Overview of Concurrency in L-Store:
2VCC - Two-version Concurrency Control

Mohammad Sadoghi

Exploratory Systems Lab
University of California, Davis

ECS165a - Winter 2023
1. Data Velocity: Index Maintenance

2. Data Volume: MVCC Concurrency

3. Decentralized & Democratic Data Platform

4. References
Extending Storage Hierarchy with Indirection Layer

Operational Data
Volume & Velocity
(Storage Architecture, Indexing & Concurrency)

Index Maintenance
VLDB'13

SSD
Reduction Index maintenance: Velocity Dimension

Observed Trends

In the absence of in-place updates in operational multi-version databases, the cost of index maintenance becomes a major obstacle to cope with data velocity.
Reducing Index maintenance: Velocity Dimension

Observed Trends

In the absence of in-place updates in operational multi-version databases, the cost of index maintenance becomes a major obstacle to cope with data velocity.

Extending storage hierarchy (using fast non-volatile memory) with *an extra level of indirection* in order to
Reducing Index maintenance: Velocity Dimension

Observed Trends

In the absence of in-place updates in operational multi-version databases, the cost of index maintenance becomes a major obstacle to cope with data velocity.

Extending storage hierarchy (using fast non-volatile memory) with an extra level of indirection in order to Decouple Logical and Physical Locations of Records to Reduce Index Maintenance
Traditional Multi-version Indexing: Updating Records

Updating random leaf pages
Traditional Multi-version Indexing: Updating Records

Updating random leaf pages
Traditional Multi-version Indexing: Updating Records

Updating random leaf pages
Traditional Multi-version Indexing: Updating Records

Updating random leaf pages
Indirection Indexing: Updating Records

HDD

RID Index

RID Index

Eliminating random leaf-page updates
Indirection Indexing: Updating Records

The diagram illustrates the concept of indirection indexing in the context of updating records. It shows the flow of data between the RID Index and the HDD (Hard Disk Drive). The RID Index is used to guide the updates, and the HDD stores the actual data. The diagram emphasizes the process of updating records efficiently by using indirect indexing, which helps in eliminating random leaf-page updates.
Indirection Indexing: Updating Records

SSD

HDD

LID: Logical Identifier

RID: Record Identifier

Indirection Index

(LtoR Mapping)

Eliminating random leaf-page updates

Mohammad Sadoghi (UC Davis)
Eliminating random leaf-page updates
Indirection Indexing: Updating Records

Eliminating random leaf-page updates

LID: Logical Identifier
RID: Record Identifier

HDD
Tail (append-only)

SSD

Mohammad Sadoghi (UC Davis)
Indirection Indexing: Updating Records

Eliminating random leaf-page updates
Analytical & Experimental Evaluations
## Time Complexity Analysis

### Legend

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td>Number of indexes</td>
</tr>
<tr>
<td>$LB$</td>
<td>LIDDBlock size</td>
</tr>
<tr>
<td>$M$</td>
<td>Number of matching records</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Imm. SSD</th>
<th>Def. SSD</th>
<th>Imm. HDD</th>
<th>Def. HDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>Deletion</td>
<td>0</td>
<td>0</td>
<td>$2 + K$</td>
<td>$\leq 1 + K$</td>
</tr>
<tr>
<td></td>
<td><strong>Single-attr. update</strong></td>
<td><strong>0</strong></td>
<td><strong>0</strong></td>
<td>$3 + K$</td>
<td>$\leq 2 + K$</td>
</tr>
<tr>
<td></td>
<td>Insertion</td>
<td>0</td>
<td>0</td>
<td>$1 + K$</td>
<td>$\leq 1 + K$</td>
</tr>
<tr>
<td></td>
<td>Search Uniq.</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Search Mult.</td>
<td>0</td>
<td>0</td>
<td>$1 + M$</td>
<td>0</td>
</tr>
<tr>
<td>Indirection</td>
<td>Deletion</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>$\leq 3$</td>
</tr>
<tr>
<td></td>
<td><strong>Single-attr. update</strong></td>
<td><strong>2</strong></td>
<td><strong>0</strong></td>
<td><strong>4</strong></td>
<td>$\leq 3$</td>
</tr>
<tr>
<td></td>
<td>Insertion</td>
<td>$2 + 2K$</td>
<td>$2K/LB$</td>
<td>1</td>
<td>$\leq 1 + 2K/LB$</td>
</tr>
<tr>
<td></td>
<td>Search Uniq.</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Search Mult.</td>
<td>$1 + M$</td>
<td>0</td>
<td>$1 + M$</td>
<td>0</td>
</tr>
</tbody>
</table>
Experimental Setting

