c}
\nu_A \text{ label } = A \\
\nu_B \text{ label } = B \\
\nu_C \text{ label } = C
\end{array} \]

\[\begin{array}{c}
\text{label } = \text{ knows} \\
\text{label } = \text{ likes}
\end{array} \]

\[\begin{array}{c}
?A \text{ (} ?A \rightarrow A, ?B \rightarrow B \text{)} \text{ tp}_1 \\
?B \text{ (} ?B \rightarrow A, ?C \rightarrow B \text{)} \text{ tp}_3 \\
?A \text{ (} ?A \rightarrow A, ?B \rightarrow B \text{)} \text{ tp}_2 \\
?A \text{ (} ?A \rightarrow A, ?B \rightarrow C \text{)} \text{ tp}_2
\end{array} \]

\[\begin{array}{c}
?B \text{ (} ?A \rightarrow A, ?B \rightarrow B \text{)} \text{ tp}_1 \\
?C \text{ (} ?B \rightarrow A, ?C \rightarrow B \text{)} \text{ tp}_3 \\
?B \text{ (} ?A \rightarrow A, ?B \rightarrow B \text{)} \text{ tp}_2 \\
?B \text{ (} ?B \rightarrow B, ?C \rightarrow C \text{)} \text{ tp}_3
\end{array} \]
## BGP Matching - Example

SELECT * WHERE {
}

**Superstep 1**

- **label = likes**
  - $v_A$ (label = A)
  - $v_B$ (label = B)
  - $v_C$ (label = C)

- **label = knows**
  - $v_A$ (label = likes) → $v_B$ (label = knows) → $v_C$ (label = knows)

- **Overestimation!**

<table>
<thead>
<tr>
<th>var</th>
<th>$\mu$</th>
<th>$tp$</th>
</tr>
</thead>
<tbody>
<tr>
<td>?A</td>
<td>(?A → A, ?B → B)</td>
<td>$tp_1$</td>
</tr>
<tr>
<td>?B</td>
<td>(?B → A, ?C → B)</td>
<td>$tp_3$</td>
</tr>
<tr>
<td>?A</td>
<td>(?A → A, ?B → B)</td>
<td>$tp_2$</td>
</tr>
<tr>
<td>?A</td>
<td>(?A → A, ?B → C)</td>
<td>$tp_2$</td>
</tr>
<tr>
<td>?B</td>
<td>(?A → A, ?B → B)</td>
<td>$tp_2$</td>
</tr>
<tr>
<td>?B</td>
<td>(?B → A, ?B → B)</td>
<td>$tp_2$</td>
</tr>
<tr>
<td>?C</td>
<td>(?B → A, ?C → B)</td>
<td>$tp_3$</td>
</tr>
<tr>
<td>?B</td>
<td>(?A → B, ?B → C)</td>
<td>$tp_1$</td>
</tr>
<tr>
<td>?B</td>
<td>(?B → B, ?C → C)</td>
<td>$tp_3$</td>
</tr>
<tr>
<td>?B</td>
<td>(?A → A, ?B → C)</td>
<td>$tp_1$</td>
</tr>
<tr>
<td>?C</td>
<td>(?B → B, ?C → C)</td>
<td>$tp_3$</td>
</tr>
<tr>
<td>?B</td>
<td>(?A → A, ?B → C)</td>
<td>$tp_2$</td>
</tr>
<tr>
<td>?B</td>
<td>(?A → B, ?B → C)</td>
<td>$tp_1$</td>
</tr>
<tr>
<td>?C</td>
<td>(?B → B, ?C → C)</td>
<td>$tp_3$</td>
</tr>
</tbody>
</table>
BGP Matching - Example

---

**Superstep 2**

Validation of Local Match Sets:

Does \((?B, (\{?B \rightarrow A, \ldots\}, tp_1))\) exist in \(v_A \Rightarrow NO!\)
BGP Matching - Example

SELECT * WHERE {
}  tp1  tp2  tp3

Superstep 2

Validation of Local Match Sets:

Does (?B , (?B → A,... ) , tp1) exist in νA ⇒ NO!

Does (?A , (?A → A, ?B → C) , tp1) exist in νA ⇒ NO!
BGP Matching - Example

SELECT * WHERE {
}  

Superstep 2

\[ \nu_A \]  
\[ \nu_B \]  
\[ \nu_C \]

\[ \var{\text{label}} = A \]  
\[ \var{\text{label}} = B \]  
\[ \var{\text{label}} = C \]

\[ \var{\text{label}} = \text{likes} \]  
\[ \var{\text{label}} = \text{knows} \]  
\[ \var{\text{label}} = \text{likes} \]

