Exploiting Back-End APIs for Feasible Ontology-Based Stream Access

Fourth Stream Reasoning Workshop, Linköping, 17th April
Institute of Information Systems
University of Lübeck
OBDA on huge datasets (w/o optimization) ⇒ Long processing times
OBDA on huge datasets (w/o optimization) ⇒ Long processing times

Processing times are reducible using additional hardware and parallelisation ⇒ Short processing times

Horizontal scaling

Q_SQL

Q_STARQL

Q_SparkSQL
STARQL Query Example

Measurements

Information need for monotonicity

Tell every minute whether the temperature measured by a sensor increased monotonically in the last 5 minutes.
STARQL Query Example

Measurements

<table>
<thead>
<tr>
<th>Time/min</th>
<th>Temp/C°</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>90.0</td>
</tr>
<tr>
<td>1.0</td>
<td>91.0</td>
</tr>
<tr>
<td>2.0</td>
<td>92.0</td>
</tr>
<tr>
<td>3.0</td>
<td>93.0</td>
</tr>
<tr>
<td>4.0</td>
<td>94.0</td>
</tr>
<tr>
<td>5.0</td>
<td>94.0</td>
</tr>
<tr>
<td>6.0</td>
<td>93.0</td>
</tr>
</tbody>
</table>

STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
   IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
   AND t_i < t_j THEN y1 <= y2
```
STARQL Query Example

Measurements

<table>
<thead>
<tr>
<th>Time/min</th>
<th>Temp/C°</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>90.0</td>
</tr>
<tr>
<td>1.0</td>
<td>91.0</td>
</tr>
<tr>
<td>2.0</td>
<td>92.0</td>
</tr>
<tr>
<td>3.0</td>
<td>93.0</td>
</tr>
<tr>
<td>4.0</td>
<td>94.0</td>
</tr>
</tbody>
</table>

STARQL Representation of monotonicity

```sql
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
   IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
   AND t_i < t_j THEN y1 <= y2
```
STARQL Query Example

Measurements

<table>
<thead>
<tr>
<th>Time/min</th>
<th>Temp/°C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>90.0</td>
</tr>
<tr>
<td>1.0</td>
<td>91.0</td>
</tr>
<tr>
<td>2.0</td>
<td>92.0</td>
</tr>
<tr>
<td>3.0</td>
<td>93.0</td>
</tr>
<tr>
<td>4.0</td>
<td>94.0</td>
</tr>
<tr>
<td>5.0</td>
<td>95.0</td>
</tr>
<tr>
<td>6.0</td>
<td>96.0</td>
</tr>
</tbody>
</table>

STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y_1, y_2
    IF hasVal(x,y_1)<t_i> AND hasVal(x,y_2)<t_j>
    AND t_i < t_j THEN y_1 <= y_2
```
STARQL Query Example

Measurements

```
STARQL Representation of monotonicity
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
  IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
  AND t_i < t_j THEN y1 <= y2
```
STARQL Query Example

### Measurements

<table>
<thead>
<tr>
<th>Time/min</th>
<th>Temp/C°</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>90.0</td>
</tr>
<tr>
<td>1.0</td>
<td>91.0</td>
</tr>
<tr>
<td>2.0</td>
<td>92.0</td>
</tr>
<tr>
<td>3.0</td>
<td>93.0</td>
</tr>
<tr>
<td>4.0</td>
<td>94.0</td>
</tr>
</tbody>
</table>

### STARQL Representation of monotonicity

```sql
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
  IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
  AND t_i < t_j THEN y1 <= y2
```
STARQL Query Example

STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
    IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
    AND t_i < t_j THEN y1 <= y2
```

The FOL template language is domain independent\(^1\):

STARQL HAVING clause can be unfolded into languages such as SQL.

\(\Rightarrow\) Process historic (e.g. timestamped datasets)

STARQL Query Example

**STARQL Representation of monotonicity**

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL \( t_i, t_j, y_1, y_2 \)
  IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
  AND \( t_i < t_j \) THEN y_1 \leq y_2
```

No function exists for executing the unfolded query per window!

- First idea: Create table with window intervals and join with historic dataset.

- Too slow?
STARQL Query Example

STARQL Representation of monotonicity

SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
  IF hasVal(x, y1) < t_i AND hasVal(x, y2) < t_j
  AND t_i < t_j THEN y1 <= y2

No function exists for executing the unfolded query per window!

▶ Second idea: Create a function for executing the unfolded query per window using PL/pgSQL.
▶ Sufficient?
STARQL Query Example

STARQL Representation of monotonicity

```
SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
   IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
   AND t_i < t_j THEN y1 <= y2
```

No function exists for executing the unfolded query per window!

