L-Store: Towards a Unified OLTP and OLAP over a Secure Platform

Mohammad Sadoghi

Exploratory Systems Lab
University of California, Davis

University of Waterloo
October 15, 2018
Data Management at Macroscale
Data Management at Microscale: Volume & Velocity

OLTP (Write-optimized)

Sales

Data Velocity
Data Management at Microscale: Volume & Velocity

OLAP (Read-optimized)

Data is Stale

Extract-Transform-Load (ETL)

Data Velocity

OLTP (Write-optimized)

Data Volume

Walmart
Data Management at Microscale: Volume & Velocity

OLAP (Read-optimized) → Extract-Transform-Load (ETL) → OLTP (Write-optimized)

Data Volume

OLAP (Read-optimized) → Reports

Data Velocity

OLTP (Write-optimized) → Sales

Walmart

Mohammad Sadoghi (UC Davis)
Big Data Landscape

Copyright © 2012 Dave Feinleib
dave@vcc.com
blogs.forbes.com/davefeinleib
One Size Does not Fit All As of 2017
Data Management at Microscale: Volume & Velocity

OLAP+OLTP
(Read & Write-optimized)

Walmart

Reports

Sales
Data Management at Microscale: Volume & Velocity

OLAP+OLTP (Read & Write-optimized)

Sales

Walmart

OLAP+OLTP
(Read & Write-optimized)
Data Management at Microscale: Volume & Velocity

OLAP+OLTP (Read & Write-optimized)

Detect Patterns (Data Streams)

Sales

Reports

Walmart

Mohammad Sadoghi (UC Davis)
Data Management at Microscale: Volume & Velocity

OLAP+OLTP (Read & Write-optimized)

Detect Patterns (Data Streams)

Walmart

Reports

Sales

Motivations

Big Picture

Indirection

2VCC

QueCC

L-Store

Evaluation

Vision: L-Store

Conclusions

References

Mohammad Sadoghi (UC Davis)
1. Data Management at Microscale
2. Data Management at Microscale
3. Data Velocity: Index Maintenance
4. Data Volume: MVCC Concurrency
5. Data Volume: Coordination-free Concurrency
6. Combining Volume & Velocity: Lineage-based Storage Architecture
7. Data at Macroscale: Decentralized & Democratic Data Platform
8. Conclusions
9. References
Big Picture

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

Data Stream Velocity
(Query Indexing)
Big Picture

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

Data Stream Velocity
(Query Indexing)

BE-Tree
SIGMOD'11
TODS'13

APP
DEBS'11
Big Picture

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

Data Stream Velocity
(Query Indexing)

Compressed Stream ICDE'14
XML/XPath EDBT'11
BE-Tree
SIGMOD'11
Top-k ICDE'12
APP DEBS'11
TODS'13
ICCS'17
Distributed Top-k
TKDE'15
ICDE'15
SIGMOD'15 (Demo)
Middleware'16

Distributed Transaction (workflow execution)
Big Picture

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

Index Maintenance
VLDB'13

NVM
RDMA

Compressed Stream
ICDE'14

Top-k
ICDE'12

Distributed Transaction
(workflow execution)

Data Stream Velocity
(Query Indexing)

BE-Tree
SIGMOD'11
TODS'13

APP
DEBS'11

XML/XPath
EDBT'11

FPGA

SQL Queries
ICDE'15 (Demo)

ICDE'15, SIGMOD Record'15, ICDE'16, ATC'16, ICDCS'17

SQL Queries
(static)

VLDB'13 (Demo)

Boolean Expressions
VLDB'10 (Demo)
DEBS'11 (Demo)
DaMoN'11

ICDE'15, SIGMOD Record'15, ICDE'16, ATC'16, ICDCS'17

Mohammad Sadoghi (UC Davis)

L-Store
U. Waterloo'18 6 / 49
Big Picture

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

- Index Maintenance
  VLDB'13
- QueCC
  Middleware'18
- EasyCommit
  EDBT'18
- 2VCC
  VLDB'14

Data Stream Velocity
(Query Indexing)

