Remembrance of Streams Past: Overload-Sensitive Management of Archived Streams

Authors: S. Chandrasekaran, M. Franklin
Published at: VLDB ‘04
(Traditional) View on DBMS

OLTP/OLAP

- Finite relations
- Data is stored
- Queries keep coming in

Q: Add an item to your amazon shopping cart

DSMS

- Finite + infinite relations
- Data keeps coming in
- Triggers(queries) are stored

Q: identify rush hour by counting # cars passing between A and B
System Under Study

Disk-based DSMS that combines real-time data streams with historical data simultaneously

e.g.: Since the last accident, report an hourly # cars pass through gate A
Where is the bottleneck?

• An analogy:
  – CPU clock: 1 second
    • Look down at your desk
  – L1 cache: 3 seconds
    • Grabbing a piece of paper from your desk
  – L2 cache: 14 seconds
    • Picking up a book for a nearby shelf
  – Main memory: 4 minutes
    • Walking down the hall to buy a Twix bar
  – Disk: ?
Disks

1 year and 3 months
(Traditional) Remedies for Overload

• Indexes and optimizations (OLTP)
  – Not quite as possible as in OLTP

• Postponing computation (Data warehousing)
  – Falls even further behind the live data

• Load shedding (DSMS)
  – Does not confront the root of the problem
  – Dropped data might be important in future
Proposed Idea

• Reduce I/O cost
  – Retrieve a reduced version of the historical data
  – In the form of:
    • Sampled (e.g. random sampling)
    • Summarized (e.g. window aggregation)

• Trade-off I/O for data quality

• Focuses on the architectural issues, rather than DR techniques
Sampling/Summarization

**Disk**

<table>
<thead>
<tr>
<th>ID</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

**Sample**

<table>
<thead>
<tr>
<th>ID</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

**Disk**

<table>
<thead>
<tr>
<th>ID</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>29</td>
</tr>
<tr>
<td>3-4</td>
<td>34</td>
</tr>
<tr>
<td>5-6</td>
<td>43</td>
</tr>
</tbody>
</table>

**Fetch**

**Application**
A Famous Windowed Aggregation: Candlestick Chart

Pressure:
Problem’s Settings

• No a-priori knowledge of queries
  – Multiple resolutions of reduced data

• Varying load/resources in the system
  – The reduction level must be decided based on the system load

• No fixed sampling/summarization techniques
  – The DSMS must have flexibility to choose their desired technique

Add an Overload-sensitive Stream Capture and Archive Reduction component
Proposed Architecture

- Sits between disk and executor
  - The disk’s content is invisible to executor
  - Only change to executor: r-UDF

- Sits between NI and disk
  - No change to NI is needed

- Maintains multiple resolutions of data
  - Finds the best matching version

- Each physical scan is associated with an R-UDF
  - Notifies OSCAR as to the reduction degree
  - Depends on the system load

- Desired reduction = 60%
Different Designs: Different Points of (uniform) Sampling

Data-arrival driven

Eager Approach

OnWrite Replicate

Randomize Then Sort

OnRead Modify

Query-arrival driven
OSCAR v1: OnWriteReplicate

At data arrival

At query arrival

0110010101
1101010001

live data

historical data

r-UDF
Uniform sampling
Desired reduction = 60%

ID | cnt
--|---
3  | 17
6  |  2

ID | cnt
--|---
2  | 17
4  | 26
6  |  2
8  | 32

Primary copy

ID | cnt
--|---
1  | 12
2  | 17
3  |  8
4  | 26
5  | 41
6  |  2
7  | 24
8  | 32

75% reduction

50% reduction

Reduced resolutions
Different Designs: Different Points of (uniform) Sampling

Data-arrival driven: 
- OnWrite
- Replicate

Randomize Then Sort

Query-arrival driven: 
- OnRead
- Modify

Lazy(on-demand) Approach
OSCAR v2: OnReadModify

At data arrival

At query arrival

x2

0110010101
1101010001

live data

OSCAR

historical
data

0110010101
1101010001

OSCAR

historical
data

Primary copy

<table>
<thead>
<tr>
<th>ID</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
</tr>
</tbody>
</table>

reduced

<table>
<thead>
<tr>
<th>ID</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
</tr>
</tbody>
</table>

Q1 Q2 Qu
r-UDF
Uniform sampling
Desired reduction
= 75%
Different Designs: Different Points of (uniform) Sampling

Data-arrival driven

- OnWrite
- Replicate

Randomize

Then

Sort

OnRead

Modify

Query-arrival driven

The Hybrid Approach
OSCAR v3: RandomizeThenSort

At data arrival

At query arrival

Run length = 2 Blks
#Tuples/Blk = 2

<table>
<thead>
<tr>
<th>ID</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
</tr>
</tbody>
</table>

Original Data (not materialized)

<table>
<thead>
<tr>
<th>ID</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
</tr>
</tbody>
</table>

Uniform sampling
Desired reduction = 50%
Experiments: Inserting Without Querying

- 4 reduced resolutions: 20%, 40%, 60%, 80%
- Is sublinear in the size of the original stream and copies
- Due to the cost amortization in the NI
- Slower than strawman due to the extra samplerate attribute
- Due to additional copy overhead for both tables
- This is the baseline
- One copy at the arrival
- Read the original copy at query time
- The slight difference is because of different wasted space in runs
Experiments: Querying While Inserting

- All methods stabilize eventually simultaneous queries
- Eager approach pays little cost at query time
- Post-vac Lazy approach has the best performance
- Hybrid solutions with larger runs have better performance
  - For RTSs with smaller runs
    - The blocks are close on disk
    - Due to pre-fetching and buffering, disk ends up doing a sequential scan
Summary

• OSCAR is a framework disk-based DSMS

• The goal is to alleviate the bottleneck (disk) by reading only a reduced historical data

• OSCAR organizes data on disk into multiple resolutions of reduced summaries
  – such as samples of different sizes.

• Depending on stream speed, the system picks the right resolution level for QP