RQL: Retrospective Query Language

Nikos Tsikoudis
ntsikudis@brandeis.edu

Liuba Shrira
liuba@brandeis.edu

Brandeis University

Retrospection

Run over data store’s past state as if it were the current state

- No lightweight data store supports retrospection yet
  - Proposed algorithms require major changes to data store
  - Preserving performance is challenging

Our goal is to provide across time retrospection analysis efficiently

- Easy to integrate to any transaction store
- Application’s code base can be reused for retrospective analysis
- Avoid slowing down programs that do not use retrospection
- Provide good performance of retrospective queries

Auditing & Trend Analysis Applications

Background

- **Retro**: modular snapshot system in Berkeley DB
  - supports single snapshot retrospective queries

  Retrospective query:
  
  ```sql
  SELECT AS OF snapshot_id FROM table_name;
  ```

- Snapshots are persisted incrementally using copy-on-write (COW), without reorganizing data store

- Persistent data structure Maplog is created at low cost and indexes snapshot pages

- Retrospection requires logical data store page names to be translated to the location of persistent snapshot pages

- **Snapshot Page Tables (SPTs)**: in memory indexes that are used at run time and map logical page names to disk locations

Supported Queries

- **Past:**
  - Query over single consistent snapshot
    - The price of a product as of a specific day in the past

  - Ongoing / Contributions:
    - Collect data of interest from an interval of consistent snapshots
      - Orders from customers placed last year while living in Boston
    - Aggregate over snapshot intervals
      - Value of unshipped orders for each day in the previous year
      - Maximum price of a product last year

RQL: Retrospective Model and Language

- New retrospective SQL language
  - Implemented using SQL UDF

- **RQL Mechanisms:**
  - Let Qq be a retrospective query and Qs a query that returns an interval of snapshots
  - **Collate Data(Qq, Qs)**
    - Collects tuples returned by the retrospective query Qq on snapshots returned by Qs and stores them in a temporary table
  - **Aggregate in Variable(Qq, Qs, BFunct)**
    - Aggregates over Qs a single value returned by Qq using the function BFunct
  - **Aggregate in Table(Qq, Qs, ListOfColFuncPairs)**
    - Aggregates over Qs multiple values returned by Qq using the functions in ListOfColFuncPairs

Experimental Evaluation

RQL implementation in SQLite Retro BDB

Application: TPC-H retrospective extension

Optimizations for building indexes over snapshots:

- On demand 1
  - Fill SPTs dynamically
  - Fetch blocks of mappings from Maplog, scan until missing mapping is found

- On demand 2 - scan the entire block
  - Like on demand 1 but scan the entire block of mappings

- Prebuild SPTs for the entire Qs
  - Fill complete SPTs for the entire snapshot range

- **Sparse Queries**

  ```sql
  SELECT collate_data(snapid, "SELECT AS OF $snapid$ o_orderstatus FROM orders WHERE o_orderkey = 1;") FROM snapshotIds WHERE snapid > 0 AND snapid < 51;
  ```

- **Dense Queries**

  ```sql
  SELECT aggregate_variable(snapid, "SELECT AS OF $snapid$ AVG(o_totalprice) FROM orders", "MAX") FROM snapshotIds WHERE snapid > 0 AND snapid < 51;
  ```

Ongoing & Future Work

- Developing a retrospective query optimization framework:
  - Optimization techniques for building the SPT indexes
  - Prefetching techniques for expediting historical snapshot index

<table>
<thead>
<tr>
<th>Ongoing &amp; Future Work</th>
<th>Prebuild</th>
<th>On demand 1</th>
<th>On demand 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total exec time (sec)</td>
<td>15.54</td>
<td>20.33</td>
<td>26.61</td>
</tr>
<tr>
<td>Mapping collection time (sec)</td>
<td>15.37</td>
<td>18.68</td>
<td>19.68</td>
</tr>
<tr>
<td># blocks fetched</td>
<td>100</td>
<td>50</td>
<td>6</td>
</tr>
</tbody>
</table>

- **Sparse Queries**

  ```sql
  SELECT collate_data(snapid, "SELECT AS OF $snapid$ o_custkey, count(*) FROM orders GROUP BY o_custkey;") FROM snapshotIds WHERE snapid > 0 AND snapid < 51;
  ```

- **Dense Queries**

  ```sql
  SELECT aggregate_variable(snapid, "SELECT AS OF $snapid$ AVG(o_custkey) FROM orders", "MAX") FROM snapshotIds WHERE snapid > 0 AND snapid < 51;
  ```

- **Sparse Queries**

  ```sql
  SELECT aggregate_table(snapid, "SELECT AS OF $snapid$ o_orderstatus, count(*) AS c FROM orders", "(c:MAX)" FROM snapshotIds WHERE snapid > 0 AND snapid < 51;
  ```

- **Dense Queries**

  ```sql
  SELECT aggregate_table(snapid, "SELECT AS OF $snapid$ AVG(o_totalprice) FROM orders", "MAX") FROM snapshotIds WHERE snapid > 0 AND snapid < 51;
  ```

<table>
<thead>
<tr>
<th>Ongoing &amp; Future Work</th>
<th>Prebuild</th>
<th>On demand 1</th>
<th>On demand 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total exec time (sec)</td>
<td>90.75</td>
<td>641.04</td>
<td>826.13</td>
</tr>
<tr>
<td>Mapping collection time (sec)</td>
<td>35.88</td>
<td>26.51</td>
<td>26.48</td>
</tr>
<tr>
<td># blocks fetched</td>
<td>100</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

* not applicable