SPOWL: Spark-based OWL 2 Reasoning Materialisation

Yu Liu and Peter McBrien

Department of Computing, Imperial College London
Table of Contents

Introduction

SPOWL Overview

SPOWL Features

Evaluation

Summary
Table of Contents

Introduction

SPOWL Overview

SPOWL Features

Evaluation

Summary
Reasoning materialisation for OWL 2 ontologies

LUBM T-Box:
- Student ⊆ Person
- Student ⊆ ∃takesCourse.Course

LUBM A-Box:
- Student(John)  (3)  Person(Lewis)  (5)
- Student(Tom)   (4)  Person(Mary)  (6)

Reasoning materialisation:
- Student := {John, Tom};  Person := {Lewis, Mary, John, Tom}
- takesCourse := {(John, ?C1), (Tom, ?C2)};  Course := {?C1, ?C2}

Querying the ontology:
- Not only explicit but also implicit facts will be returned.
Reasoning materialisation for OWL 2 ontologies

Materialising reasoning results:

Student := \{John, Tom\}

Person := \{Lewis, Mary, John, Tom\}

takesCourse := \{(John, ?C2), (Tom, ?C2)\}

Course := \{?C1, ?C2\}

▶ Queries directly read the materialised results.
▶ Faster query processing and larger space required.
▶ Maintenance of the materialisation is difficult.
▶ Ideal case: queries are much more frequent than updates.
▶ Example systems: SPOWL, Oracle’s RDF Store, WebPIE, etc.
Rule evaluation for reasoning materialisation

- Rule format: \[ \text{if } \langle \text{antecedent} \rangle \text{ then } \langle \text{consequent} \rangle: \]
  
  Example: \[ \text{if } C \sqsubseteq D, C(x) \text{ then } D(x) \]
  
  \[ \Rightarrow \text{ if Student } \sqsubseteq \text{Person}, \text{Student}(x) \text{ then Person}(x) \]

- Well-known rule sets:
  - RDFS entailment rules.
  - OWL ter Horst rules.
  - OWL 2 RL/RDF rules.

- Limitations:
  - No use of tableaux reasoners (e.g. Pellet and Hermit).
  - Reasoning relies on which set of entailment rules is chosen.
  - Inefficient rule matching process.
Table of Contents

Introduction

SPOWL Overview

SPOWL Features

Evaluation

Summary
SPOWL architecture

- T-Box is small enough for tableaux reasoners.
- The number of queries is much larger than the number of updates.
SPOWL overview

1. Classes & properties to Spark RDDs:
   \[ C \leadsto C_{rdd}(id) \quad \text{and} \quad P \leadsto P_{rdd}(\text{domain, range}) \]

2. T-Box axioms are mapped to entailment rules \( R_{\text{axiom}} \):
   \[ C \subseteq D \leadsto R_{C \subseteq D} ::= \text{if } C_{rdd}(x) \text{ then } D_{rdd}(x) \]

3. \( R_{\text{axiom}} \) are further implemented as Spark programmes \( P_{\text{axiom}} \):
   \[ R_{C \subseteq D} \leadsto P_{C \subseteq D} ::= D_{rdd} = D_{rdd}.\text{union}(C_{rdd}) \]

4. \( P_{\text{axiom}} \) are iteratively executed to build up the RDDs.
Table of Contents

Introduction

SPOWL Overview

SPOWL Features

Evaluation

Summary
SPOWL uses tableaux reasoner

- More complete T-Box reasoning:
  
  e.g. classifying $C \sqsubseteq D \sqcup E$ gives us $C \sqsubseteq E$

  $C \cap D \sqsubseteq \bot$

- Entailment rules are specific to the A-Box data:
  
  - No need to evaluate rules that are irrelevant to the ontological data.
SPOWL partitions reasoning materialisation

- Data of each class or property is stored separately in HDFS:
  \[ C \sim \text{hdfs://$\{C\_PATH}\}/ \quad P \sim \text{hdfs://$\{P\_PATH\}/} \]

- A variant of the vertical partitioning model.
  - Only the partitions storing the relevant data need to be accessed.
    e.g. \texttt{Student\_rdd = sc.textfile("hdfs://$\{Student\_PATH\}/")}
  - Otherwise, the whole ontology should be read and a fragment of it should be filtered out.
SPOWL handles axioms beyond OWL 2 RL

- **SomeValuesFrom** forms a superclass expression (i.e. $C \sqsubseteq \exists P.D$)
  e.g. $\text{Student} \sqsubseteq \exists \text{takesCourse}.\text{Course}(2)$

- Non-deterministic reasoning (OWL 2 RL Interpretation $\mathcal{I}$):
  $$\mathcal{I} \models C \sqsubseteq \exists P.D \iff C^\mathcal{I} \subseteq \{x \mid \exists y : \langle x, y \rangle \in P^\mathcal{I} \text{ and } y \in D^\mathcal{I}\}$$

