Optimization of Nonsequenced Queries using Log-Segmented Timestamps
Curtis Dyreson
Utah State University, Logan, Utah, USA

Outline
- Timestamp representation
- Application to nonsequenced queries
- Evaluation
- Conclusion

Timestamp Representation
- An interval/period timestamp

- Intervals are
  - [0, 3]
  - [5, 8]
- Features
  - Just start and stop times
  - Minimal information, smallest in terms of storage
  - What is an alternative and why would anyone do it differently?

Temporal Grouping and Aggregation
- Temporal data about football players

<table>
<thead>
<tr>
<th>players</th>
<th>team</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wilson</td>
<td>ManU</td>
<td>[0-5]</td>
</tr>
<tr>
<td>Arnhelm</td>
<td>ManU</td>
<td>[3-8]</td>
</tr>
</tbody>
</table>

- How many players on each team at the same time?
  - Groups vary over time
  - Player belongs to potentially n^2 groups
  - Special aggregation techniques for temporal aggregation

Sequenced Aggregation in Map/Reduce
- Interval representation is bad in Map/Reduce

- Need a new kind of timestamp
  - Curtis E. Dyreson: Using CouchDB to Compute Temporal Aggregates. HPCC/SmartCity/DSS 2016: 1131-1138

Log-segmented Timestamp
- Problem: can’t shard intervals
- Introduce log-segmented timestamp
- Partition timeline into pre-defined segments
Log-segmented Example

Log-segmented label for period \([8, 9]\) is 1100

Log Segment Example

Log-segmented label for period \([5, 5]\) is 10101

Convert Periods to Segments

- Period \([2-8]\) is \{1001, 101, 11000\}

- Compact – \(2^{\log_2(n)}\) segments can represent any period

Log-segmented Sequenced Semantics

- Sequenced semantics for relational DBs

Outline

- Timestamp representation
- Application to nonsequenced queries
- Evaluation
- Conclusion

Fabio Grandi E-mail

- Fabio: Interesting idea, but what about log-segmented for nonsequenced queries, after all, nonsequenced is more important than sequenced
- Me: Yes, nonsequenced is important, but log-segmented timestamps don’t improve nonsequenced
Nonsequenced Semantics

- Most common temporal extension of a query language
- Nonsequenced: query has explicit temporal predicates and constructors
- Benefit
  - Temporal can be added to any DBMS
  - Layer, no DBMS modification

Example Nonsequenced Join

```
SELECT s.dept, OVERLAPS(r, s)
FROM tesco s, walmart r
WHERE r OVERLAPS s
```

System Architecture

TEMPORAL SQL Query

\[\text{Temporal to Nontemporal Translation Layer}\]

SQL Query

\[\text{Relational Database Management System}\]

Example Nonsequenced Join

```
SELECT s.dept, GREATEST(r.time.start, s.time.start) AS start,
       LEAST(r.time.stop, s.time.stop) as stop
FROM tesco s, walmart r
WHERE ((r.start <= s.start AND s.start <= r.stop)
       OR (s.start <= r.start AND r.start <= s.stop))
```

(c) Result of the nonsequenced evaluation of the query in Figure 1.

System Architecture

```
RDBMS
SQL Compiler/Optimizer
Indexes
Runtime Engine
```

Query Execution Plan

- Cost of query highlighted in red
- Note use of indexes highlighted in yellow
- We can lower cost of query from 30,587,076 to 1,376,011 using techniques in the paper
Segment Columns

- Try to avoid range query on index
- Keep columns for normal timestamp
- Add columns for segments
  - Note at most two segments of any given length
  - Column s2 – first segment of length 2
  - Column s2x – other segment of length 2
  - Nulls are common in segment columns

<table>
<thead>
<tr>
<th>Data</th>
<th>Time Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dept</td>
<td>Start</td>
</tr>
<tr>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>2</td>
</tr>
</tbody>
</table>

Segment Endpoint Containment

- Precompute and store segments that contain a start or stop time
- Consider the interval [1, 11]

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Segment Endpoint Containment

- Precompute and store segments that contain a start or stop time
- Consider the interval [1, 11]

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Segment Endpoint Containment

- Precompute and store segments that contain a start or stop time
- Consider the interval [1, 11]

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Help Determine Endpoint Containment

Add columns for segments that could contain the start and stop points

- Prefix column p2 – What segment of length 2 contains start?
- Prefix column p2e – What segment of length 2 contains stop?