- Compressed Stream
  ICDE'14
- XML/XPath
  EDBT'11
- Top-k
  ICDE'12

- Top-k
  ICDE'15
- Distributed
  Middleware'18
- EasyCommit
  VLDB'15
- BE-Tree
  SIGMOD'11
- SIGMOD'15 (Demo)
- Middleware'16
- TODS'13
- DEBS'11
- APP

- SQL Queries
  (static)
  ICDE'12 (Demo)
- SQL Queries
  (dynamic)
  VLDB'13 (Demo)
- Boolean Expressions
  VLDB'10 (Demo)
- DEBS'11 (Demo)
- DaMon'11

ICDE'15, SIGMOD Record'15, ICDE'16, ATC'16, ICDCS'17

NVM
RDMA

Mohammad Sadoghi (UC Davis)
Big Picture

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

Index Maintenance VLDB'13
EasyCommit EDBT'18
QueCC Middleware'18

In-memory Key Value Store Middleware'16
2VCC VLDB'14

Hybrid Storage CASCON'14

NVM RDMA

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

BE-Tree SIGMOD'11 TODS'13
APP DEBS'11

Data Stream Velocity (Query Indexing)

SQL Queries (static) ICDE'12 (Demo)
SQL Queries (dynamic) VLDB'13 (Demo)

FPGA

Compressed Stream ICDE'14
XML/XPath EDBT'11

Top-k ICDE'12
Distributed ICS'17 MV'17

ICDE'15, SIGMOD Record'15, ICDE'16, ATC'16, ICDCS'17

Dr. Mohammad Sadoghi (UC Davis)
L-Store
U. Waterloo'18
Big Picture

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

Unifying OLTP & OLAP
EDBT'18, VLDBJ'16, ICDCS'16, 30+ Patents

Index Maintenance
VLDB'13

EasyCommit
EDBT'18

QueCC
Middleware'18

2VCC
VLDB'14

In-memory Value Store
Middleware'16

CaSSanDra
ICDE'14

Hybrid Storage
CASCON'14

NVM
RDMA

Compressed Stream
ICDE'14

XML/XPath
EDBT'11

Top-k
ICDE'12

Distributed Transaction
(workflow execution)

Data Stream Velocity
(Query Indexing)

BE-Tree
SIGMOD'11
TODS'13

BE-Tree
SIGMOD'11
TODS'13

APP
DEBS'11

SQL Queries
ICDE'15
(Static)

SQL Queries
ICDE'15
(Demo)

Boolean Expressions
VLDB'10 (Demo)

BooL
DaMon'11

FPGA

ICDE'15, SIGMOD Record'15, ICDE'16, ATC'16, ICDCS'17

Mohammad Sadoghi (UC Davis)
Deep Dive: Unifying OLTP & OLTP

Modern Hardware (SSDs & HTM)
- Hierarchical Caching (Bufferpool Extension)
- MVCC Data Size Independence on HTM

Unifying OLAP & OLTP
- Rendezvous-based Optimistic Concurrency
- Transient Snapshot (In-page Log)
- Deferred Updates (Query Rewriting)
- Pre-play Concurrency (Storage Hierarchy)

Concurrency
- Batching Queries & Inserting
- Range Queries Support (Latch-free R-Hash)
- QueCC (Coordination-free)

Storage (Columnar)
- Hierarchical Bufferpool (latch-free)
- Synopsis Alignment
- Delta-Compression on Multi-Version Databases
- Data & Synopsis Unification
- Efficient Update (Lineage-based Storage)

Indexing
- Index Maintenance (Indirection Technique)

Efficient Update
- (Lineage-based Storage)

Publication Summary
2 VLDB, 1 SIGMOD, 3 EDBT, 1 VLDBJ, 1 ICDE, 1 ICDCS, 2 Middleware, 30+ Patents

Mohammad Sadoghi (UC Davis)
1. Data Management at Microscale

2. Data Management at Microscale

3. Data Velocity: Index Maintenance

4. Data Volume: MVCC Concurrency

5. Data Volume: Coordination-free Concurrency

6. Combining Volume & Velocity: Lineage-based Storage Architecture

7. Data at Macroscale: Decentralized & Democratic Data Platform

8. Conclusions

9. References
Extending Storage Hierarchy with Indirection Layer

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

Index Maintenance
VLDB'13

SSD
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

Mohammad Sadoghi (UC Davis)
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
Indirection Indexing: Updating Records