- Entailment rule $\mathcal{R}_{C \sqsubseteq \exists P.D}$:
  $$\text{if } C_{rdd}(x), \neg P_{rdd}(x, y) \text{ then } P_{rdd}(x, null)$$

- Spark programme $\mathcal{P}_{C \sqsubseteq \exists P.D}$:
  $$P_{rdd} = P_{rdd}.\text{union}(C_{rdd}.\text{subtract}(P_{rdd}.\text{map}(\lambda (x, y) : x)).\text{map}(\lambda x : (x, null)))$$
The advantage of using Spark (1)

Spark caches RDDs in distributed memory as much as possible:

- reduce the needs to write/read intermediate results to/from disk.
- reduce I/O overhead.
- suitable for iterative computation (e.g. computing transitive closure).
Data caching in distributed memory

Iterative computation:

▶ TransitiveProperty $P \ (P \circ P \subseteq P)$.

(7)

subOrganisationOf $\circ$ subOrganisationOf $\subseteq$ subOrganisationOf

▶ Entailment rule $R_{P \circ P \subseteq P}$:

if $P_{rdd}(x, y), P_{rdd}(y, z)$ then $P_{rdd}(x, z)$

▶ Spark programme $P_{P \circ P \subseteq P}$:

```python
while True do
    $P_{tmp} = P_{rdd}$.map(lambda $(x_p, y_p) : (y_p, x_p)$).join($P_{rdd}$)
    $P_{rdd}$.map(lambda $(y_k, (x_p, z_p)) : (x_p, z_p)$)

    if $P_{tmp}$.isEmpty() then break

$P_{rdd} = P_{rdd}$.union($P_{tmp}$)
end
```
Data caching in distributed memory

Iterative computation:

- **TransitiveProperty** $P \ (P \circ P \subseteq P)$.

  \[
  \text{subOrganisationOf} \circ \text{subOrganisationOf} \subseteq \text{subOrganisationOf} \quad (7)
  \]

- **Entailment rule** $\mathcal{R}_{P \circ P \subseteq P}$:
  
  \[
  \text{if } \ P_{\text{rdd}}(x, y), P_{\text{rdd}}(y, z) \ \text{then } P_{\text{rdd}}(x, z)
  \]

- **Spark programme** $\mathcal{P}_{P \circ P \subseteq P}$:

  ```
  while True do
    P_{tmp} = P_{\text{rdd}}.map(lambda (x_p, y_p) : (y_p, x_p)).join(P_{\text{rdd}})
    .map(lambda (y_k, (x_p, z_p)) : (x_p, z_p))
    P_{tmp}.cache()
    if P_{tmp}.isEmpty() then break
    P_{\text{rdd}} = P_{\text{rdd}}.union(P_{tmp})
  end
  ```
Data caching in distributed memory

- GraduateStudent_{rdd} will be used three times:

```
R_{GraduateStudent ⊑ Person} ↘
Person_{rdd}
```

```
R_{GraduateStudent ⊑ ∃ takesCourse.GraduateCourse} ↘
takesCourse_{rdd}
```

```
R_{GraduateStudent ⊑ Student} ↘
Student_{rdd}
```

**Figure:** Caching GraduateStudent_{rdd} for Repeated Usage
The advantage of using Spark (2)

More flexible job scheduling as compared to Hadoop:

Figure: Job Scheduling between Hadoop (left) and Spark (right)
DAG for parallelising reasoning

Consider Person $\cap \exists$ takesCourse.Course $\subseteq$ Student:

- $\mathcal{R}_{\text{Person} \cap \exists \text{takesCourse.Course} \subseteq \text{Student}}$:
  
  if Person$_{rdd}(x)$, takesCourse$_{rdd}(x, y)$, Course$_{rdd}(y)$
  
  then Student$_{rdd}(x)$

- $\mathcal{P}_{\text{Person} \cap \exists \text{takesCourse.Course} \subseteq \text{Student}}$:
  
  Student$_{tmp1} = \text{takesCourse}_{rdd}.\text{map}(\lambda (x_t, y_t) : (y_t, x_t))$

  .join(\text{Course}_{rdd}.\text{map}(\lambda y_c : (y_c, y_c)))

  .map(\lambda (y_k, (x_t, y_c)) : x_t))

  Student$_{tmp2} = \text{Student}_{tmp1}.\text{intersection(\text{Person}_{rdd})}$

  Student$_{rdd} = \text{Student}_{rdd}.\text{union(\text{Student}_{tmp2})}$
DAG for parallelising reasoning

