C-Store: Looking Back and Looking Forward

Daniel Abadi
Michael Stonebraker

(On behalf of all C-Store authors: Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth O’Neil, Pat O’Neil, Alex Rasin, Nga Tran, Stan Zdonik)
Thank You

» 100s of Vertica employees (and consultants) who realized the C-Store vision
» Reviewers who accepted the paper in 2005
» Everyone who presented the C-Store paper in a seminar or database course
» Everyone who used ideas from C-Store paper and/or cited it
Column-stores have been around since the 70s

Pre-1985 Seminal Papers


Column-Stores that pre-date VLDB 2005

SybaseIQ

CWI: MonetDB, MonetDB/X100

Wisconsin: PAX, Fractured Mirrors

Michigan: Data Morphing

CMU: Clotho
Column-stores have been around since the 70s

C-Store’s Practical System Design
Expanded Applicability of Column-stores
Row vs. Column-Stores

Row Store

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

<table>
<thead>
<tr>
<th>Last Name</th>
<th>First Name</th>
<th>E-mail</th>
<th>Phone #</th>
<th>Street Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Column Store

- Only need to read in relevant columns
- SELECT X,Y,Z requires partial row reconstruction
- Tuple writes might require multiple seeks

<table>
<thead>
<tr>
<th>Last Name</th>
<th>First Name</th>
<th>E-mail</th>
<th>Phone #</th>
<th>Street Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C-Store’s Key Architectural Features

Hybrid Storage Architecture

> Write Optimized Store (WOS)
- Memory based
- Unsorted / Uncompressed
- Low latency / Small quick inserts

> Read Optimized Store (ROS)
- On disk
- Dense-packed
- Sorted / Compressed
- Large data loaded direct

TUPLE MOVER
Asynchronous Data Transfer

New Data
### Key Features: Compression

#### Quarter
- **Q1**: (1, 1, 300)
- **Q2**: (301, 350)
- **Q3**: (651, 500)
- **Q4**: (1151, 600)

#### Category
- **Electronics**
- **Clothing**
- **Food**

#### Price
- **Q1**: 5
- **Q2**: 7
- **Q3**: 9
- **Q4**: 8

#### Run-Length Encoding
- **Day**: (1, 1, 5) (2, 6, 2) ...
- **Quarter**: (Q1, 1, 300) (Q2, 301, 350) (Q3, 651, 500) (Q4, 1151, 600)

#### Dictionary Encoding
- **Category**
  - **Electronics**: 1 2 1
  - **Clothing**: 3 3 3
  - **Food**: 2 ...

#### Bit-Vector Encoding
- **Price**
  - **Q1**: 5
  - **Q2**: 7
  - **Q3**: 9
  - **Q4**: 8

---

**DBg Database Group**
MIT Computer Science and Artificial Intelligence Lab
Key Features: Operating on Compressed Data

```
<table>
<thead>
<tr>
<th>Quarter</th>
<th>Product ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Q1, 1, 300)</td>
<td>1 0 0</td>
</tr>
<tr>
<td>(Q2, 301, 6)</td>
<td>0 0 1</td>
</tr>
<tr>
<td>(Q3, 307, 500)</td>
<td>0 1 0</td>
</tr>
<tr>
<td>(Q4, 807, 600)</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>
```

Index Lookup + Offset jump

```
SELECT ProductID, COUNT(*)
FROM table
WHERE (Quarter = Q2)
GROUP BY ProductID
```
Key Features: Late Materialization

Advantages of Late Materialization:
- Construct fewer tuples
- Limits data decompression
- Better memory bandwidth utilization

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

Early Materialization

Late Materialization

AGGREGATE

SELECT prodID, storeID, custID, price
AGGREGATE

Construct

Select + Aggregate
Effect on C-Store performance

Average for SSBM queries on C-Store

- Simple column-store
- Column-oriented join algorithm
- Enable compression & operate on compressed
- Enable late materialization
Summary of key C-Store features

» Storage layout
- columnar storage
- header/ID elimination
- compression
- dense-packed values

» Execution engine
- column operators
- avoid decompression
- late materialization
The Impact of Vertica

<table>
<thead>
<tr>
<th>Warehouse vendor</th>
<th>2006</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>Row store</td>
<td>Column store</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Row store</td>
<td>Column store</td>
</tr>
<tr>
<td>Oracle</td>
<td>Row store</td>
<td>Rumors abound</td>
</tr>
<tr>
<td>Teradata</td>
<td>Row store</td>
<td>Row or column store</td>
</tr>
<tr>
<td>Greenplum</td>
<td>Row store</td>
<td>Row or column store</td>
</tr>
<tr>
<td>Cloudera</td>
<td></td>
<td>Column store</td>
</tr>
<tr>
<td>Amazon</td>
<td></td>
<td>Column store</td>
</tr>
</tbody>
</table>
Business Intelligence (Stupid Analytics)

» Is yesterday’s news

» The coming world is “data science” (smart analytics)
Smart Analytics

» Complex math operations (machine learning, clustering, trend detection, ....)
  - The world of the “quants” and the “rocket scientists”
  - Mostly specified as linear algebra on array data

» A dozen or so common ‘inner loops’
  - Matrix multiply
  - QR decomposition
  - SVD decomposition
  - Linear regression
Smart Analytics
An Example

» Consider closing price on all trading days for the last 5 years for two stocks A and B

» What is the covariance between the two time-series?

\[
\frac{1}{N} \times \text{sum} ((A_i - \text{mean}(A)) \times (B_i - \text{mean}(B)))
\]
Now Make It Interesting ...

- Do this for all pairs of 4000 stocks
  - The data is the following 4000 x 1000 matrix

<table>
<thead>
<tr>
<th>Stock</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>$t_6$</th>
<th>$t_7$</th>
<th>...</th>
<th>$t_{1000}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_{4000}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Array Answer

- Ignoring the \((1/N)\) and subtracting off the means ....

\[ \text{Stock} \times \text{Stock}^T \]
What Does a Data Scientist Do? (after he curates his data)

» Until (tired) {
  Data management
  Analytics
}

MIT Computer Science and Artificial Intelligence Lab
How to Support This?

» Array DBMS (e.g. SciDB)
» Column store plus embedded stat (e.g. R, MatLab ..)
  - From Vertica, Cloudera, ...

"DBg Database Group
MIT Computer Science and Artificial Intelligence Lab"
And....

- Array stores do not need to cast tables to arrays (expensive $N^{**}2$ operation)
  - So they have an unfair advantage

- But careful integration required in either case!!!
  - Block cyclic allocation
  - Same compression, same block layout, ...
Linear Algebra

» Sparse and dense are both needed
  – And they are different
» On dense, there are 4 orders of magnitude difference between Python and C
» On dense, another order of magnitude from hardware-specific tweaking
» Focus on a few “inner loop” operations
Key Features: Late Materialization

QUERY:
SELECT custID, SUM(price)
FROM table
WHERE (prodID = 4) AND
    (storeID = 1) AND
GROUP BY custID
Key Features: Late Materialization

**QUERY:**

```
SELECT custID, SUM(price)
FROM table
WHERE (prodID = 4) AND (storeID = 1) AND GROUP BY custID
```
Key Features: Late Materialization

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

AGG

AND

Data Source custID
Data Source price

Data Source prodID
Data Source storeID

custID
price
Key Features: Late Materialization

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

Data Source

AGG

AND

AGG

prodID

storeID
custID

price

3

93

3

13

80

4

2

2

7

4

1

3

13

42

80