LID: Logical Identifier
RID: Record Identifier

SSD
HDD

Indirection Index (LtoR Mapping)
Indirection Indexing: Updating Records

Eliminating random leaf-page updates
Indirection Indexing: Updating Records

Eliminating random leaf-page updates
Indirection Indexing: Updating Records

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

<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><strong>Base</strong></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>0</td>
<td>0</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><strong>Indirection</strong></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>2</td>
<td>0</td>
<td>4</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>

### Legend

- $K$: Number of indexes
- $LB$: LIDBlock size
- $M$: Number of matching records
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 benchmark
Indirection: Effect of Indexes in Operational Data Stores

TPC-H: all tables; Scale Factor: 20

Substantially improving the update time ...
... Consequently affording more indexes and significantly reducing the query time
1. Data Management at Microscale
2. Data Management at Microscale
3. Data Velocity: Index Maintenance
4. Data Volume: MVCC Concurrency
5. Data Volume: Coordination-free Concurrency
6. Combining Volume & Velocity: Lineage-based Storage Architecture
7. Data at Macroscale: Decentralized & Democratic Data Platform
8. Conclusions
9. References
Introducing Multi-version Concurrency Control

Data Volume
(Storage Architecture, Indexing & Concurrency)

2VCC
VLDB'14

SSD
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).
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
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

Decouple Readers/Writers to Reduce Contention

(Pessimistic and Optimistic Concurrency Control Coexistence)
2V-Indirection Indexing: Updating Records

Recap: Indirection technique for reducing index maintenance

LID: Logical Identifier
RID: Record Identifier

Indirection Mapping
2V-Indirection Indexing: Updating Records

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

Motivations

Big Picture

Indirection

2VCC

QueCC

L-Store

Evaluation

Vision: L-Store

Conclusions

References

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

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

Decoupling readers/writers to eliminate contention

Mohammad Sadoghi (UC Davis)
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

Growing Phase: Acquiring Locks
Shrinking Phase: Releasing Locks
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)
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

**Certify Phase:**
- Upgrading Locks

**Shrinking Phase:**
- Releasing Locks

**Update Intent**
- Exclusive Locks (relaxed)
- Wait Dependency
- Speculative Reads
- Shared Locks

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

**Growing Phase:**
- Acquiring Locks

**Shrinking Phase:**
- Releasing Locks

**Exclusive Locks**

**Certify Phase:**
- Upgrading Locks

**Lock Waits** (counter + queue)

**Blocking**

**Trade-offs between blocking (i.e., locks) vs. non-blocking (i.e., read counters)**
Experimental Analysis
Substantial gain by reducing the read/write contention & using non-blocking operations
2VCC: Effect of Parallel Update Transactions

Substantial gain by reducing the read/write contention & using non-blocking operations
1. Data Management at Microscale
2. Data Management at Microscale
3. Data Velocity: Index Maintenance
4. Data Volume: MVCC Concurrency
5. **Data Volume: Coordination-free Concurrency**
6. Combining Volume & Velocity: Lineage-based Storage Architecture
7. Data at Macroscale: Decentralized & Democratic Data Platform
8. Conclusions
9. References
Introducing Coordination-free Concurrency Control

Data Volume (Storage Architecture, Indexing & Concurrency)

QueCC
Middleware'18

SSD
Observed Trends

In operational databases, the use of pre-compiled stored procedures is predominant. There is a tremendous opportunity to exploit transaction prior knowledge to eliminate the need for coordination.
Confrontation-free Concurrency Control

Observed Trends
In operational databases, the use of pre-compiled stored procedures is predominant. There is a tremendous opportunity to exploit transaction prior knowledge to eliminate the need for coordination.

