April 2024

C-Store: Column-oriented DBMS

Authors: Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik

Group 10: Isabela Fernandez, Dora Sasson, Edwin Shen, Shrey Shivaiah

BACKGROUND & MOTIVATION

BACKGROUND

- Address the challenge of designing a column-oriented DBMS
  - Warehouse-style queries
  - OLTP-style transactions
- Aim to create hybrid architecture
  - Read-optimized component
  - Write-optimized component
- C-Store seeks to provide high performance for
  - Query processing
  - Transactional updates

MOTIVATION

- Traditional relational DBMSs optimized for write operations
- Leads to inefficiencies in read-heavy environments
  - Data warehouses
- Balance crucial for organizations
  - Real-time data visibility
  - On-line updates
  - Improved query performance

RELATED WORK

RELATED WORK

- Previous work only focused on either
  - write-optimized systems for OLTP applications
  - read-optimized systems for data warehousing
- Traditional row-oriented DBMSs excel at write operations
  - suffer from inefficiencies when handling ad-hoc queries on large datasets
- Column-oriented DBMSs have shown advantages in query performance
  - lack support for efficient transactional updates
- Have not fully addressed problem
- Due to the inherent trade-offs between optimizing for reads vs writes

OVERVIEW

OVERVIEW

- C-Store introduces a novel column-oriented DBMS architecture
- Handles both read-heavy queries and transactional updates
- Hybrid design with:
  - Read-Optimized Store (RS) for query performance
  - Writeable Store (WS) for high-performance inserts and updates
  - Connected by tuple mover to facilitate data movement
- Focus on optimizing storage representation on disk through data coding and dense-packing
- Overlapping projections of tables for efficient query processing
- Efficient snapshot isolation

TECHNICAL DETAILS

Data Model

Projections:

- Projections contain only specific columns from a base table to serve targeted query needs.
- Access only the necessary columns, reducing disk I/O and speeding up query execution.
- Each projection is pre-sorted based on the most frequently queried columns to facilitate rapid data retrieval.
- Projections allow for more efficient data compression, as columns with similar data types or patterns are stored together.

Name	Age	Dept	Salary
Bob	25	Math	10K
Bill	27	EECS	50K
Jill	24	Biology	80K

Table 1: Sample EMP data

```
EMP1(name, age| age)
EMP2(dept, age, DEPT.floor| DEPT.floor)
EMP3(name, salary| salary)
DEPT1(dname, floor| floor)
```

Data Model

Horizontal Partitioning:

- Data is divided into segments or chunks, allowing for more manageable and efficient processing.
- Enables the database to perform operations on different segments simultaneously, reducing overall query time.

Join Indices:

- Map the relationships between rows in different projections that are related by common attributes.
- Provide a fast pathway to access joined data, reducing the time to execute complex queries.