- **Hardware:**
  - (2 × 8-core) Intel(R) Xeon(R) CPU E7-4820 @ 2.00GHz, 32GB, 2 × HDD, SSD Fusion-io

- **Software:**
  - Database: IBM DB2 9.7
  - Prototyped in a commercial proprietary database
  - Prototyped in Apache Spark by UC Berkeley
  - LIBGist v.1.0: Generalized Search Tree C++ Library by UC Berkeley (5K LOC) (Predecessor of Generalized Search Tree (GiST) access method for PostgreSQL)
  - **LIBGist\textsuperscript{mv} Prototype:** Multi-version Generalized Search Tree C++ Library over LIBGist supporting Indirection/LIDBlock/DeltaBlock (3K LOC)

- **Data:**
  - TPC-H benchmark
  - Microsoft Hekaton micro micro benchmark
Indirection: Effect of Indexes in Operational Data Stores

TPC-H: all tables; Scale Factor: 20

Substantially improving the update time ...

Mohammad Sadoghi (UC Davis)
Indirection: Effect of Indexes in Operational Data Stores

... Consequently affording more indexes and significantly reducing the query time
1. Data Velocity: Index Maintenance

2. Data Volume: MVCC Concurrency

3. Decentralized & Democratic Data Platform

4. References
Introducing Multi-version Concurrency Control

Data Volume
(Storage Architecture, Indexing & Concurrency)

2VCC
VLDB'14

SSD
Observed Trends

In operational multi-version databases, there is a tremendous opportunity to avoid clashes between readers (scanning a large volume of data) and writers (frequent updates).
Generalized Concurrency Control: Volume Dimension

Observed Trends

In operational multi-version databases, there is a tremendous opportunity to avoid clashes between readers (scanning a large volume of data) and writers (frequent updates).

Introducing a (latch-free) two-version concurrency control (2VCC) by extending indirection mapping (i.e., central coordination mechanism) and exploiting existing two-phase locking (2PL) in order to...
In operational multi-version databases, there is a tremendous opportunity to avoid clashes between readers (scanning a large volume of data) and writers (frequent updates).

Introducing a (latch-free) \textit{two-version concurrency control (2VCC)} by extending indirection mapping (i.e., central coordination mechanism) and exploiting existing two-phase locking (2PL) in order to Decouple Readers/Writers to Reduce Contention (Pessimistic and Optimistic Concurrency Control Coexistence)
Recap: Indirection technique for reducing index maintenance
2V-Indirection Indexing: Updating Records

Extending the indirection to committed/uncommitted records
Extending the indirection to committed/uncommitted records.
2V-Indirection Indexing: Updating Records

Decoupling readers/writers to eliminate contention
Decoupling readers/writers to eliminate contention
2V-Indirection Indexing: Updating Records

Decoupling readers/writers to eliminate contention
Overview of Two-version Concurrency Control Protocol

Two-phase locking (2PL) consisting of growing and shrinking phases
Overview of Two-version Concurrency Control Protocol

Two-phase locking (2PL) consisting of growing and shrinking phases
Overview of Two-version Concurrency Control Protocol

Two-phase locking (2PL) consisting of growing and shrinking phases
Overview of Two-version Concurrency Control Protocol

Growing Phase: Acquiring Locks

Shrinking Phase: Releasing Locks

Extending 2PL with certify phase
Overview of Two-version Concurrency Control Protocol

- **Growing Phase:** Acquiring Locks
- **Shrinking Phase:** Releasing Locks
- **Certify Phase:** Upgrading Locks