?var | \mu | tp
--- | --- | ---
\?A | (?A \rightarrow A, \?B \rightarrow B) | tp_1
\?B | (?B \rightarrow A, \?C \rightarrow B) | tp_3
\?A | (?A \rightarrow A, \?B \rightarrow B) | tp_2
\?B | (?B \rightarrow A, \?B \rightarrow C) | tp_2

?var | \mu | tp
--- | --- | ---
\?B | (?A \rightarrow A, \?B \rightarrow C) | tp_2
\?C | (?B \rightarrow A, \?B \rightarrow C) | tp_3
\?B | (?A \rightarrow B, \?B \rightarrow C) | tp_1
\?C | (?B \rightarrow B, \?C \rightarrow C) | tp_3

Removed!
BGP Matching - Example

SELECT * WHERE {
  ?B knows ?C
}

Superstep 2

?var  μ  tp
?A  (?A → A, ?B → B)  tp₁
?A  (?A → A, ?B → B)  tp₂

?var  μ  tp
?B  (?A → A, ?B → B)  tp₁
?C  (?B → A, ?C → B)  tp₃
?B  (?A → A, ?B → B)  tp₂
?B  (?B → B, ?C → C)  tp₃

label = knows

label = likes

label = knows

label = likes
BGP Matching - Example

```
SELECT * WHERE {
}
```

### Superstep 3

- **Label = likes**
- **Label = knows**

**Validation of Remote Match Sets:**

Does \((?B, (?B \rightarrow A, ?C \rightarrow B), tp_3)\) exist in \(v_A \Rightarrow NO!\)
BGP Matching - Example

### Superstep 3

```sparql
SELECT * WHERE {
}
```

- **tp₁**
- **tp₂**
- **tp₃**

<table>
<thead>
<tr>
<th>?var</th>
<th>μ</th>
<th>tp</th>
</tr>
</thead>
<tbody>
<tr>
<td>?A</td>
<td>(? A → A, ?B → B)</td>
<td>tp₁</td>
</tr>
<tr>
<td>?A</td>
<td>(? A → A, ?B → B)</td>
<td>tp₂</td>
</tr>
<tr>
<td>?B</td>
<td>(? A → A, ?B → B)</td>
<td>tp₁</td>
</tr>
<tr>
<td>?C</td>
<td>(? B → A, ?C → B)</td>
<td>tp₃</td>
</tr>
<tr>
<td>?B</td>
<td>(? A → A, ?B → B)</td>
<td>tp₂</td>
</tr>
<tr>
<td>?B</td>
<td>(? B → B, ?C → C)</td>
<td>tp₃</td>
</tr>
</tbody>
</table>

- No Changes!
- **Removed!**
- No Changes!
BGP Matching - Example

SELECT * WHERE {
}  

\[ v_A \quad label = A \]
\[ v_B \quad label = B \]
\[ v_C \quad label = C \]

\[ label = likes \]
\[ label = knows \]

\[ ?A \quad (\ ?A \rightarrow A , \ ?B \rightarrow B ) \quad tp_1 \]
\[ ?A \quad (\ ?A \rightarrow A , \ ?B \rightarrow B ) \quad tp_2 \]

\[ ?B \quad (\ ?A \rightarrow A , \ ?B \rightarrow B ) \quad tp_1 \]
\[ ?B \quad (\ ?A \rightarrow A , \ ?B \rightarrow B ) \quad tp_2 \]
\[ ?B \quad (\ ?B \rightarrow B , \ ?C \rightarrow C ) \quad tp_3 \]

\[ ?C \quad (\ ?B \rightarrow B , \ ?C \rightarrow C ) \quad tp_3 \]
BGP Matching - Example

```
SELECT * WHERE {
}                     tp₁   tp₂   tp₃
```

- **Superstep 4**

  ![Diagram of graph matching]

  - \( v_A \)  
    - label = \( A \)
    - \( ?A \)  
        - \( ?A \rightarrow A \)
        - \( ?A \rightarrow B \)
        - \( ?A \rightarrow C \)
  - \( v_B \)  
    - label = \( B \)
    - \( ?B \)  
        - \( ?A \rightarrow A \)
        - \( ?B \rightarrow B \)
        - \( ?B \rightarrow C \)
  - \( v_C \)  
    - label = \( C \)
    - \( ?C \)  
        - \( ?B \rightarrow B \)
        - \( ?C \rightarrow C \)

<table>
<thead>
<tr>
<th>(? var)</th>
<th>( \mu )</th>
<th>( tp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(? A)</td>
<td>( ?A \rightarrow A, ?B \rightarrow B)</td>
<td>( tp₁)</td>
</tr>
<tr>
<td>(? A)</td>
<td>( ?A \rightarrow A, ?B \rightarrow B)</td>
<td>( tp₂)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(? var)</th>
<th>( \mu )</th>
<th>( tp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(? B)</td>
<td>( ?A \rightarrow A, ?B \rightarrow B)</td>
<td>( tp₁)</td>
</tr>
<tr>
<td>(? B)</td>
<td>( ?A \rightarrow A, ?B \rightarrow B)</td>
<td>( tp₂)</td>
</tr>
<tr>
<td>(? B)</td>
<td>( ?B \rightarrow B, ?C \rightarrow C)</td>
<td>( tp₃)</td>
</tr>
</tbody>
</table>

- No Changes!