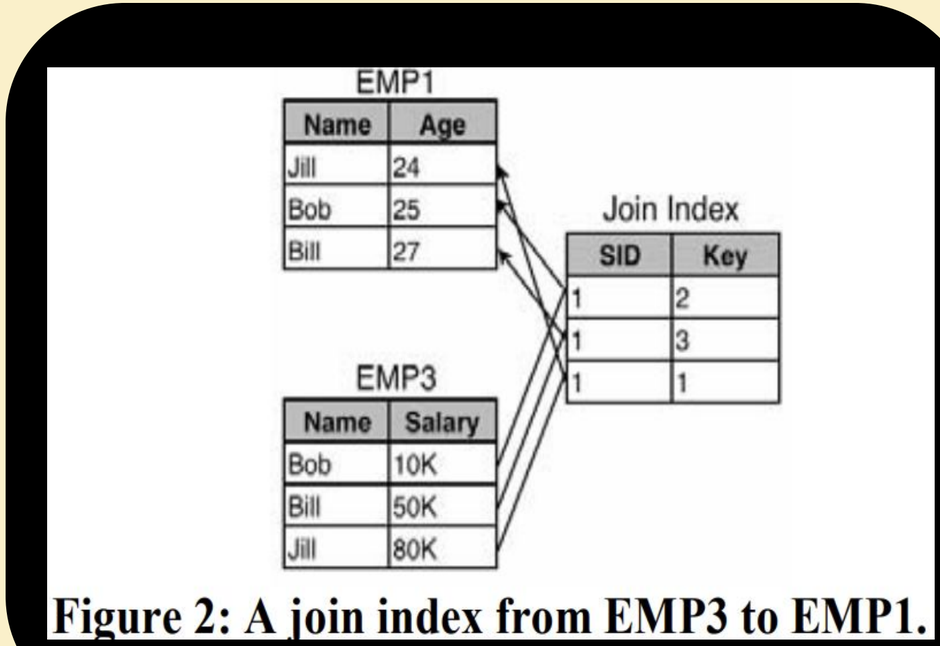


Figure 2: A join index from EMP3 to EMP1.

Read-optimized Column Store

Type 1 - Self-order:

- Best suited for **Low Cardinality**, where a limited number of distinct values occur repeatedly
- Data is stored as sequence of triples: **(*v*-value, *f*-start position, *n*-times appears)**
- **Example:** group of 4's appears in positions 12-18 can be stored as **(4, 12, 7)**

Type 2 - Foreign-order:

- Best suited for **few distinct values**
- Represented by tuple, **(*v*-value, *b*-bitmap)**
- Bitmap indicates the positions in which the value is stored.
- **Example:** A column of integers **0,0,1,1,2,1,0,2,1** can be encoded to three pairs: **(0, 110000100), (1, 001101001), and (2,000010010)**

Read-optimized Column Store

Type 3 - Self-order:

- Best suited for **many distinct** but sequentially related values.
- Only the **first value** and **subsequent deltas** (differences) are stored
- **Example:** A column of integers **1,4,7,7,8,12** can be encoded **(1,3,3,0,1,4)**

Type 4 - Foreign-order:

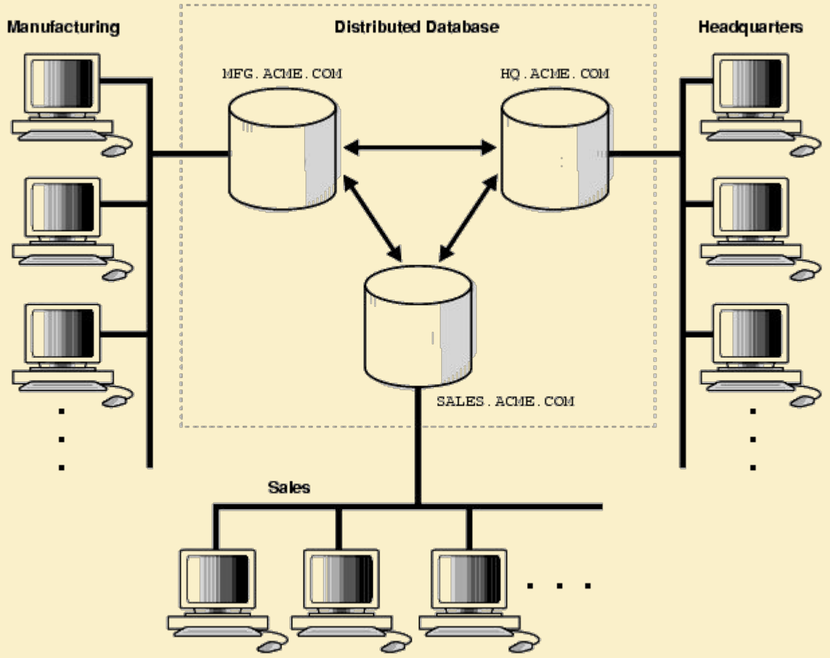
- Best suited for **many distinct values**
- Still under investigation in the paper for possible compression techniques.

Write-optimized Column Store

- Same logical design as RS but differs in physical representation to allow efficient updates.
- Does not compress data due to the transactional nature of updates.
- Each column in WS is indexed by a B-tree for maintaining the sort order and facilitates the efficient lookup of storage keys
- Designed to handle real-time data modifications, supporting transactional applications with high update frequencies.
- Seamlessly collaborates with the Read-optimized Store, allowing data to be moved to RS for query efficiency after initial writes and updates.

