Big Data Storage in Modern Databases

Jay Urbain, PhD

Credits:
Michael Stonebraker, Ugur Çetintemel
"One Size Fits All": An Idea Whose Time Has Come and Gone. ICDE 2005
Michael Stonebraker, Matei Zaharia , Samuel Madden

DB History

- **1970's: relational model invented**
- **1984: DB2 released, RDBMS declared mainstream**
- **Circa 1990: RDBMS takes over**
  - "One-size fits all" solution
  - I'm the guy with the hammer; everything is a nail
- **2006: ICDE paper**
  - "One-size does not fit all"
  - Co-existence of several solutions
- **2013: One size fits none**

Traditional RDBMS Wisdom

- **Dynamic row-level locking**
- **Aries-style write-ahead log**
- **Replication (asynchronous or synchronous)**
  - Update the primary first
  - Then move the log to other sites
  - And roll forward at the secondary(s)

Traditional RDBMS Wisdom

- **Data is in disk block formatting, heavily encoded, either 512 or 4K bytes.**
- **With a main memory buffer pool of blocks (pages).**
- **Query plans**
  - Optimize CPU, I/O
  - Fundamental operation is read a row
- **Indexing via B-trees**
  - Clustered or unclustered

One size DB solution does not fit all

- RDBMS's date from the 1980's
- Basically legacy systems
- Try to provide general solution to most data management problems, end up as master of none
- Suffer from "The Innovators Dilemma"
- Need to reconsider for big data, analytics, different apps

Three main DBMS markets

- One-third data warehouses OLAP
- One-third OLTP
- One-third everything else

Data warehousing

- Column stores are well along at replacing row stores for OLAP
- Why?
  - **Because they are a factor of 50 – 100 faster**

Dimensional Data Model

- **Most warehouses have a central fact table**
  - Who bought what item in what store at what time.
  - What patient was give what drug at what time by who.
- **Surrounded by “dimension” tables**
  - Store, time, product, customer, ...
- **So-called “star/snowflake schema”**
  - See anything written by Ralph Kimball

Warehouses/OLAP

- **Typical warehouse query reads 4-5 attributes from a 100 column fact table**
  - Row store - *reads all 100*
  - Column store - *reads just the ones you need*
- **Compression is way easier and more productive in a column store**
  - Each block has only one type of attribute

Column Store

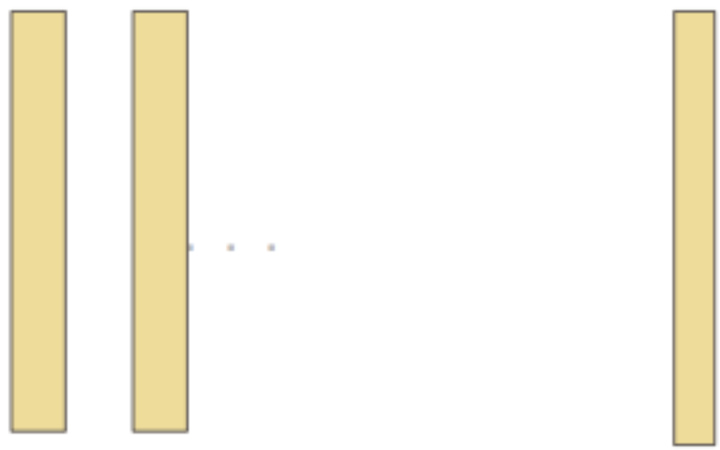
- **No big record headers in a column store**
  - They don't compress well
- **A column executor is wildly faster than a row executor**
  - Because of "vector processing"
  - See pioneering paper by Martin Kersten on this topic
  - See Vectorwise DB spun off from MonetDB

Column Store Vendors

- **Native column store vendors**
  - HP/Vertica, SAP/Hana, Paraccel (Amazon), SAP/Sbase/IQ
- **Native column store vendors open source**
  - MonetDB, LucidDB, Lucid Impala, MySQL InfiDB, MySQL ICE
- **Native row store vendors**
  - Microsoft, Oracle, DB2, Netezza, PostgreSQL, MySQL
- **In transition**
  – Teradata, Asterdata, Greenplum (build on PostgreSQL)

Vertica

- Table is decomposed into a collection of materialized views, stored by column and sorted on all attributes left-to-right



Vertica

- A column is stored in 64K "chunklets".
- 1st attribute may be uncompressed (key) or delta encoded; remainder are compressed (delta compression, lempel-zipf, repeated values, huffman, ...)
- Chunklets are decompressed only when necessary
- Fundamental operation is "process a column"

Vertica

- **To load fast, there is a main memory row-store in front of this column store.**
  - Newly loaded tuples go there
  - In bulk, groups of rows are sorted, converted to column format and compressed
  - And written to new disk segments
  - Segment merge makes these segments bigger and bigger
  - Queries go to both places

