Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data

Maribel Acosta & Maria-Esther Vidal
Motivation (1)
Motivation

Mechanisms to access RDF data on the Web
Motivation (1)

Mechanisms to access RDF data on the Web
Motivation (1)

Mechanisms to access RDF data on the Web

Dereferencing

Expressivity

Low

High
Motivation (1)

Mechanisms to access RDF data on the Web

Dereferencing

Low

Expressivity

High

SPARQL Endpoint

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Motivation (1)

Mechanisms to access RDF data on the Web

Dereferencing  Triple Pattern Fragment Server  SPARQL Endpoint

Low  Expressivity  High

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Motivation (1)

Mechanisms to access RDF data on the Web

Dereferencing | Triple Pattern Fragment Server | SPARQL Endpoint

Low | Expressivity | High
Motivation (2)

Triple Pattern Fragment (TPF) Servers

Client

?s rdf:type yago:Alcohols

TPF Server
Motivation (2)

Triple Pattern Fragment (TPF) Servers

Client

?s rdf:type yago:Alcohols

TPF Server

Fragment:

{(dbr:Beraprost, rdf:type, yago:Alcohols),
 (dbr:Ethynol, rdf:type, yago:Alcohols),
 ...}

Metadata:

Count: 529, Pagesize: 100
Motivation (3)

Retrieve resources classified as DBpedia and YAGO alcohols that have the same route of administration.

```
SELECT * WHERE {
① ?d1 dcterms:subject dbc:Alcohols .
③ ?d1 dbp:routesOfAdministration ?o .
④ ?d2 dbp:routesOfAdministration ?o . }
```
Motivation (3)

Retrieve resources classified as DBpedia and YAGO alcohols that have the same route of administration.

```
SELECT * WHERE {
  ?d1 dcterms:subject dbc:Alcohols .
  ?d1 dbp:routesOfAdministration ?o .
  ?d2 dbp:routesOfAdministration ?o .
}
```

TPF Client

State-of-the-art

[Van Herwegen et al.]
Motivation (3)

Retrieve resources classified as DBpedia and YAGO alcohols that have the same route of administration.

```
SELECT * WHERE {
  1. ?d1 dcterms:subject dbc:Alcohols .
  3. ?d1 dbp:routesOfAdministration ?o .
  4. ?d2 dbp:routesOfAdministration ?o .}
```

Left-linear plan

TPF Server

TPF Client

State-of-the-art

[Van Herwegen et al.]

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Retrieve resources classified as DBpedia and YAGO alcohols that have the same route of administration.

```
SELECT * WHERE {
  ① ?d1 dcterms:subject dbc:Alcohols .
  ③ ?d1 dbp:routesOfAdministration ?o .
  ④ ?d2 dbp:routesOfAdministration ?o .
}
```

**Left-linear plan**

**TPF Client**

*State-of-the-art*

[Van Herwegen et al.]

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Motivation (3)

Retrieve resources classified as DBpedia and YAGO alcohols that have the same route of administration.

```sql
SELECT * WHERE {
  ① ?d1 dcterms:subject dbc:Alcohols .
  ③ ?d1 dbp:routesOfAdministration ?o .
  ④ ?d2 dbp:routesOfAdministration ?o .
}
```

Left-linear plan

TPF Client
State-of-the-art
[Van Herwegen et al.]

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Motivation

Left-linear plan

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Motivation

Left-linear plan

Execution Results

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Left-linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time (sec.)</td>
<td>318.90</td>
</tr>
<tr>
<td>Results</td>
<td>1,398</td>
</tr>
<tr>
<td>Requests</td>
<td>1,693</td>
</tr>
</tbody>
</table>

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Motivation (4)

**Left-linear plan**

```
2
| ?d2 |
```

```
4
| ?d2 |
```

```
3
| ?o |
```

```
1
| ?d1 |
```

```
529
```

```
2,430
```

```
695
```

**Bushy tree plan**

```
2
| ?d2 |
```

```
4
| ?d1 |
```

```
1
| ?o |
```

```
529
```

```
2,430
```

```
695
```

```
2,430
```

Execution Results

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Left-linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time (sec.)</td>
<td>318.90</td>
</tr>
<tr>
<td>Results</td>
<td>1,398</td>
</tr>
<tr>
<td>Requests</td>
<td>1,693</td>
</tr>
</tbody>
</table>
Motivation