- No Changes!
Experiments

- **Small Cluster with low-end configuration**
  - 10 machines (1 master, 9 worker), 2 disks each, Gigabit network
  - 24 GB RAM for Spark, 60% assigned for graph caching
    (typical Hadoop nodes currently ≥ 256 GB RAM, ≥ 12 disks, ≥ 10 Gigabit network)
  - CDH 5.3, Spark 1.2.0, Pig 0.12

- **WatDiv Benchmark [2]**
  - Combines e-commerce scenario with a kind of social network
  - All different kinds of query shapes (*star, linear, snowflake, complex*)
  - Scale Factors:
    - 10 (10K users, ~0.1M vertices)
    - 100 (100K users, ~1M vertices)
    - 1000 (1M users, ~10M vertices)

- **Compared with PigSPARQL [3]**
  - SPARQL Query Processing Baseline for Hadoop
  - Proven to be competitive to other Hadoop research approaches
Experiments

- **Geometric Mean per Query Group**

  - **S2X** outperforms **PigSPARQL** by up to an order of magnitude
  - Runtime of S2X strongly depends on result size (better for small output)
  - Runtime can be highly influenced by adjusting parameters of Spark/GraphX
  - GraphX currently quite “memory-heavy” (early stage of development)
  - Very good preliminary results but still much room for improvement in S2X
Summary

- **S2X (SPARQL on Spark with GraphX)**
  - SPARQL Query Processor for Hadoop based on Spark GraphX
  - Property Graph representation for RDF
  - Combines graph-parallel and data-parallel computation
    - graph-parallel: BGP matching
    - data-parallel: BGP solution mapping generation + other SPARQL operators

- **Experiments**
  - Very good preliminary results
    - Up to an order of magnitude performance improvement compared to a state-of-the-art SPARQL Processor baseline for Hadoop
  - GraphX is in an early stage of development
    - Many improvements can be expected in future GraphX versions
  - Many possible improvements for S2X
    - Incremental exchange of match sets \(\rightarrow\) I/O reduction
    - Optimization of graph partitioning \(\rightarrow\) avoid unbalanced workloads
    - Incremental BGP matching for selective queries \(\rightarrow\) Reduction of Overestimation
Thank you for your attention!

Questions?
References

[1] Alexander Schätzle, Martin Przyjaciel-Zablocki, Thorsten Berberich, Georg Lausen: 
*S2X: Graph-Parallel Querying of RDF with GraphX*. In: Big-O(Q) 2015


*PigSPARQL: A SPARQL Query Processing Baseline for Big Data*. In: ISWC 2013 Posters
Experiments

- Star Queries

**Geometric Mean**

<table>
<thead>
<tr>
<th></th>
<th>S2X</th>
<th>PigSPARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.63</td>
<td>41.43</td>
</tr>
<tr>
<td>100</td>
<td>5.33</td>
<td>41.28</td>
</tr>
<tr>
<td>1000</td>
<td>31.53</td>
<td>46.74</td>
</tr>
</tbody>
</table>

**S2X - BGP Matching vs. Result Generation**

<table>
<thead>
<tr>
<th></th>
<th>S2X</th>
<th>PigSPARQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>0.70</td>
<td>1.60</td>
</tr>
<tr>
<td>BGP</td>
<td>1.93</td>
<td>3.73</td>
</tr>
<tr>
<td>1000</td>
<td>6.87</td>
<td>24.66</td>
</tr>
</tbody>
</table>
Experiments

- **Linear Queries**

  - Strong variation in runtimes (min: ~14s, max: ~348s)
  - For the long running queries, result generation takes more time than BGP matching, e.g. 348s = (124s, 224s) → many results
Experiments

- Snowflake Queries

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2X</td>
<td>5.84</td>
<td>12.99</td>
<td>76.63</td>
</tr>
<tr>
<td>PigSPARQL</td>
<td>85.47</td>
<td>85.21</td>
<td>93.85</td>
</tr>
</tbody>
</table>

**Geometric Mean**

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>100</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2X</td>
<td>2.58</td>
<td>6.20</td>
<td>32.39</td>
</tr>
<tr>
<td>BGP</td>
<td>3.26</td>
<td>6.79</td>
<td>44.25</td>
</tr>
</tbody>
</table>

S2X - BGP Matching vs. Result Generation
Experiments

- Complex Queries

- Queries produce many results
  - for some queries, result generation takes more time than BGP matching