- Second idea: Create a function for executing the unfolded query per window using PL/pgSQL.
- Sufficient?
CREATE TYPE window_state AS (memory measurements[],
    wid bigint, start timestamp, stop timestamp, pulse timestamp);

CREATE TABLE measurements_data AS SELECT NULL::bigint
    AS wid, NULL::timestamp AS timestamp, NULL::
    integer AS sensor, NULL::numeric(12,3) AS value
    WHERE false;

CREATE OR REPLACE FUNCTION moving_window (source text,
    pulse interval, range interval, slide interval)
    RETURNS SETOF measurements_data AS $$

DECLARE
    win window_state;
    cnt bigint;
    line_cursor refcursor;
    line measurements;
BEGIN
    OPEN line_cursor FOR EXECUTE 'SELECT * FROM ' format
        ('%1$s', source) ' ORDER BY timestamp ASC';
    FETCH line_cursor INTO line;

    win.start := line.timestamp;
$$
win.stop := line.timestamp + range;
win.pulse := line.timestamp;

WHILE line.timestamp < win.stop LOOP
  win.memory := array_append(win.memory, line);
  FETCH line_cursor INTO line;
END LOOP;

win.wid := 0;
RETURN QUERY SELECT win.wid, (unnest(win.memory::measurements[])).*;

win.pulse := win.pulse + pulse;
WHILE line.timestamp IS NOT NULL LOOP
  IF win.pulse < win.stop AND win.pulse < win.start + slide THEN
    win.wid := win.wid + 1;
    RETURN QUERY SELECT win.wid, (unnest(win.memory::measurements[])).*;
    win.pulse := win.pulse + pulse;
  ELSIF win.pulse >= win.stop AND win.pulse < win.start + slide THEN
    win.pulse := win.pulse + pulse;
    win.start := win.start + slide;
STARQL Query Example

```
39    win.stop := win.stop + slide;
40    WHILE line.timestamp < win.stop LOOP
41        win.memory := array_append(win.memory, line);
42        FETCH line_cursor INTO line;
43    END LOOP;
44    cnt := 1;
45    FOR i IN coalesce(array_lower(win.memory, 1), 1) ..
          coalesce(array_upper(win.memory, 1), 1) LOOP
46        IF win.memory[i].timestamp < win.start THEN
47            cnt := cnt + 1;
48        ELSE
49            EXIT;
50        END IF;
51    END LOOP;
52    win.memory := win.memory[cnt:];
53    ELSIF win.pulse >= win.start + slide THEN
54        win.start := win.start + slide;
55        win.stop := win.stop + slide;
56    WHILE line.timestamp < win.stop LOOP
57        win.memory := array_append(win.memory, line);
58        FETCH line_cursor INTO line;
59    END LOOP;
60    cnt := 1;
```
FOR i IN coalesce(array_lower(win.memory, 1), 1) ..
    coalesce(array_upper(win.memory, 1), 1) LOOP
    IF win.memory[i].timestamp < win.start THEN
        cnt := cnt + 1;
        ELSE
            EXIT;
        END IF;
    END LOOP;
    win.memory := win.memory[cnt:];
END IF;
END LOOP;
win.wid := win.wid + 1;
RETURN QUERY SELECT win.wid, (unnest(win.memory::
    measurements[])).*;
CLOSE line_cursor;
END
$$ language plpgsql;
STARQL Query Example

STARQL Representation of monotonicity

SELECT x
FROM measurements [NOW - PT5M, NOW] -> PT1M
WHERE Sensor(x)
HAVING FORALL t_i, t_j, y1, y2
  IF hasVal(x,y1)<t_i> AND hasVal(x,y2)<t_j>
  AND t_i < t_j THEN y1 <= y2

Function exists for executing the unfolded query per window!

- Apache Spark SQL scales horizontally and vertically.
- Scalable?
Processing times using different back ends

![Graph showing processing times for PostgreSQL, PL/pgSQL, and Apache Spark SQL over measurement durations ranging from 0 to 16 days. The graph illustrates the trend of processing time in minutes as a function of measurement duration in days.]
Processing times using different back ends

![Graph showing processing times using different back ends.

- **PL/pgSQL**
- **Apache Spark SQL**

The graph plots processing time in minutes against measurement duration in days.
Conclusion and Future Work

Conclusion:

▶ Window function can be realized by using PL/pgSQL
▶ Speed gain by using Apache Spark SQL
▶ Complexity hidden by STARQL

Future Work:

▶ Incremental stream processing (Not possible for every STARQL query)

---

Processing times using distributed Apache Spark SQL

![Graph showing processing times for different numbers of nodes over measurement durations in years.]