Left-linear plan

Bushy tree plan

Execution Results

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Left-linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time (sec.)</td>
<td>318.90</td>
</tr>
<tr>
<td>Results</td>
<td>1,398</td>
</tr>
<tr>
<td>Requests</td>
<td>1,693</td>
</tr>
</tbody>
</table>
Motivation (4)

### Left-linear plan

![Left-linear plan diagram]

### Bushy tree plan

![Bushy tree plan diagram]

#### Execution Results

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Left-linear</th>
<th>Bushy tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time (sec.)</td>
<td>318.90</td>
<td>3.03</td>
</tr>
<tr>
<td>Results</td>
<td>1,398</td>
<td>5,651</td>
</tr>
<tr>
<td>Requests</td>
<td>1,693</td>
<td>67</td>
</tr>
</tbody>
</table>

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Motivation

**Left-linear plan**

```
  ?d2  ?o  ?d1
  2    3    1
  529  2,430 695
```

**Bushy tree plan**

```
  ?o  ?d1
  2    4
  529  2,430

  1
  695

  3
  2,430
```

**Nested Loop Join**

**Symmetric Hash Join**

**Execution Results**

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Left-linear</th>
<th>Bushy tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time (sec.)</td>
<td>318.90</td>
<td>3.03</td>
</tr>
<tr>
<td>Results</td>
<td>1,398</td>
<td>5,651</td>
</tr>
<tr>
<td>Requests</td>
<td>1,693</td>
<td>67</td>
</tr>
</tbody>
</table>

**Goal 1:**
Producing plans that are executed efficiently
Motivation (5)

Even efficient plans can be affected during query execution.
Even efficient plans can be affected during query execution.
Motivation (5)

Even efficient plans can be affected during query execution.

Unpredictable conditions:
- Data transfer delays
- Server workload
- Selectivity and data distribution

Adapt the plans to unpredictable conditions.

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Motivation (5)

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Motivation (5)

Source of 2 becomes slow
Motivation (5)

Source of 2 becomes slow
Motivation (5)

Source of 2 becomes slow
Motivation (5)

Source of 2 becomes slow

Source of 1 becomes slow
Motivation

Source of 2 becomes slow

Source of 1 becomes slow
Motivation (5)

Source of 2 becomes slow

Source of 1 becomes slow

Adapted plan
Motivation (5)

Source of 2 becomes slow

Source of 1 becomes slow

Adapted plan

Execution Results (Network with Delays)

Adapted plan: 3.86 seconds.

Fixed plan: 5.03 seconds.
**Motivation (5)**

Source of \( \textcircled{2} \) becomes slow

Source of \( \textcircled{1} \) becomes slow

Adapted plan

---

**Execution Results**

(Network with Delays)

Adapted plan: 3.86 seconds.

Fixed plan: 5.03 seconds.

---

**Goal 2:**

Adapting plans according to execution conditions
OUR APPROACH:
NETWORK OF LINKED DATA EDDIES
nLDE:
Network of Linked Data Eddies

• Client side query processing engine.

• SPARQL query optimizer for Triple Pattern Fragments.

• Opportunistically executes queries against TPF servers.
nLDE:
Network of Linked Data Eddies

• Client side query processing engine.

• SPARQL query optimizer for Triple Pattern Fragments.

Goal 1: Producing plans that are executed efficiently.

• Opportunistically executes queries against TPF servers.
nLDE:
Network of Linked Data Eddies

• Client side query processing engine.

• SPARQL query optimizer for Triple Pattern Fragments.

Goal 1: Producing plans that are executed efficiently.

Goal 2: Adapting plans according to execution conditions.
nLDE Architecture

SPARQL Query $Q$

Input

Query Optimizer

Metadata

Optimized plan

Routing Policies

Adaptive Engine

TPF Server

Results for $Q$

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
nLDE Architecture

Goal 1: Producing plans that are executed efficiently

SPARQL Query $Q$

Input

Query Optimizer

Metadata

Optimized plan

Routing Policies

Adaptive Engine

TPF Server

Output

Results for $Q$

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
nLDE Query Optimizer

• Generates bushy tree plans

• Steps:
  1. Retrieve triple pattern metadata
  2. Build star-shaped groups [Vidal et al.]
  3. Place physical operators
  4. Combine star-shaped groups