OLTP Databases – 3 basic decisions

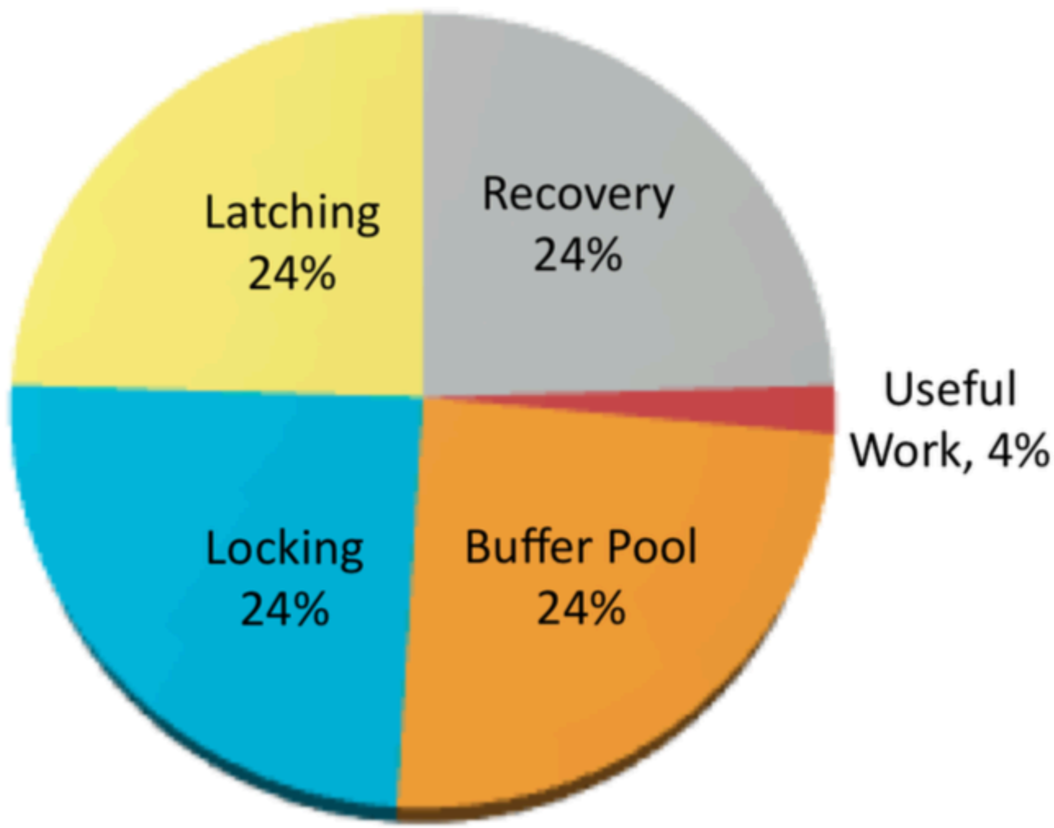
- Main memory *versus* disk orientation
- Replication strategy
- Concurrency control strategy

Reality Check on OLTP Data Bases

- TP data base size grows at the rate transactions increase
- 1 Tbyte is a really big TP data base
- 1 Tbyte of main memory < $30K (2015)
- ~ 64 GBytes per server in 16 servers
- If your data doesn't fit in main memory now, wait a couple of years and it will.....

Reality Check – Main Memory Performance

TPC-C CPU cycles



To Go Fast

- **Must focus on overhead**
  – Better B-trees to minimize path access length
- **Must get rid of the big four pie slices**
  – Anything less gives you a marginal win
  – x10 as an example improvement

Single Threading

- **Toast unless you do this**
  - Unless you get rid of queuing/contention
  - Or eliminate shared data structures
- **H-Store (and VoltDB) statically divide shared memory among the cores**
  - Would be interesting to look at more flexible schemes

Main Memory

- **Toast unless you do this**
- **What happens if my data doesn't fit?**
  - VLDB '14 paper by Debrabant et. al. – "Anti-caching"
  - Alternative distributed systems

To overcome the restriction that all data fit in main memory, a new technique, called anti-caching, where cold data is moved to disk in a transactionally-safe manner as the database grows in size. Because data initially resides in memory, an anti-caching architecture reverses the traditional storage hierarchy of disk-based systems. Main memory is now the primary storage device.

Concurrency Control

- MVCC (Multi-Version Concurrency Control) popular (Oracle, MySQL InnoDB Engine, NuoDB, Hekaton)
  - Each user sees a snapshot of the data at a particular point in time
- Time stamp order popular (H-Store/VoltDB)
  - Non-locking concurrency control (e.g., Lamport timestamp)
- Lightweight combinations of time stamp order and dynamic locking (Calvin, Dora)
  - Dynamically determine the most cost effective level of granularity for locking row, rows, table, etc., based on query.
  - Normal dynamic locking – is too slow

Logging

- **Command logging much faster than data logging**
  - ICDE '14 paper by Malvaiya
  - 1.5x higher throughput
- **HA (high availability) is now a requirement**
  - Failover to a replica; rarely recover from a log

The Old Way vs The New Big Data Way

- Main memory not disk
- Anti-caching not caching
- Command logging not data logging
- Failover not recovery from a log
- MVCC or timestamp order, not dynamic locking
- Single threaded not multi-threaded

Everything Else

- NoSQL
- Array stores
- GraphDBMSs
- Hadoop

NoSQL ~ 75 or so Vendors

- **Give up SQL**
  - Completely misguided notion
  - SQL mathematically precise relational language
  - SQL does not map well to OO NoSQL – i.e., object level
  - Nobody codes in assembler any more!!!
  - Never bet against the compiler!!