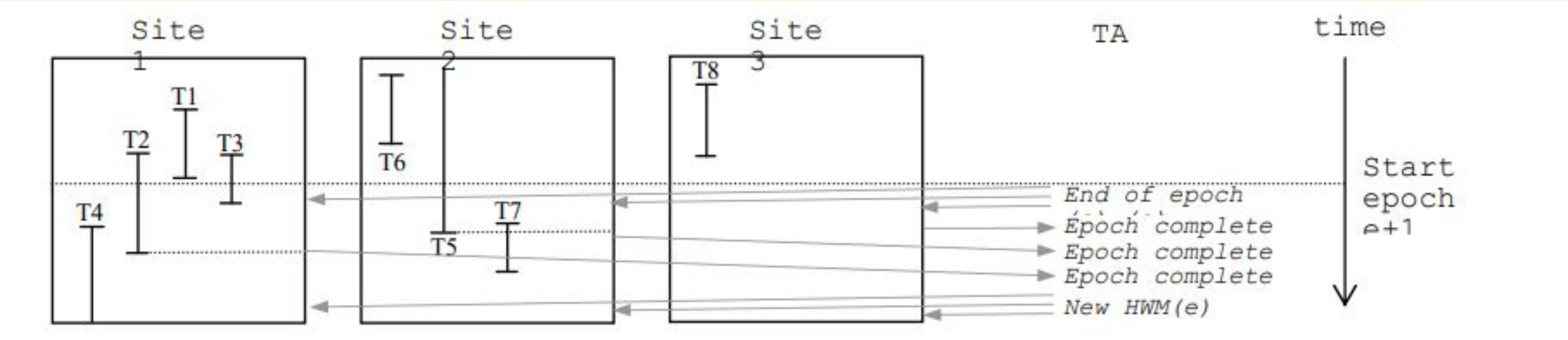
Storage Management

- How do we store projections across many compute units across our network?
- C-store uses a concept called co-location



Update and Transactions

- Snapshot Isolation is a transaction that allows access to past states of the database
- Snapshot isolation uses “Maintaining the High Water Mark” which takes snapshots of database at certain epochs
- Locking-based concurrency for updates use 2pl for write operations
- Distributed Commit processing uses a master to assign locks to a particular transaction
- Distributed commit processing differs from 2PC in that it doesn’t send a prepare message
- If the master sends a commit message they release all locks and delete the UNDO log.
- Transaction Rollback is used when you abort



Update and Transactions

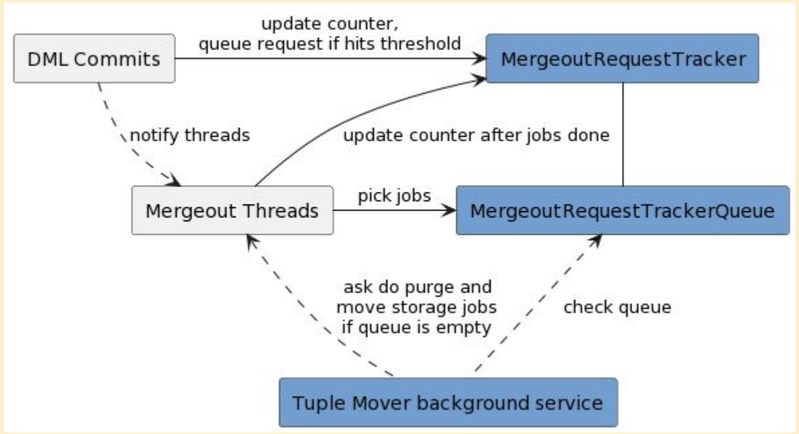
- Recovery processes in c-store are designed to handle various
- This explanation outlines a method for recovering data in a Writeable Store (WS) segment of a database system, specifically within the context of C-Store, which uses a combination of Writeable Store (WS) for updates and Read-optimized Store (RS) for querying.
- When a WS segment needs to be recovered at a site, the system first checks for projections that cover the required key range and have an insertion timestamp indicating they are up-to-date or more recent than the lost data. This is done to identify which data segments can be used to restore the missing information.

```
SELECT desired_fields,
       insertion_epoch,
       deletion_epoch
FROM recovery_segment
WHERE insertion_epoch > t_lastmove(Sr)
      AND insertion_epoch <= HWM
      AND deletion_epoch = 0
          OR deletion_epoch >= LWM
      AND sort_key in K
```