Goal 1: Producing plans that are executed efficiently
1. Retrieve triple pattern metadata

```
SELECT * WHERE {

① ?d1 dcterms:subject dbr:Category:Alcohols.
③ ?d1 dbp:routesOfAdministration ?o .
④ ?d2 dbp:routesOfAdministration ?o .}
```
1. Retrieve triple pattern metadata

```
SELECT * WHERE {
  ① ?d1 dcterms:subject dbr:Category:Alcohols.
  ③ ?d1 dbp:routesOfAdministration ?o .
  ④ ?d2 dbp:routesOfAdministration ?o .
}
```

Metadata

# Count: 695
# Count: 529
# Count: 2430
# Count: 2430

Pagesize: 100
2. Build star-shaped groups (1)

```
SELECT * WHERE {
  1 ?d1 dcterms:subject dbr:Category:Alcohols.
  3 ?d1 dbp:routesOfAdministration ?o .
  4 ?d2 dbp:routesOfAdministration ?o .}
```

Query Plan:

```
Metadata

# Count:  695
# Count:  529
# Count:  2430
# Count:  2430
Pagesize:  100
```
2. Build star-shaped groups (1)

```
SELECT * WHERE {
  3. ?d1 dbp:routesOfAdministration ?o .
  4. ?d2 dbp:routesOfAdministration ?o .}
```

Query Plan:

```
Pagesize: 100
```

Metadata

```
# Count: 695
# Count: 529
# Count: 2430
# Count: 2430
Pagesize: 100
```
2. Build star-shaped groups (1)

SELECT * WHERE {
  ③ ?d1 dbp:routesOfAdministration ?o .
  ④ ?d2 dbp:routesOfAdministration ?o .}

Query Plan:

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
2. Build star-shaped groups (2)

```
SELECT * WHERE {
  ① ?d1 dcterms:subject dbr:Category:Alcohols.
  ③ ?d1 dbp:routesOfAdministration ?o  .
  ④ ?d2 dbp:routesOfAdministration ?o  .}
```

Query Plan:

Metadata

- # Count: 695
- # Count: 529
- # Count: 2430
- # Count: 2430

Pagesize: 100
2. Build star-shaped groups (2)

```
SELECT * WHERE {
  3. ?d1 dbp:routesOfAdministration ?o .
  4. ?d2 dbp:routesOfAdministration ?o .}
```

Query Plan:

```
Metadata
# Count:   695
# Count:   529
# Count:  2430
# Count:  2430
Pagesize:   100
```
2. Build star-shaped groups (2)

**Select** * WHERE {

1. \(?d1\) dcterms:subject dbr:Category:Alcohols.
2. \(?d2\) rdf:type yago:Alcohols.
3. \(?d1\) dbp:routesOfAdministration \(?o\).
4. \(?d2\) dbp:routesOfAdministration \(?o\).}

**Query Plan:**

**Metadata**

- # Count: 695
- # Count: 529
- # Count: 2430
- # Count: 2430

**Pagesize:** 100
3. Place physical operator

**Query Plan:**

- Nested Loop Join
- Symmetric Hash Join

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
3. Place physical operator

Count

\( \begin{align*}
\text{\%2}\text{Requires} & 529 \div 100 = 6 \text{ requests} \\
\text{Pagesize} & \\
\text{\%4}\text{Requires} & 2430 \div 100 = 25 \text{ requests} 
\end{align*} \)

Query Plan:

- Nested Loop Join
- Symmetric Hash Join

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
3. Place physical operator

Count

\[ \text{Pagesize} \]

\[ \frac{529}{100} \]

\[ 6 \text{ requests} \]

\[ \frac{2430}{100} \]

\[ 25 \text{ requests} \]

**Query Plan:**

- **Nested Loop Join**
  - Requires \[ \frac{529}{100} \] = 6 requests
  - Produces 6 + 529 = 535 requests

- **Symmetric Hash Join**
  - Requires \[ \frac{2430}{100} \] = 25 requests
  - Produces 6 + 25 = 31 requests
3. Place physical operator

Count

\( \text{Nested Loop Join} \)

Produces \( 6 + 529 = 535 \) requests

\( \text{Symmetric Hash Join} \)