<table>
<thead>
<tr>
<th>Data</th>
<th>Time Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dept</td>
<td>Start</td>
</tr>
<tr>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>5</td>
</tr>
</tbody>
</table>
Using

• Is 2 contained in [1,11]?

<table>
<thead>
<tr>
<th>Dept</th>
<th>Start</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>2</td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1001</td>
</tr>
</tbody>
</table>

• Yes, p2 == s2

• Is 2 contained in [5,6]?

<table>
<thead>
<tr>
<th>Dept</th>
<th>Start</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>2</td>
<td>1010</td>
</tr>
<tr>
<td>...</td>
<td>6</td>
<td>1010</td>
</tr>
</tbody>
</table>

• No, no N such that sN == pN or sNx == pN

Outline

• Timestamp representation
• Application to nonsequenced queries
• Evaluation
• Conclusion

Using

• Is 2 contained in [1,11]?

<table>
<thead>
<tr>
<th>Dept</th>
<th>Start</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>2</td>
<td>1010</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1001</td>
</tr>
</tbody>
</table>

• Yes, p2 == s2

• Is 2 contained in [5,6]?

<table>
<thead>
<tr>
<th>Dept</th>
<th>Start</th>
<th>Stop</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>2</td>
<td>1010</td>
</tr>
<tr>
<td>...</td>
<td>6</td>
<td>1010</td>
</tr>
</tbody>
</table>

• No, no N such that sN == pN or sNx == pN

Experiment Setup

• Test machine
  • Oracle Cloud Instance
  • 4 CPUs – 2.4 GHZ
  • 32GB RAM
  • 1 TB SSD drive
  • Linux

• Test DBMS
  • Postgres, version 14
  • Made no adjustments to out-of-the-box settings
  • EXPLAIN – optimizes and generates query execution plan
  • EXPLAIN ANALYZE – runs query as well
Evaluation

- Compare timestamped vs. log-segmented
- One relation
  - Timestamped
    - Employees(id, name, department, start, stop)
  - Log-segmented
    - Employees(id, name, department, start, stop, s1, s2, ..., s19, s1x, s2x, ..., s19x, p1, p2, ..., p19, p1e, p2e, ..., p19e)

Experiment Data and Queries

- Test data
  - Synthetically generated
  - 100 departments, 90% different names
  - 10K to 50K tuples
  - Timeline of $2^{19}$
  - Timestamps $2^8$, randomly generated
- Create indexes for everything!
- Test query
  - Join of employee with itself, only on the temporal attributes
    - Focus on timestamps, not non-temporal columns
  - Three predicates for join
    - Overlaps
    - Contains
    - Starts

Evaluation - Overlaps

- SQL query WHERE clause is ugly

<table>
<thead>
<tr>
<th>WHERE ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x.s1 = y.p1 OR r.s2 = y.p2 OR ... OR x.s19 = y.p19)</td>
</tr>
<tr>
<td>OR (x.s1 = y.p1e OR r.s2 = y.p2e OR ... OR x.s19 = y.p19e)</td>
</tr>
<tr>
<td>OR (x.s1 = y.p2e OR r.s2 = y.p2e OR ... OR x.s19 = y.p19e)</td>
</tr>
</tbody>
</table>

- Query execution plan

Evaluation - Contains

- Starts performs worse with log-segmented

Disadvantages

- Space cost increases (next slide)
Results – DB Size

Outline

• Timestamp representation
• Application to nonsequenced queries
• Evaluation
• Conclusion

Conclusion

• Query optimization technique
  • Log segmented stores both normal and log-segmented timestamps
  • Run optimizer on both, choose best plan
  • Downside is extra space
  • Additional benefit – sequenced semantics!
• Tested only a small part of query optimization space

Future Work

• Log-segmented Cypher
• New temporal hash-join technique

Size of result, time-line size, value conditions, etc.