Tuple Mover

- The tuple mover's role involves transferring blocks of tuples from a Write Segment (WS) to a corresponding Read Segment (RS) while updating any join indexes during the process.
- It functions as an automated task that scans for suitable segment pairs and, upon finding one, initiates a merge-out process (MOP) on the identified WS and RS segment pair.

- The MOP segregates records in the WS segment based on their insertion time relative to the Low Water Mark (LWM), categorizing them into two groups:



C-Store Query Execution

- Query operators and plan format
- 10 different node types which are capable of interacting with data in the form of projections, columns, or bitstrings, and additional arguments.
- Many operators such as decompress, select, mask, and project
- Query optimization process
- Plan construction such as employing cost-based estimations to build efficient query plans
- cost based estimation takes into account the costs in terms of IO and memory usage of input data
- Decision making to make critical decisions

EVALUATION

EVALUATION

HYPOTHESIS

- A column store architecture can be more efficient for read-mostly applications compared to traditional row store architectures.
- A hybrid architecture with a read and write-optimized component, heavily compressed columns with a column-oriented executor/optimizer, and redundant storage in overlapping projections can achieve high performance on both warehouse-style queries and OLTP-style transactions.
- The combination of materialized views, snapshot isolation, transaction management, and high availability techniques in C-Store can result in improved performance, K-safety, efficient retrieval, and high-performance transactions.

JUSTIFICATION AND BASELINES

- **Design principles behind C-Store:** benefits of a column store architecture for read-mostly applications, the advantages of a hybrid architecture for balancing write and read operations, and the use of snapshot isolation for query performance.

- **Performance data:** performance comparison using TCP-H queries to demonstrate that C-Store is substantially faster than popular commercial products.

- **Comparisons with existing systems:** compared C-Store with other systems that store data by columns, such as Sybase IQ, Addamark, Monet, and KDB, highlighting the unique features of C-Store like overlapping materialized projections, hybrid architecture and storing tables with indexing and record identifiers. Also compared with data mirrors, which goal is better query performance, while C-Store achieves better performance on update workloads and queries.

C-Store	Row Store	Column Store
1.987 GB	4.480 GB	2.650 GB

Query	C-Store	Row Store	Column Store
Q1	0.03	6.80	2.24
Q2	0.36	1.09	0.83
Q3	4.90	93.26	29.54
Q4	2.09	722.90	22.23
Q5	0.31	116.56	0.93
Q6	8.50	652.90	32.83
Q7	2.54	265.80	33.24

C-Store	Row Store	Column Store
1.987 GB	11.900 GB	4.090 GB

Query	C-Store	Row Store	Column Store
Q1	0.03	0.22	2.34
Q2	0.36	0.81	0.83
Q3	4.90	49.38	29.10
Q4	2.09	21.76	22.23
Q5	0.31	0.70	0.63
Q6	8.50	47.38	25.46
Q7	2.54	18.47	6.28

CONCLUSION

CONCLUSION

- Aimed at the “read-mostly” DBMS market.
- A column store representation, with an associated query execution engine.
- A data model consisting of overlapping projections of tables, unlike the standard fare of tables, secondary indexes, and projections.
- A hybrid architecture that allows transactions on a column store.

STUDY QUESTIONS

STUDY QUESTIONS

1. How does the hybrid architecture of C-Store impact query performance compared to traditional row-oriented and column-oriented DBMSs?

2. How does C-Store ensure high availability and data redundancy through its overlapping projections and K-safety mechanisms?