Produces \( 6 + 25 = 31 \) requests

\( \text{Page size} \)

\( \frac{529}{100} \) = 6 requests

\( \frac{2430}{100} \) = 25 requests

Query Plan:

- Nested Loop Join
- Symmetric Hash Join
3. Place physical operator

- **Count**
  - Requires \(\frac{529}{100} = 6\) requests

- **Pagesize**
  - \(\frac{2430}{100} = 25\) requests

**Query Plan:**

- **Nested Loop Join**
  - Produces \(6 + 529 = 535\) requests

- **Symmetric Hash Join**
  - Produces \(6 + 25 = 31\) requests
4. Combine star-shaped groups

```sql
SELECT * WHERE {
    ① ?d1 dcterms:subject dbr:Category:Alcohols .
    ③ ?d1 dbp:routesOfAdministration ?o .
    ④ ?d2 dbp:routesOfAdministration ?o .
}
```

Query Plan:

- Nested Loop Join
- Symmetric Hash Join

Pagesize: 100

# Count: 695
# Count: 529
# Count: 2430
# Count: 2430
4. Combine star-shaped groups

```
SELECT * WHERE {
  3. ?d1 dbp:routesOfAdministration ?o .
}
```

Query Plan:

- Nested Loop Join
- Symmetric Hash Join

Metadata:

- # Count: 695
- # Count: 529
- # Count: 2,430
- # Count: 2,430
- Pagesize: 100
4. Combine star-shaped groups

```
SELECT * WHERE {
    3. ?d1 dbp:routesOfAdministration ?o .
    4. ?d2 dbp:routesOfAdministration ?o .}
```

Query Plan:

- Nested Loop Join
- Symmetric Hash Join

Metadata

- # Count: 695
- # Count: 529
- # Count: 2430
- # Count: 2430
- Pagesize: 100
4. Combine star-shaped groups

```
SELECT * WHERE {
  ?d1 dcterms:subject dbr:Category:Alcohols .
  ?d1 dbp:routesOfAdministration ?o .
  ?d2 dbp:routesOfAdministration ?o .
}
```

Query Plan:

- Nested Loop Join
- Symmetric Hash Join

```
Pagesize: 100
# Count: 695
# Count: 529
# Count: 2430
# Count: 2430
```

Metadata

- Optimized physical plan tailored for TPFs.

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
nLDE Architecture

SPARQL Query $Q$

Input

Query Optimizer

Metadata

Optimized plan

Routing Policies

Adaptive Engine

Output

TPF Server

Results for $Q$
nLDE Architecture

Goal 2:
Adapting plans according to execution conditions

SPARQL Query $Q$

Input

Query Optimizer

Metadata

Optimized plan

Routing Policies

Adaptive Engine

TPF Server

Output

Results for $Q$

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
nLDE Adaptive Engine

- Executes the optimized plan.

- Performs two types of **adaptivity**:
  - **Intra-operator**: Produce results incrementally.
  - **Routing operator**: The order of the plan is changed on-the-fly.

- Adaptivity is performed on a **tuple-based** basis.

- Components: Adaptive Operators + Eddy Operators

**Goal 2:**
Adapting plans according to execution conditions
Adaptive Operator

- Physical operator able to produce tuples incrementally.

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Adaptive Operator

- Physical operator able to produce tuples incrementally.
Eddy Operator (1)

[Avnur & Hellerstein]

- Operator that dynamically routes tuples through operators.
Eddy Operator (1) [Avnur & Hellerstein]

- Operator that dynamically routes tuples through operators.
Eddy Operator (2)

- Operator that dynamically routes tuples through operators.
- To produce **sound results**, an eddy relies on tuple annotations for routing.
Eddy Operator (2)

• Operator that dynamically routes tuples through operators.
• To produce **sound results**, an eddy relies on tuple annotations for routing.

**Ready vector**
Indicates operators eligible to process a tuple. For example, tuples $t$ from ① should be processed by operators 0 and 2, but not by operator 1:

$$\text{Ready}_t = 101$$
Eddy Operator

- Operator that dynamically routes tuples through operators.
- To produce **sound results**, an eddy relies on tuple annotations for routing.