Is it possible to have concurrent execution over shared data (not limited to partitionable workloads) without having any concurrency controls?
Confrontation-free Concurrency Control

Observed Trends

In operational databases, the use of pre-compiled stored procedures is predominant. There is a tremendous opportunity to exploit transaction prior knowledge to eliminate the need for coordination.

Is it possible to have concurrent execution over shared data (not limited to partitionable workloads) without having any concurrency controls?

Introducing a *queue-oriented, control-free concurrency (QueCC)* based on two parallel & independent phases of priority-driven planning & execution.
Observed Trends

In operational databases, the use of pre-compiled stored procedures is predominant. There is a tremendous opportunity to exploit transaction prior knowledge to eliminate the need for coordination.

Is it possible to have concurrent execution over shared data (not limited to partitionable workloads) without having any concurrency controls?

Introducing a *queue-oriented, control-free concurrency (QueCC)* based on two parallel & independent phases of priority-driven planning & execution. 

**Execution and Synchronization Decoupling**
Queue-oriented, Control-free Concurrency (QueCC)

Execution & Synchronization Decoupling: Deterministic priority-based planning followed by queue-oriented execution
Queue-oriented, Control-free Concurrency (QueCC)

Execution & Synchronization Decoupling: Deterministic priority-based planning followed by queue-oriented execution
Queue-oriented, Control-free Concurrency (QueCC)

Execution & Synchronization Decoupling: Deterministic priority-based planning followed by queue-oriented execution
Queue-oriented, Control-free Concurrency (QueCC)

Execution & Synchronization Decoupling: Deterministic priority-based planning followed by queue-oriented execution

Mohammad Sadoghi (UC Davis)
Queue-oriented, Control-free Concurrency (QueCC)

Execution & Synchronization Decoupling: Deterministic priority-based planning followed by queue-oriented execution
Experimental Analysis
QueCC: Effect of Parallel Update Transactions

Avoiding thread coordination & eliminating all execution-induced aborts
Unifying OLTP and OLAP

Operational Data
Volume & Velocity
(Storage Architecture, Indexing & Concurrency)
Unifying OLTP and OLAP: Velocity & Volume Dimensions

Observed Trends

In operational databases, there is a pressing need to close the gap between the write-optimized layout for OLTP (i.e., row-wise) and the read-optimized layout for OLAP (i.e., column-wise).
Observed Trends

In operational databases, there is a pressing need to close the gap between the write-optimized layout for OLTP (i.e., row-wise) and the read-optimized layout for OLAP (i.e., column-wise).

Introducing a *lineage-based storage architecture*, a contention-free update mechanism over a native columnar storage in order to
Unifying OLTP and OLAP: Velocity & Volume Dimensions

Observed Trends

In operational databases, there is a pressing need to close the gap between the write-optimized layout for OLTP (i.e., row-wise) and the read-optimized layout for OLAP (i.e., column-wise).

Introducing a *lineage-based storage architecture*, a contention-free update mechanism over a native columnar storage in order to lazily and independently stage stable data from a write-optimized layout (i.e., OLTP) into a read-optimized layout (i.e., OLAP).
Storage Layout Conflict

Write-optimized (i.e., uncompressed & row-based) vs. read-optimized (i.e., compressed & column-based) layouts
Lineage-based Storage Architecture (LSA): Intuition

Physical Update Independence: De-coupling data & its updates (reconstruction via in-page lineage tracking and lineage mapping)
Lineage-based Storage Architecture (LSA): Intuition

- **Monotonically Increasing Lineage**: (updates are assigned RIDs in an increasing order)
- **Base Pages (Read-only)**
- **Tail Pages (Append-only)**
- **Latest Version**: (monotonically increasing RIDs)
- **Append-only Updates**: (physical update independence)
- **Base Version**: (anchored RIDs)
- **Lineage Mapping**: (indirection layer, stable LID-to-RID mapping)
- **In-page Lineage Tracking**: Points to Stable LIDs (i.e., anchored RID)

**Physical Update Independence**: De-coupling data & its updates (reconstruction via in-page lineage tracking and lineage mapping)
Lineage-based Storage Architecture (LSA): Intuition

Monotonically Increasing Lineage
(updates are assigned RIDs in an increasing order)