Figure: DAG Scheduling for $\mathcal{R}_{\text{Person} \sqcap \exists \text{takesCourse}.\text{Course} \sqsupseteq \text{Student}}$
Optimising programme execution order

Executing job\textsubscript{a}, job\textsubscript{b} and job\textsubscript{c} before job\textsubscript{d} is the best order.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{dagscheduling.png}
\caption{DAG Scheduling for $R_{\text{Person}} \sqcap \exists \text{takesCourse. Course} \sqsubseteq \text{Student}$}
\end{figure}
Ordering Spark Programmes

Consider $P_1 \sqsubseteq P_2$, $P_2 \circ P_2 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_3$:

![Diagram](image)

**Figure:** Acyclic property hierarchy

How about considering an addition axiom $P_3 \equiv P_1^-$?

![Diagram](image)

**Figure:** Cyclic property hierarchy
# Table of Contents

- Introduction
- SPOWL Overview
- SPOWL Features
- Evaluation
- Summary
Evaluating SPOWL of reasoning materialisation

- **Evaluation environment**
  - A cluster of 9 machines running on a private cloud environment.
  - Each node with CPU @ 2.5GHz, 4 Cores, and 16 GB of Memory.

- **Benchmarking dataset LUBM**
  - LUBM-2000: about 270 million A-Box facts and 44GB in size.

- **Comparison system: WebPIE**
  - Using MapReduce as the computation framework.
  - Not using tableaux reasoners.
  - Not partitioning reasoning materialisation.
  - Compressing data before reasoning materialisation.
## Performance of reasoning materialisation

### Reasoning materialisation by SPOWL

<table>
<thead>
<tr>
<th>SPOWL</th>
<th>LUBM-400</th>
<th>LUBM-800</th>
<th>LUBM-1200</th>
<th>LUBM-1600</th>
<th>LUBM-2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Load</td>
<td>9m08s</td>
<td>20m30s</td>
<td>27m50s</td>
<td>41m20s</td>
<td>54m10s</td>
</tr>
<tr>
<td>Reasoning</td>
<td>10m19s</td>
<td>16m28s</td>
<td>33m20s</td>
<td>38m58s</td>
<td>58m08s</td>
</tr>
<tr>
<td>Total Time</td>
<td>19m27s</td>
<td>36m58s</td>
<td>1h01m10s</td>
<td>1h20m18s</td>
<td>1h52m18s</td>
</tr>
</tbody>
</table>

![Graph showing time comparison for different datasets]

---

Y. Liu & P. McBrien
BeyondMR17
Performance of reasoning materialisation

- **Reasoning materialisation by SPOWL**

<table>
<thead>
<tr>
<th>SPOWL</th>
<th>LUBM-400</th>
<th>LUBM-800</th>
<th>LUBM-1200</th>
<th>LUBM-1600</th>
<th>LUBM-2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Load</td>
<td>9m08s</td>
<td>20m30s</td>
<td>27m50s</td>
<td>41m20s</td>
<td>54m10s</td>
</tr>
<tr>
<td>Reasoning</td>
<td>10m19s</td>
<td>16m28s</td>
<td>33m20s</td>
<td>38m58s</td>
<td>58m08s</td>
</tr>
<tr>
<td>Total Time</td>
<td>19m27s</td>
<td>36m58s</td>
<td>1h01m10s</td>
<td>1h20m18s</td>
<td>1h52m18s</td>
</tr>
</tbody>
</table>

- **Reasoning materialisation by WebPIE**

<table>
<thead>
<tr>
<th>WebPIE</th>
<th>LUBM-1000</th>
<th>LUBM-2000</th>
<th>LUBM-3000</th>
<th>LUBM-4000</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>29m04s</td>
<td>59m37s</td>
<td>1h31m52s</td>
<td>2h01m59s</td>
</tr>
<tr>
<td>reasoning</td>
<td>30m36s</td>
<td>46m02s</td>
<td>58m27s</td>
<td>70m13s</td>
</tr>
<tr>
<td>decompress</td>
<td>14m03s</td>
<td>28m35s</td>
<td>49m16s</td>
<td>1h03m7s</td>
</tr>
<tr>
<td>Total</td>
<td>1h13m43s</td>
<td>2h14m14s</td>
<td>3h19m35s</td>
<td>4h15m19s</td>
</tr>
</tbody>
</table>
# Table of Contents

- Introduction
- SPOWL Overview
- SPOWL Features
- Evaluation
- Summary
Summary

- SPOWL: a compiler for translating OWL axioms to Spark programmes.
  - Combine tableaux reasoning and rule-based reasoning.
  - Partition reasoning materialisation.
  - Use Spark to implement entailment rules.
  - Optimise the order of executing Spark programmes.
  - Preliminary evaluation over LUBM datasets.