**Ready vector**
Indicates operators eligible to process a tuple.
For example, tuples $t$ from \( 1 \) should be processed by operators 0 and 2, but not by operator 1:

$$\text{Ready}_t = 101$$

**Done vector**
Indicates operators that have processed a tuple.
For example, a tuple $t$ has been processed by operator 1:

$$\text{Done}_t = 010$$
Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Several eddies can be part of an nLDE to distribute the workload.
Several eddies can be part of an nLDE to distribute the workload.
Each eddy is autonomous.
Several eddies can be part of an nLDE to distribute the workload.
Each eddy is autonomous.
An nLDE maintains information about triple patterns from the query to select adaptive operators in the Triple Pattern Descriptor.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>?d1</td>
<td>dcterms:subject</td>
<td>dbpedia:Category:Alcohols</td>
<td>Count = 695</td>
</tr>
<tr>
<td>?d2</td>
<td>rdf:type</td>
<td>yago:Alcohols</td>
<td>Count = 529</td>
</tr>
<tr>
<td>?d1</td>
<td>dbprop:routeOfAdministation</td>
<td>?o</td>
<td>Count = 2430</td>
</tr>
<tr>
<td>?d2</td>
<td>dbprop:routeOfAdministation</td>
<td>?o</td>
<td>Count = 2430</td>
</tr>
</tbody>
</table>

Triple Pattern Descriptor (TD)
nLDE Architecture

SPARQL Query $Q$

Input

Query Optimizer

Metadata

Optimized plan

Routing Policies

Adaptive Engine

TPF Server

Results for $Q$
Routing Policy: From Eddy to Adaptive Operator

- Each eddy $e$ inserts incoming tuples into a Routing Buffer.

Routing Policy (RP$_e$)

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Ready</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {d1=dbpedia:Avanagil, d2=Beraprost, o=&quot;Oral&quot; }</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>2 {d1=dbpedia:Bupranolol, o=&quot;Oral, topical&quot;}</td>
<td>101</td>
<td>001</td>
</tr>
<tr>
<td>3 {d2=dbpedia:Ethynol}</td>
<td>110</td>
<td>000</td>
</tr>
</tbody>
</table>

Routing Buffer (RB$_e$)
Routing Policy: From Eddy to Adaptive Operator

- Each eddy $e$ inserts incoming tuples into a **Routing Buffer**.

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Ready</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ({d1=\text{dbpedia:Avanagil}, , d2=\text{Beraprost}, , o=\text{“Oral”}})</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>2 ({d1=\text{dbpedia:Bupranolol}, , o=\text{“Oral, topical”}})</td>
<td>101</td>
<td>001</td>
</tr>
<tr>
<td>3 ({d2=\text{dbpedia:Ethynol}})</td>
<td>110</td>
<td>000</td>
</tr>
</tbody>
</table>

Routing Buffer (RB$_e$)

Routing Policy (RP$_e$)

All Done entries ON

Output

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Routing Policy: From Eddy to Adaptive Operator

- Each eddy \( e \) inserts incoming tuples into a **Routing Buffer**.
- \( \text{Ready}_t – \text{Done}_t \) determines the operators that still need to be executed.

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Ready</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {d1=dbpedia:Avanagil, d2=Beraprost, o=“Oral” }</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>2 {d1=dbpedia:Bupranolol, o=“Oral, topical”}</td>
<td>101</td>
<td>001</td>
</tr>
<tr>
<td>3 {d2=dbpedia:Ethynol}</td>
<td>110</td>
<td>000</td>
</tr>
</tbody>
</table>

Routing Buffer (\( \text{RB}_e \))

Routing Policy (\( \text{RP}_e \))

- All \( \text{Done} \) entries ON
- Eligible operators 101
- 001= 100

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Routing Policy: From Eddy to Adaptive Operator

- Each eddy $e$ inserts incoming tuples into a **Routing Buffer**.
- $\text{Ready}_t – \text{Done}_t$ determines the operators that still need to be executed.
- If several operators are eligible, choose one with the **highest priority**:

$$\text{priority}(i) = \frac{\#\text{tuples received from } i}{\#\text{tuples routed to } i}$$

Estimates operator selectivity

Combined with SPARQL Heuristics [Tsialimanis et al.]