Base Pages
(Read-only)

Tail Pages
(Append-only)

Latest Version
(monotonically increasing RIDs)

Append-only Updates
(physical update independence)

Monotonically Increasing In-page Lineage

Points to Stable LIDs
(i.e., anchored RID)

Lazy Update Consolidation
(snapshot reconstruction via lineage mapping & in-page tracking)

In-page Lineage Tracking

Data Block RIDs Remain Unchanged
(stable reference, anchored RIDs)

Consolidated Data
(Read-only)

Base Pages
(Read-only)

Index

Base Version
(stable anchored RIDs)

Lineage Mapping
(indirection layer, stable LID-to-RID mapping)

Mohammad Sadoghi (UC Davis)
Overview of the lineage-based storage architecture
*(base pages and tail pages are handled identically at the storage layer)*
Records are range-partitioned and compressed into a set of ready-only **base pages** (accelerating analytical queries)
Recent updates for a range of records are clustered in their *tails pages* (transforming costly point updates into an amortized analytical-like query)
Recent updates for a range of records are clustered in their **tails pages**
(transforming costly point updates into an amortized analytical-like query)
Recent updates are strictly appended, uncompressed in the pre-allocated space (eliminating the read/write contention)
Achieving (at most) 2-hop access to the latest version of any record (avoiding read performance deterioration for point queries)
L-Store: Detailed Design

Achieving (at most) 2-hop access to the latest version of any record
(avoiding read performance deterioration for point queries)
Achieving (at most) 2-hop access to the latest version of any record (avoiding read performance deterioration for point queries)
L-Store: Contention-free Merge

Contention-free merging of only stable data: read-only and committed data
(no need to block on-going and new transactions)
L-Store: Contention-free Merge

Lazy independent merging of base pages with their corresponding tail pages (resembling a local left outer-join of the base and tail pages)
L-Store: Contention-free Merge

Asynchronous Lazy Merge
(committed, consecutives updates)

In-page, Independent Lineage Tracking

Read Optimized
(compressed, read-only pages)

Write Optimized
(uncompressed, append-only updates)

Indirection Column
(uncompressed, in-place update)

Independently tracking the lineage information within every page
(no need to coordinate merges among different columns of the same records)
L-Store: Epoch-based Contention-free De-allocation

Contention-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)
L-Store: Epoch-based Contention-free De-allocation

Contestation-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)
L-Store: Epoch-based Contention-free De-allocation

Contestion-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)
L-Store: Epoch-based Contention-free De-allocation

Epoch-based De-allocation (longest running query)

Page Directory

Read Optimized
(compressed, read-only pages)

Write Optimized
(uncompressed, append-only updates)

Asynchronous Lazy Merge

Indirection Column
(uncompressed, in-place update)

Contention-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)
L-Store: Epoch-based Contention-free De-allocation

Contention-free page de-allocation using an epoch-based approach
(no need to drain the ongoing transactions)
Experimental Analysis
Experimental Settings

- **Hardware:**
  - 2 × 6-core Intel(R) Xeon(R) CPU E5-2430 @ 2.20GHz, 64GB, 15 MB L3 cache

- **Workload:** Extended Microsoft Hekaton Benchmark
  - Comparison with *In-place Update + History* and *Delta + Blocking Merge*
  - Effect of varying contention levels
  - Effect of varying the read/write ratio of short update transactions
  - Effect of merge frequency on scan
  - Effect of varying the number of short update vs. long read-only transactions
  - Effect of varying L-Store data layouts (row vs. columnar)
  - Effect of varying the percentage of columns read in point queries
  - Comparison with log-structured storage architecture (*LevelDB*)
Effect of Varying Contention Levels

Throughput (M txns/s) versus Number of Parallel Short Update Transactions

- L-Store
- In-place Update + History
- Delta + Blocking Merge

Achieving up to 40× as increasing the update contention
Effect of Merge Frequency on Scan Performance

Mixed OLTP + OLAP Workload; Low Contention
(1 Scan + 1 Merge Threads, Page Size = 32 KB)