Exclusive locks held for shorter period (inherently optimistic)

<table>
<thead>
<tr>
<th>Mohammad Sadoghi (UC Davis)</th>
<th>L-Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirection</td>
<td>2VCC</td>
</tr>
<tr>
<td>------------</td>
<td>-----</td>
</tr>
</tbody>
</table>
Overview of Two-version Concurrency Control Protocol

Growing Phase: Acquiring Locks

Shrinking Phase: Releasing Locks

Exclusive Locks

Certify Phase: Upgrading Locks

Exclusive locks held for shorter period (inherently optimistic)
Overview of Two-version Concurrency Control Protocol

Growing Phase: Acquiring Locks
Shrinking Phase: Releasing Locks
Certify Phase: Upgrading Locks

Relaxed exclusive locks to allow speculative reads (increased optimism)
Overview of Two-version Concurrency Control Protocol

Trade-offs between blocking (i.e., locks) vs. non-blocking (i.e., read counters)
Experimental Analysis
**2VCC: Effect of Parallel Update Transactions**

Substantial gain by reducing the read/write contention & using non-blocking operations

---

**Update Only Workload; High Contention**

- **Update Execution Time in seconds**
- **Number of Parallel Transactions**
- **Improvement ratio**

<table>
<thead>
<tr>
<th>Number of Parallel Transactions</th>
<th>Single-version</th>
<th>Multi-version</th>
<th>Improvement ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
</tr>
<tr>
<td>16</td>
<td>3.4</td>
<td>3.4</td>
<td>3.4</td>
</tr>
<tr>
<td>24</td>
<td>6.2</td>
<td>6.2</td>
<td>6.2</td>
</tr>
<tr>
<td>32</td>
<td>8.6</td>
<td>8.6</td>
<td>8.6</td>
</tr>
<tr>
<td>64</td>
<td>43.4</td>
<td>43.4</td>
<td>43.4</td>
</tr>
</tbody>
</table>
2VCC: Effect of Parallel Update Transactions

Substantial gain by reducing the read/write contention & using non-blocking operations

Mohammad Sadoghi (UC Davis)
1 Data Velocity: Index Maintenance

2 Data Volume: MVCC Concurrency

3 Decentralized & Democratic Data Platform

4 References
Recap: Data Management Challenges at Microscale

OLTP and OLAP data are isolated at microscale.
Recap: Data Management Challenges at Microscale

First step is to unify OLTP and OLAP
Platform Scaling: Data Partitioning

Moving towards distributed environment
Platform Scaling: Non-blocking Agreement Protocols

First Transmit and then Commit
(Message Redundancy)

Data Partitioning
(within in a data center)

OLAP+OLTP
(Read & Write-optimized)

Reports

Message redundancy vs. latency trade-offs [EasyCommit, EDBT’18]
Central Control: Data Gate Keeper

Conform to trusting the central authority and governance
Decentralized Control: Removing Data Barrier

Seek trust in *decentralized* and *democratic* governance [PoE (EDBT’21), RCC (ICDE’21)]
Democratic Control: Removing Trust Barrier

Seek trust in **decentralized** and **democratic** governance [PoE (EDBT’21), RCC (ICDE’21)]
Global-scale Reliable Platform over Unreliable Hardware

- OLAP+OLTP (Read & Write-optimized)
- Data Partitioning (within in a data center)
- Reports

Self-managed infrastructure
Global-scale Reliable Platform over Unreliable Hardware

Cloud-managed infrastructure (trust the provider)
Global-scale Reliable Platform over Unreliable Hardware

Cloud-managed infrastructure (trust the provider)
Global-scale Reliable Platform over Unreliable Hardware

Light-weight, fault-tolerant, trusted middleware [Blockplane, (ICDE’18)]
Global-scale Reliable Platform over Unreliable Hardware

Global Scale fault-tolerant protocols [GeoBFT (VLDB’20), Delayed Replication (ICDT’20)]
Questions?

Thank you!

Exploratory Systems Lab (ExpoLab)
Website: https://expolab.org/
Related Publications (Patents Omitted)