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Ready</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {d1=dbpedia:Avanagil, d2=Beraprost, o=&quot;Oral&quot; }</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>2 {d1=dbpedia:Bupranolol , o=&quot;Oral, topical&quot;}</td>
<td>101</td>
<td>001</td>
</tr>
<tr>
<td>3 {d2=dbpedia:Ethynol}</td>
<td>110</td>
<td>000</td>
</tr>
</tbody>
</table>

Routing Buffer ($\text{RB}_e$)

Routing Policy ($\text{RP}_e$)

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
EXPERIMENTAL RESULTS
Experimental Settings

• TPF Server for the English version of DBpedia

• 45 SPARQL Queries:
  • Benchmark 1: 20 non-selective queries (4 – 14 triple patterns)
  • Benchmark 2: 25 selective queries (3 – 6 triple patterns)

• Implementation details:
  • nLDE was implemented in Python 2.7.6.
  • Timeout was set to 1,800 secs.

• Technical specifications: Debian Wheezy 64 bit with CPU 2x Intel(R) Xeon(R) CPU E5-2670 2.60GHz (16 physical cores), and 256GB RAM.

• Each query was executed 10 times.
Effectiveness of nLDE Optimization Techniques: Benchmark 1 (non-selective)

Plans generated by nLDE reduce execution time.
Effectiveness of nLDE Optimization Techniques: Benchmark 1 (non-selective)
Effectiveness of nLDE Optimization Techniques: Benchmark 1 (non-selective)

nLDE is able to produce the same amount or more answers than state-of-the-art.
Effectiveness of nLDE Optimization Techniques: Benchmark 2 (selective)

Plans generated by nLDE reduce execution time, even for selective queries.
Effectiveness of nLDE Optimization Techniques: Benchmark 2 (selective)

In the majority of the queries, nLDE minimizes the number of requests submitted to the server.
③ Adaptivity of the nLDE Engine

Network delays following a Gamma distribution ($\alpha=1; \beta=0.3$).

nLDE adapted plans are able to produce complete results faster than fixed plans.

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
RELATED WORK
Summary of Related Work

SPARQL query processing approaches over HTTP

**Linked Data Traversal**
[Hartig et al.]

Plans cannot be adapted to unexpected data distributions

**TPF Client**
[Van Herwegen et al.]

Plans cannot be adapted to unexpected transfer delays

**Federated Engines**
- ANAPSID [Acosta et al.]
- SPLENDID [Görtlitz & Staab]
- FedEx [Schwarte et al.]

Optimize-then-execute

Dereferencing

Triple Pattern Fragment Server

SPARQL Endpoint

Expressivity

Low

High

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
CONCLUSIONS & FUTURE WORK
nLDE: Client-side query engine against TPF servers.

- **Research contributions** of this work:
  - Optimization techniques tailored for TPFs.
  - Autonomous eddies.
  - Routing policies tailored for SPARQL queries.

- Experiments confirmed that nLDE:
  - Overcomes state-of-the-art
  - Adapts execution to unexpected delays
    - Execution time
    - Number of requests
    - Number of answers

Conclusions
Future Work

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
Future Work

1. Define cost models to estimate the selectivity of TPF.
Future Work

1. Define cost models to estimate the selectivity of TPF.

2. Study different routing policies.
Future Work

1. Define cost models to estimate the selectivity of TPF.

2. Study different routing policies.

3. Application to federations.

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
References


Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data

Overcomes state-of-the-art

- \(\downarrow\) Execution time
- \(\downarrow\) Number of requests
- \(\uparrow\) Number of answers

Adapts execution to unexpected delays

Maribel Acosta & Maria-Esther Vidal
THANK YOU

Network of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data – Acosta and Vidal
3. Place physical operator

Query Plan:

- Nested Loop Join
- Symmetric Hash Join
3. Place physical operator

Count

1 Requires \[ \frac{695}{100} \] = 7 requests

Pagesize

Query Plan:

- Nested Loop Join
- Symmetric Hash Join
3. Place physical operator

Count

\[\text{Page size} \times \frac{695}{100} = 7 \text{ requests}\]

\[\text{Page size} \times \frac{2430}{100} = 25 \text{ requests}\]

Query Plan:

- Nested Loop Join
- Symmetric Hash Join
3. Place physical operator

Count

1. Requires \( \frac{695}{100} \) = 7 requests

Pagesize

3. Requires \( \frac{2430}{100} \) = 25 requests

Nested Loop Join

Produces \( 7 + 695 = 702 \) requests

Query Plan:

