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

Daniel Abadi, Samuel Madden, Nabil Hachem

Presented by Guozhang Wang
November 18th, 2008

Several slides are from Daniel Abadi and Michael Stonebraker

Column Stores for Read-Mostly Data Warehouses

- Storage-Level
  - Vertical Partitioning (VLDB 05)
  - Column-Specific Compression (SIGMOD 06)
- Executer-Level
  - Fast Join using Positions (VLDB 05)
  - Compressed Query Execution (SIGMOD 06)
  - Late Materialization (ICDE 07)

One Question:

- Do you really need to buy a Vertica or Sybase IQ?
  - Can we adapt our row-store to get column-store performance? Currently No
  - If not, what makes column-store not simulatable?

  Optimizations at query execution level

On the Other Hand…

- Directly comparing row-store with column-store is difficult
  - Some performance differences are fundamental differences between column stores and row stores
  - While some others are implementation artifacts.

Comparison Methodology

- Compare row-stores with row-stores and column-stores with column-stores.
  - Compare row-stores with "column like" row-stores.
  - Compare column-stores with "row like" column-stores

The Benchmark – SSBM

- Most (if not all) warehouses use star or snowflake schema
- Star Schema Benchmark (SSBM) is a simplified derivation from TPC-H
- One fact table (17 columns, 60,000,000 rows), and four dimension table (6 – 15 columns, at most 80,000 rows)
- Four types of queries, joining at most 3 dimensional tables

Row-Store Execution

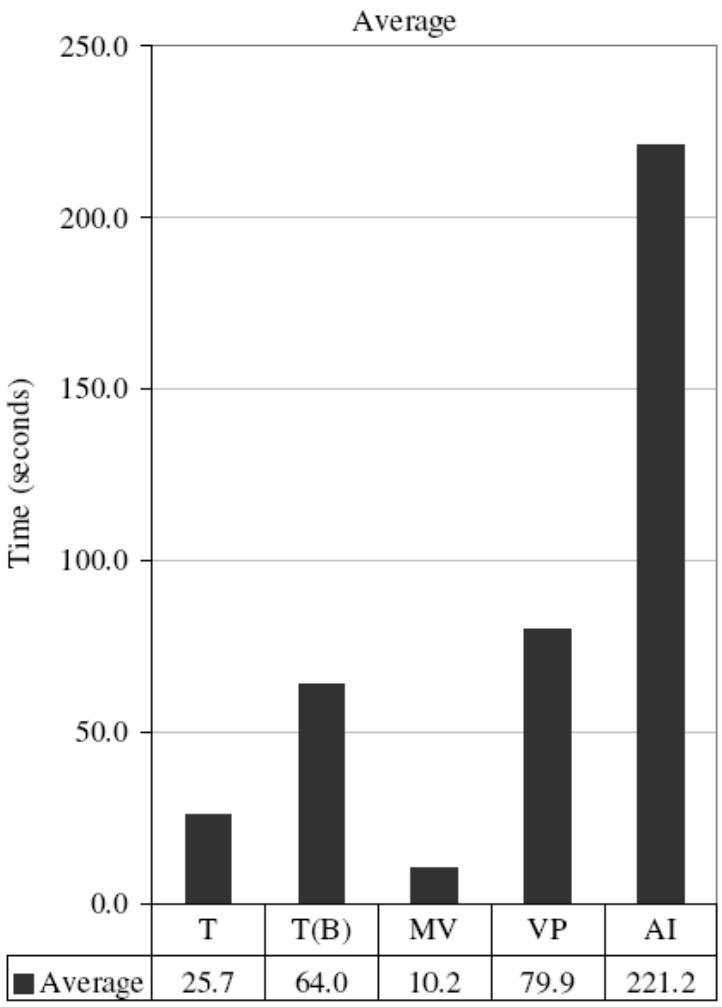
- Vertical Partitioning
  - each attribute is a two-column table: (values, position)
- Index-All
  - unclustered B+Tree index for every column of every table
- Materialized View
  - ***optimal*** set of materialized views for every query

Experiments: Row vs. Row

MV
Materialized View

VP
Vertcal Partitioning

AI
Index-All

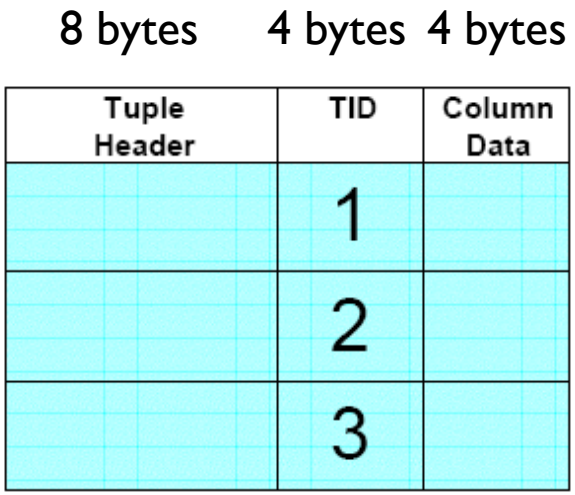


Reasons

- Tuple header overhead for VP
  - Complete f_table takes up

    ~4 GB (compressed)
  - VP tables take up

    0.7-1.1 GB each (compressed)



- Hash join is slow
  - But is probably the best option for Index-all

Part of this slide comes from Daniel Abadi

Conclusion 1

- Index-all approach is a poor way to simulate a column-store
  - it forces system to join columns at the beginning, but cannot defer them
- Problems with vertical partitioning are NOT fundamental
  - its disadvantages can be alleviated

Column-Store Execution

- Compression
- Late Materialization
- Block Iteration
- Invisible Join
  - move predicates on d_table to f_table to minimize out-of-order value extractions.