Scan Execution Time (in seconds)
Number of Tail Records Processed per Merge

Merge process is essential in maintaining efficient scan performance.
Effect of Mixed Workloads: Update Performance

**Mixed OLTP + OLAP Workload; Medium Contention**
(Total of 17 Threads + 1 Merge Thread, Page Size = 32 KB)

- **Lineage-based Data Store (L-Store)**
- **In-place Update + History**
- **Delta + Blocking Merge**

Eliminating latching & locking results in a substantial performance improvement

Mohammad Sadoghi (UC Davis)
Coping with tens of update threads with a single merge thread
1. Data Management at Microscale
2. Data Management at Microscale
3. Data Velocity: Index Maintenance
4. Data Volume: MVCC Concurrency
5. Data Volume: Coordination-free Concurrency
6. Combining Volume & Velocity: Lineage-based Storage Architecture
7. Data at Macroscale: Decentralized & Democratic Data Platform
8. Conclusions
9. 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
Moving towards distributed environment
Platform Scaling: Non-blocking Agreement Protocols

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

Conform to trusting the central authority and governance
Seek trust in *decentralized* and *democratic* governance [PoE (under submission)]
Democratic Control: Removing Trust Barrier

Seek trust in \textit{decentralized} and \textit{democratic} governance [PoE (under submission)]
Global-scale Reliable Platform over Unreliable Hardware

Motivations

Big Picture

Indirection

2VCC

QueCC

L-Store

Evaluation

Vision: L-Store

Conclusions

References

Self-managed infrastructure

OLAP+OLTP
(Read & Write-optimized)

Data Partitioning
(within in a data center)

Reports

Walmart

Global-scale Reliable Platform over Unreliable Hardware

OLAP+OLTP
(Read & Write-optimized)

Data Partitioning
(within in a data center)

Reports

Walmart

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, (under submission)]
Global-scale Reliable Platform over Unreliable Hardware

Fault-tolerant protocols vs. consistency models [MultiBFT, GeoBFT (under submission)]
ExpoDB: Exploratory Data Platform Architecture

A decentralized & democratic platform to unify OLTP and OLAP

Mohammad Sadoghi (UC Davis)
1. Data Management at Microscale
2. Data Management at Microscale
3. Data Velocity: Index Maintenance
4. Data Volume: MVCC Concurrency
5. Data Volume: Coordination-free Concurrency
6. Combining Volume & Velocity: Lineage-based Storage Architecture
7. Data at Macroscale: Decentralized & Democratic Data Platform
8. Conclusions
9. References
Contributions & Outlook

ExpoDB: Decentralized & Democratic Platform
- **Decentralized & Democratic Control**: PoE, MultiBFT, GeoBFT [under submission]
- **Reliability over Unreliable Hardware**: Blockplane [under submission]

Operational Data Stores: Velocity & Volume
- **Index Maintenance**: Indirection Technique [VLDB’13, VLDBJ’16]
- **Concurrency Control**: 2VCC Technique [VLDB’14, Middleware’16], EasyCommit [EDBT’18], QueCC [Middleware’18]
- **Hybrid Storage**: Enhancing Key-Value Store [VLDB’12, ICDE’14]
- **Real-time OLTP+OLAP**: Lineage-based Data Store (L-Store) [EDBT-18, ICDCS’16, 30+ Patents]

Stream Processing: Velocity
- **High-dimensional Indexing**: BE-Tree [SIGMOD’11, TODS’13], Compressed Stream Processing [ICDE’14]
- **(Distributed) Top-k Indexing**: BE*-Tree [ICDE’12, ICDCS’13, Middleware’17, ICDCS’17]
- **Hardware Acceleration**: FPGAs [VLDB’10, ICDE’12, VLDB’13, ICDE’15, SIGMOD Record’15, ICDE’16, USENIX ATC’16, ICDCS’17, ICDE’18]
- **Novel Mappings**: XML/XPath [EDBT’11], Distributed Workflow [TDKE’15, SIGMOD’15, ICDE’16, Middleware’16]
Questions?

Thank you!

Exploratory Systems Lab (ExpoLab)
Website: https://msadoghi.github.io/