Column-Stores vs. Row-Stores: How Different Are They Really?

Daniel Abadi (Yale),
Samuel Madden (MIT),
Nabil Hachem (AvantGarde Consulting)
June 12th, 2008
Row vs. Column-Store

Row-Store

- Easy to add a new record
- Might read in unnecessary data

Column-Store

- Only need to read in relevant data
- Tuple writes might require multiple seeks

Daniel Abadi -- Yale University
Column-Stores

- Really good for read-mostly data warehouses
  - Lot’s of column scans and aggregations
  - Writes tend to be in batch
  - [CK85], [SAB+05], [ZBN+05], [HLA+06], [SBC+07] all verify this
  - Top 3 in TPC-H rankings (Exasol, ParAccel, and Kickfire) are column-stores
    - Factor of 5 faster on performance
    - Factor of 2 superior on price/performance
Data Warehouse DBMS Software

- $4.5 billion industry (out of total $16 billion DBMS software industry)
- Growing 10% annually
Momentum

- Right solution for growing market → $$$$$
- Vertica, ParAccel, Kickfire, Calpont, Infobright, and Exasol new entrants
- Sybase IQ’s profits rapidly increasing
- Yahoo’s world largest (multi-petabyte) data warehouse is a column-store (from Mahat Technologies acquisition)
Paper Looks At Key Question

- How much of the buzz around column-stores just marketing hype?
  - Do you really need to buy Sybase IQ or Vertica?
  - How far will your current row-store take you?
    - Can you get column-store performance from a row-store?
    - Can you simulate a column-store in a row-store?
Comparing row-store vs. column-store is dangerous/borderline meaningless
Instead, compare row-store vs. row-store and column-store vs. column-store
- Simulate a column-store inside of a row-store
- Remove column-oriented features from column-store until it behaves like a row-store
Simulate Column-Store
Inside Row-Store

Option A: Vertical Partitioning

Option B: Index Every Column

Daniel Abadi -- Yale University
Experiments

• Star Schema Benchmark (SSBM)
  - Fact table contains 17 columns and 60,000,000 rows
  - 4 dimension tables, biggest one has 80,000 rows
  - Queries perform 2-4 joins between fact table and dimension tables, aggregate 1-2 columns from fact table
  - [OOC06]

• Implemented by professional DBA
  - Original row-store plus 2 column-store simulations on same row-store product
SSBM Averages

<table>
<thead>
<tr>
<th></th>
<th>Normal Row-Store</th>
<th>Vertically Partitioned Row-Store</th>
<th>Row-Store With All Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average</strong></td>
<td>25.7</td>
<td>79.9</td>
<td>221.2</td>
</tr>
</tbody>
</table>
What’s Going On?

- **Vertically Partitioned Case**
  - Tuple Sizes
  - Horizontal Partitioning

- **All Indexes Case**
  - Tuple Reconstruction
**Tuple Size**

- Queries touch 3-4 foreign keys in fact table, 1-2 numeric columns
- Complete fact table takes up ~4 GB (compressed)
- Vertically partitioned tables take up 0.7-1.1 GB (compressed)
Horizontal Partitioning

- Fact table horizontally partitioned on year
  - Year is an element of the ‘Date’ dimension table
  - Most queries in SSBM have a predicate on year
  - Since vertically partitioned tables do not contain the ‘Date’ foreign key, row-store could not similarly partition them
What’s Going On?

- Vertically Partitioned Case
  - Tuple Sizes
  - Horizontal Partitioning
- All Indexes Case
  - Tuple Construction
Tuple Construction

- Common type of query:
  - SELECT store_name, SUM(revenue) 
    FROM Facts, Stores 
    WHERE fact.store_id = stores.store_id 
    AND stores.country = "Canada" 
    GROUP BY store_name
Tuple Construction

- Result of lower part of query plan is a set of TIDs that passed all predicates
- Need to extract SELECT attributes at these TIDs
  - BUT: index maps value to TID
  - You really want to map TID to value (i.e., a vertical partition)
  - Tuple construction is SLOW
So....

- All indexes approach is a poor way to simulate a column-store
- Problems with vertical partitioning are NOT fundamental
  - Store tuple header in a separate partition
  - Allow virtual TIDs
  - Allow HP using a foreign key on a different VP
- So can row-stores simulate column-stores?
Row-Store vs. Column-Store

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>Row-Store</th>
<th>Row-Store (M V)</th>
<th>C-Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average: 25.7  1.7  4.4

Daniel Abadi -- Yale University
Row-Store vs. Column-Store

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>Row-Store</th>
<th>Row-Store (MV)</th>
<th>C-Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>25.7</td>
<td>11.7</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Daniel Abadi -- Yale University
Column-Store Experiments

- Start with column-store (C-Store)
- Remove column-store-specific performance optimizations
- End with column-store with a row-oriented query executer
Compression

- Higher data value locality in column-stores
  - Better ratio → reduced I/O
- Can use schemes like run-length encoding
  - Easy to operate on directly for improved performance ([AMF06])
Early vs. Late Materialization

- Early Materialization: create rows first. But:
  - Poor memory bandwidth utilization
  - Lose opportunity for vectorized operation

QUERY:
SELECT custID, SUM(price) FROM table
WHERE (prodID = 4) AND (storeId = 1) AND
GROUP BY custID

Daniel Abadi -- Yale University
Other Column-Store Optimizations

- Invisible join
  - Column-store specific join
  - Optimizations for star schemas
  - Similar to a semi-join
- Block Processing
Simplified Version of Results

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>Original C-Store</th>
<th>C-Store, No Compression</th>
<th>C-Store, Early Materialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>4.4</td>
<td>14.9</td>
<td>40.7</td>
</tr>
</tbody>
</table>
Conclusion

• Might be possible to simulate a row-store in a column-store, BUT:
  ❖ Need better support for vertical partitioning at the storage layer
  ❖ Need support for column-specific optimizations at the executer level
• Working with HP Labs to find out
Come Join the Yale DB Group!

Daniel Abadi -- Yale University