Removing these optimizations gives a "row-store like" column-store

Experiments: Col vs. Col

T vs. t

Tuple vs. Block

I vs. i

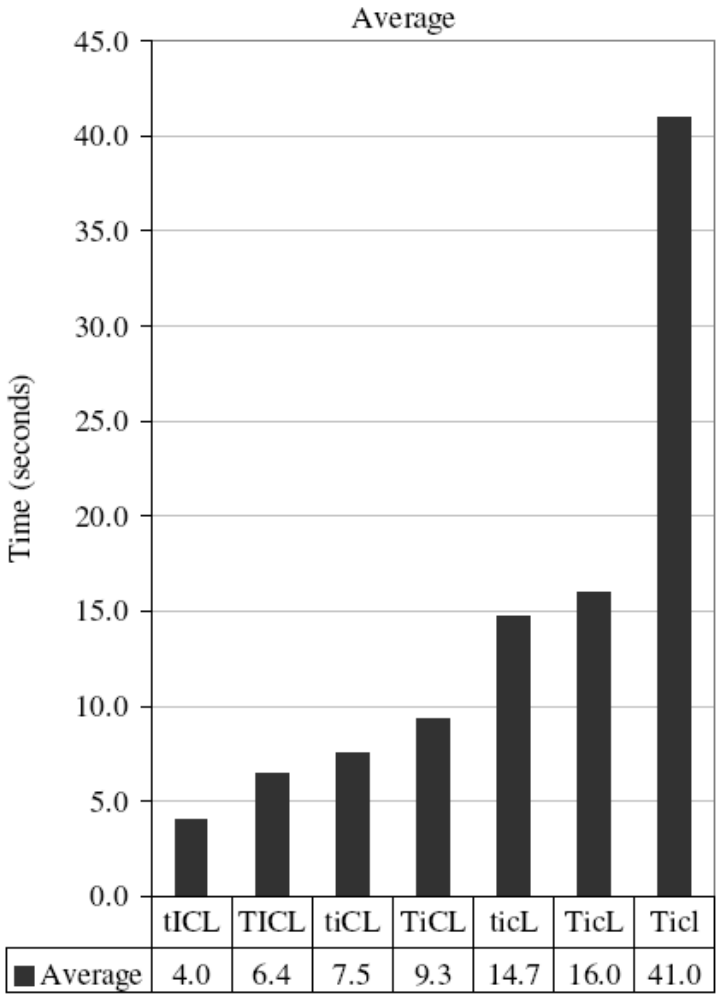
Invis. Join vs. Disabled

C vs. c

Comp. vs. Disabled

L vs. l

Late Mat. vs. Disabled



Performance Analysis

- Block: 5% - 50% depending on compression
- Invisible Join: 50% - 75%, but it is special optimization for star schemas
- Compression: almost x2 averagely, while x10 on sorted data
- Late materialization: x3 because of selective predicates

Conclusion II

- The most significant optimizations are compression and late materialization
- After all the optimizations are removed, the column store acts just like a row store
- Invisible join works so well that denormalization is not very useful for column store

Answer to the Question:

Can we adapt a row-store to get column-store performance?

- It might be possible, BUT:
  - need better support for vertical partitioning at the Storage Level
    - store tuple header separately
    - virtual record-id
  - need support for column specific optimizations at the Executer Level
    - late materialization
    - direct operator on compressed data

Questions?

One Size Fits All? – Part 2: Benchmarking Results

Michael Stonebraker, et al

One Size for All DBMS Needs?

- In the 1970s
  - Killer application: transaction processing
  - Relational gold standard
    - Record stored contiguously on disk
    - B-Tree indexing
    - Row-oriented query optimizer and executor
    - More...
- Over the years
  - New needs appear: XML, Data Warehouses…
  - New features are added in order to continue selling the original structure for these needs.

However…

- OSFA RDBMS is losing
  - To proprietary file systems in text search engines (GFS, Bigtable)
  - To column store systems in data warehouses (Vertica)
  - To specialized designed engine in stream processing (StreamBase)
  - To customized tools in scientific and intelligence data bases (Matlab)

Benchmark Results

- Telco Call Benchmark
  - Vertica 47X on 1/100 the hardware cost
- SSBM
  - Vertica 8X in ½ the space
- Split Adjusted Price & Forward First Arrival
  - StreamBase 25X if required state implemented as an RDBMS table
- Dot Product & Matrix Multiplication
  - ASAP 100X against RDBMS, and 10X against Matlab

ASAP Design

- ChunkyStore: like vertical partition, linear algorithm to read each chunk just once
- Compression: like column-specific compression, delta encode arrays
- Integration of "Cooking" and Storage: like WS and RS, same data model
- Data Uncertainty: convert between $R_{1,2,3}$
  - Value-probability pair: accurate
  - Expectation-variance pair: performance
  - Upper-lower bound pair

Reason?

- Different applications have different characteristics and requirements
  - Text search: semi/no structure, relaxed answers, no transaction…
  - Data warehouse: few uploads, ad hoc reads, star schema tables…
  - Stream processing: main memory storage, single tuple processing…
  - Scientific computation: Multi-D array storage, uncertainty management…

Conclusion

- Conflicting application requirements need custom architectures: OSFA is no longer true.
- What is next to OSFA DBMS?
  - No change: one RDBMS with high end specialization
  - K systems united by common parser
  - Data federations of incompatible systems
  - A scratch rewrite? (much more general engine which encompass all the requirements)

Obvious Research Agenda

- Find a market where OSFA doesn't work and customers are in pain
- Figure out what does

This slide comes from Michael Stonebraker

Thanks