L-Store: Lineage-based Storage Architectures

ECS165A: Winter 2022

Slides are adopted from Sadoghi, et al.

*L-Store: A Real-time OLTP and OLAP System, EDBT’18*
Data Management at Macroscale: The Four V’s of Big Data

Motivations
L-Store
Evaluation
Conclusions

John Doe

Anthem
BlueCross BlueShield

Walmart

MasterCard

FICO Score
The score lenders use

VISA

IRS

Mohammad Sadoghi
ECS165A
Data Management at Macroscale: The Four V’s of Big Data

Anthem BlueCross BlueShield

Walmart

MasterCard

FICO

Visa

IRS

John Doe
Data Management at Microscale: Volume & Velocity

OLTP (Write-optimized)

Data Velocity

Sales

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

OLAP (Read-optimized)

Data is Stale

OLTP (Write-optimized)

Extract-Transform-Load (ETL)

Data Velocity

Sales

Data Volume

Walmart
Data Management at Microscale: Volume & Velocity

OLAP (Read-optimized)

OLTP (Write-optimized)

Extract-Transform-Load (ETL)

Sales

Data Velocity

Walmart

Motivations

Evaluation

Conclusions

Mohammad Sadoghi

ECS165A

2 / 16
One Size Does not Fit All As of 2012

Big Data Landscape

Vertical Apps
Log Data Apps
Data As A Service
Analytics Infrastructure
Analytics and Visualization
Ad/Media Apps
Business Intelligence
Operational Infrastructure
Infrastructure As A Service
Structured Databases

Technologies

Copyright © 2012 Dave Feinleib
dave@rcdave.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

Sales

Reports

SAP HANA ORACLE

Microsoft SQL Server IBM BLU Acceleration
Write-optimized (i.e., uncompressed & row-based) vs. read-optimized (i.e., compressed & column-based) layouts
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.
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
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
Updating random leaf pages
Traditional Multi-version Indexing: Updating Records

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

Updating random leaf pages

Mohammad Sadoghi (UC Davis)
Indirection Indexing: Updating Records

The diagram shows the process of updating records using HDD and RID Index. The HDD is connected to two RID Indexes, indicating the flow of data and updates. The concept of eliminating random leaf-page updates is illustrated through this diagram, emphasizing the importance of efficient indexing for record management.
Indirection Indexing: Updating Records

HDD
RID Index

RID Index

Eliminating random leaf-page updates
Indirection Indexing: Updating Records

SSD

HDD

LID: Logical Identifier

RID: Record Identifier

Indirection Index

(LtoR Mapping)

LID Index

Eliminating random leaf-page updates

Mohammad Sadoghi (UC Davis)

L-Store

ECS165a - 2022
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

Mohammad Sadoghi (UC Davis)
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
Motivations

L-Store

Evaluation

Conclusions

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).
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)

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

Motivations

L-Store

Evaluation

Conclusions

Physical Update Independence: De-coupling data & its updates (reconstruction via in-page lineage tracking and lineage mapping)
Overview of the lineage-based storage architecture
(base pages and tail pages are handled identically at the storage layer)
L-Store: Detailed Design

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

- Motivations
- L-Store
- Evaluation
- Conclusions

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)
L-Store: Contention-free Merge

Contestion-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
(committing, consecutive updates)

In-page, Independent Lineage Tracking

Write Optimized
(uncompressed, append-only updates)

Indirection Column
(uncompressed, in-place update)

Read Optimized
(compressed, read-only pages)

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)
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)
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

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

Achieving up to $40 \times$ 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

Eliminating latching & locking results in a substantial performance improvement
Effect of Mixed Workloads: Read Performance

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

Coping with tens of update threads with a single merge thread
L-Store Key Contributions

- Unifying OLAP & OLTP by introducing lineage-based storage architecture (LSA)
- LSA is a native multi-version, columnar storage model that lazily & independently stages data from a write-optimized layout into a read-optimized one
- Contention-free merging of only stable data without blocking ongoing or incoming transactions
- Contention-free page de-allocation without draining ongoing transactions
- L-Store outperforms in-place update & delta approaches by factor of up to $8 \times$ on mixed OLTP/OLAP workloads and up to $40 \times$ on update-intensive workloads
Questions?
Thank you!

Exploratory Systems Lab (ExpoLab)
Website: https://expolab.org/