NoSQL ~ 75 or so Vendors

- **Give up ACID**
  - If you are guaranteed that you won't need it (now or in the future) then you are ok
  - Otherwise, you're in trouble

NoSQL ~ 75 or so Vendors

- **Schema later**
  - Most support semi-structured data - adding a new “column” is trivial
  - Don’t have to think about your data upfront
    - Good or bad depending on your point of view!

NoSQL – Summary

- **Moving quickly toward SQL**
  - Cassandra and MongoDB are moving towards SQL!
  - SparkSQL, Impalla, Hive
- **Moving toward ACID**
  - Even Jeff Dean (Google) now admits ACID is a good idea!
- **NoSQL**
  - Used to mean "No SQL"!
  - Then meant "Not only SQL"!
  - Moving toward "Not yet SQL" (i.e. convergence)!

NoSQL – Summary

- **Systems are fine for "low end" applications**
  - E.g webby things!
  - E.g. protection/ authentication data bases
  - Etc.

Array DBMSs and Complex Analytics

- Machine learning
- Data clustering
- Predictive models
- Recommendation engines
- Regressions
- Estimators

i.e. “Data Mining”

Complex Analytics

- By and large, they are defined on arrays
- As collections of linear algebra operations
- They are not in SQL!
- And often
  - Are defined on large amounts of data
  - And/or in high dimension

Complex Analytics on Array Data – Basic Example

- Consider the closing price on all trading days for the last 20 years for two stocks A and B
- What is the covariance between the two time-series?

$(1/N) * \text{sum}\ (A_i - \text{mean}(A)) * (B_i - \text{mean}\ (B))$

Complex Analytics on Array Data – Basic Example

- Do this for all pairs of 15000 stocks

Stock	t_1	t_2	t_3	t_4	t_5	t_6	t_7	….	t_{4000}
S_1									
S_2									
…									
S_{15000}									

- … or 100,000 * n 000's of recommendations

Array Answer

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

  Stock * Stock^T

System Requirements

- **Complex analytics**
  - Covariance is just the start – Defined on arrays
- **Data management**
  - Leave out outliers
  - E.g., Just on securities with market cap > $10B
- **Need scalability to many cores, many nodes and out-of-memory data (no additional memory available)**

Array DBMSs -- Summary

- **Array SQL**
  – For joins filters,...
- **Built in functions**
  – For SVD, Co-variance, ANOVA...
- **User-defined extensions**
  – If you don't see what you need
- **Likely to get tractions**
  – When the world moves to complex analytics
- **Does not look at all like the traditional wisdom**
- **Array DBMSs -- e.g. SciDB**

Graph DBMSs

- Focus on things like Facebook/twitter graphs
  - GraphLab
- **OLTP focus**
  - Neo4J
- **Analytics focus (shortest path, minimum cut set, ...)**
- **Can you beat** – RDBMS simulations?
  - Array simulations
  - **Jury is still out**

What is Hadoop?

- **File system HDFS + MapReduce + extensions**
- **Open source version of Google's GFS + Map-Reduce**
- **MapReduce**
  - Map (basically filter, transform) – Reduce (basically rollup)
- **Very good for "embarrassingly parallel" operations**
  - E.g. document indexing, document search, word count

The Hadoop Stack

- **Hive (or Pig) at the top**
  - Think SQL
- **Hadoop (Map-Reduce) in the middle**
- **HDFS (a file system) at the bottom**
- **Runs across any number of nodes**
  - Very scalable!

Possible Uses for Hadoop Stack

- **Embarrassingly parallel computations**
  - Good – indexing, document search
- **SQL aggregates (e.g. warehouse-style queries)**
  - Factor of 100 worse than a warehouse DBMS
- **Complex analytics**
  - Factor of 100 worse than an array DBMS
- **Scientific (e.g. computational fluid**
- **dynamics)**
  - Factor of 100 worse than MPI-based systems

Hadoop Usage at Facebook

- **95+% Hive or Cassandra**
  - Hadoop layer is a disaster

Most Likely Future

- **Cloudera, Hortonworks, and Facebook are ALL doing the same thing**
  - Defining and building an execution engine that processes Hive without using Hadoop layer
- **Effectively moving to compete in the warehouse market**
  - All warehouse vendors have Hive-like interfaces
- **There is a relatively small market for embarrassingly parallel Hadoop framework**
- **There is a much bigger market for a Hive-SQL framework**
  - Execution engines will look like data warehouse products
- **HDFS may or may not survive**
  - It is also horribly inefficient
- **Spark**
  - Distributed memory

Futures…

- **Warehouses will be a column store market**
- **OLTP will be a main memory market**
- **Array DBMSs and Graph DBMS may get traction**
  - Should understand what they are good for
- **NoSQL**
  - Popular for low-end applications
  - Especially document management, web stuff and places where you want schema-later
  - ACID-less