Nested Loop Join

Symmetric Hash Join
3. Place physical operator

**Count**

1. Requires \[ \frac{695}{100} \] = 7 requests

2. Requires \[ \frac{2430}{100} \] = 25 requests

**Nested Loop Join**
Produces \[ 7 + 695 = 702 \] requests

**Symmetric Hash Join**
Produces \[ 7 + 25 = 32 \] requests

**Query Plan:**

- Nested Loop Join
- Symmetric Hash Join
3. Place physical operator

Count

1. Requires \( \frac{695}{100} \) = 7 requests

Pagesize

3. Requires \( \frac{2430}{100} \) = 25 requests

Nested Loop Join

Produces \( 7 + 695 = 702 \) requests

Symmetric Hash Join

Produces \( 7 + 25 = 32 \) requests

Query Plan:
Routing Policies (1)

Routing Policy: From Eddy to Adaptive Operator

Routing Policy: From Adaptive Operator to Eddy
Eddy Operator (3)

- Operator that dynamically routes tuples through operators.
- To produce **sound results**, an eddy relies on tuple annotations for routing.
- Each eddy $e$ inserts incoming tuples into a **Routing Buffer**.

<table>
<thead>
<tr>
<th>Tuple</th>
<th>Ready</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>{d1=dbpedia:Avanagil, d2=Beraprost, o=“Oral”}</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>{d1=dbpedia:Bupranolol, o=“Oral, topical”}</td>
<td>101</td>
<td>001</td>
</tr>
<tr>
<td>{d2=dbpedia:Ethynol}</td>
<td>110</td>
<td>000</td>
</tr>
<tr>
<td>{d1=dbpedia:Carteolol}</td>
<td>101</td>
<td>000</td>
</tr>
<tr>
<td>{d1=dbpedia:Atenolol, o=“Oral or IV”}</td>
<td>101</td>
<td>000</td>
</tr>
</tbody>
</table>

Routing Buffer ($RB_e$)
Before sending a tuple $t$ to an eddy, the operator computes the \textit{Ready} and \textit{Done} vectors of $t$.

The adaptive operator randomly chooses an eddy following a uniform distribution.

The adaptive operator randomly chooses an eddy following a \textbf{uniform distribution}.
Before sending a tuple $t$ to an eddy, the operator computes the \textit{Ready} and \textit{Done} vectors of $t$.

\begin{itemize}
  
  \item The adaptive operator randomly chooses an eddy following a uniform distribution.

  \begin{itemize}
  
    \item \{d1=dbr:Avanagil, o=“Oral”\}
    \item \{d2=dbr:Beraprost, o=“Oral”\}
    \item \{d1=dbr:Avanagil, d2=dbr:Beraprost, o=“Oral”\}

  \end{itemize}

\end{itemize}
Routing Policy: From Adaptive Operator to Eddy

- Before sending a tuple $t$ to an eddy, the operator computes the \textit{Ready} and \textit{Done} vectors of $t$.

$$\text{Ready}_{ti} = 101, \quad \text{Done}_{ti} = 001$$

$$\{d1=dbr:Avanagil, o=\text{"Oral"}\}$$

$$\{d2=dbr:Beraprost, o=\text{"Oral"}\}$$

$$\text{Ready}_{tj} = 110, \quad \text{Done}_{tj} = 010$$

- The adaptive operator randomly chooses an eddy following a \textit{uniform distribution}.
Routing Policy: From Adaptive Operator to Eddy

- Before sending a tuple \( t \) to an eddy, the operator computes the *Ready* and *Done* vectors of \( t \).

\[
\text{Ready}_{ti} = 101, \quad \text{Done}_{ti} = 001
\]
\[
\{d1=dbr:Avanagil, \ o=\text{“Oral”}\}
\]

\[
\text{Ready}_{tj} = 110, \quad \text{Done}_{tj} = 010
\]
\[
\{d2=dbr:Beraprost, \ o=\text{“Oral”}\}
\]

\[
\begin{align*}
\text{Ready}_t &= 101 \text{ OR } 110 = 111 \\
\text{Done}_t &= 001 \text{ OR } 010 \text{ OR } 100 = 111
\end{align*}
\]

- The adaptive operator randomly chooses an eddy following a uniform distribution.