MaSM: Efficient Online Updates in Data Warehouses

by

Roberto Alcalde Diego
Introduction
Background

- Data Warehouses are optimized for read-only query performance
- When did the inserting and updating take place?
- Why does this not meet business needs anymore?
Background

• Data Warehouses are optimized for read-only query performance
• When did the inserting and updating take place?
• Why does this not meet business needs anymore?

Solution: Active Data Warehousing
What is the current situation?

• The main issue: How to efficiently execute analysis queries in the presence of online updates
• There are 2 main approaches for supporting online updates:
What is the current situation?

• The main issue: How to efficiently execute analysis queries in the presence of online updates
• There are 2 main approaches for supporting online updates:
  • In-Place Updates
    • Traditional
    • Straightforward
    • SLOW
What is the current situation?

• The main issue: How to efficiently execute analysis queries in the presence of online updates

• There are 2 main approaches for supporting online updates:
  
  • In-Place Updates
    • Traditional
    • Straightforward
    • SLOW
  
  • Differential Updates
Differential Updates - Basics

• Cache incoming updates in an in-memory buffer
• Take the cached updates into account on-the-fly during query processing -> Queries report fresh outputs
• Migrate the cached updates to the main data whenever the buffer is getting full
Differential Updates - Basics

• Cache incoming updates in an in-memory buffer
• Take the cached updates into account on-the-fly during query processing -> Queries report fresh outputs
• Migrate the cached updates to the main data whenever the buffer is getting full

• Any problems? If not, congrats we get to go home!
Differential Updates – The big problem

- If the cache is in memory, you have to choose
  - Small buffers: Small memory requirements -> you can use memory for something else. -> Many migrations
  - Big buffers: Few migrations -> memory will be occupied but you don’t have to introduce updates into the main data disks until later.

Is there a better way?
Summary of the Problem

- We want differential updates to match business needs
- We don’t want to have to compromise between memory requirements and migration overhead in the incredibly expensive way that current systems make us.

<table>
<thead>
<tr>
<th>Update Approach</th>
<th>Freshness</th>
<th>Performance</th>
<th>↓ mem overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batched</td>
<td>X</td>
<td>😊</td>
<td>😊</td>
</tr>
<tr>
<td>In place</td>
<td>😊</td>
<td>X</td>
<td>😊</td>
</tr>
<tr>
<td>In-memory differential</td>
<td>😊</td>
<td>😊</td>
<td>X</td>
</tr>
</tbody>
</table>
Cache Updates in SSDs instead of RAM

A few Specs:
- Cache size is 1% – 10% of main data size
- Both the disks and the SSD cache are searched when queries are received.
- Data is migrated to main disks when:
  - Load on the system is low
  - The SSD cache is almost full
Cache Updates in SSDs instead of RAM

- A few Specs:
  - Cache size is 1% – 10% of main data size
  - Both the disks and the SSD cache are searched when queries are received.
  - Data is migrated to main disks when:
    - Load on the system is low
    - The SSD cache is almost full
Cache Updates in SSDs instead of RAM

- A few Specs:
  - Cache size is 1% – 10% of main data size
  - Both the disks and the SSD cache are searched when queries are received.
  - Data is migrated to main disks when:
    - Load on the system is low
    - The SSD cache is almost full

- But, what are the limitations of SSDs?
Implementation
System Goals

1. **Low Query Overhead & Small Memory Footprint:**
   - Avoid having to make a lot of migrations and also avoid taking up RAM

2. **No Random SSD writes:**
   - Often leads to expensive additional operations and can degrade performance of SSD.

3. **Low total SSD writes per update:**
   - Since eventually the SSD memory will wear out, it is good to minimize writes per update to decrease wear out rate.

4. **Efficient in place migration:**
   - Previous approaches make a copy of the entire disk and then add updates, which requires twice as much disk space.

5. **ACID:**
   - Ensure that traditional concurrency control and crash recovery techniques still work.
Implementation

• What must be implemented:
  • Merging: Essentially an outer join
  • Caching: Coordinate Buffer and SSD
  • Migrating: Placing Updates in Disks

All of this must be done taking the 5 goals into account.
Prior Proposals to enable Indexed Updates

• In-Memory Indexed Updates (IU): Keep cache in memory and index it.
  • During Query time: Random access to find relevant cached updates
  • During Migration: Make a copy of the entire disk then make it available when migration is completed.
Prior Proposals to enable Indexed Updates

- In-Memory Indexed Updates (IU): Keep cache in memory and index it.
  - During Query time: Random access to find relevant cached updates
  - During Migration: Make a copy of the entire disk then make it available when migration is completed.

- Simply extended IU to SSDs:
  - Adds random access to SSDs
  - An entire SSD page must be read for retrieving each entry
Prior Proposals to enable Indexed Updates

• In-Memory Indexed Updates (IU): Keep cache in memory and index it.
  • During Query time: Random access to find relevant cached updates
  • During Migration: Make a copy of the entire disk then make it available when migration is completed.

• Use log-structured merge trees:
  • We reduce random reads
  • We increase writes per update
MaSM: Materialized Sort-Merge

• How is Merging Handled?
  • Sort Merge:
    • Cheaper than hash-based alternatives
    • Preserve the record order

• How is Caching Handled?
  • SSD Storage and External Sorting:
    • SSD Storage reduces memory footprint
    • External sorting is expensive!

• How is Migration Handled?
  • Full table Scan and Write Back to Disk
MaSM – 2M: Minimizing SSD writes

- **In-Memory Cache:**
  - M pages to store new updates
  - When the buffer is full, create a materialized sorted run of size M in the SSD. (Add a read-only index)

- **SSD:**
  - Capacity: $M^2$ – At most M runs

- **Query:**
  - One page per run for each run in SSD
  - M pages for Table Range Scan
MaSM – 2M: Details

• Timestamps:
  • On Every Update and Every query -> Each query only sees previous updates
  • To support in place migration, each page has the last update timestamp.

• Update Record:
  • Format: (timestamp, key, type, content)

• Online Updates & Range Scan:
  • Thanks to timestamps, only case when online updates can generate issues on scans are when
    the cache must be flushed, so mutexes are used to protect the update buffer.

• Concurrent Range Scans:
  • Supported thanks read only indexes in SSDs and timestamps on cache

• In- Place Migration:
  • Perform a full range scan, returning pages instead of records. Apply the updates to the pages
    and write them back to the disk
MaSM – M: Reducing Memory Footprint

- 2 Main Differences:
  - Better Memory Management - M pages
    - S of the M pages: Updates
    - Rest: Queries
  - Not all SSD runs have equal size:
    - The query pages can only handle M-S materialized sorted runs

The algorithm merges multiple smaller runs (1 pass) into larger runs. (2 pass runs)
MaSM – αM: Generalizing MaSM

• Recall M is number of pages allocated to MaSM
• Details:
  • Tunable Memory Usage - αM pages
    • Range $2/V \sqrt{M}$ to 2,
  • Think of previous as special cases:
    • MaSM – 2M: $\alpha = 2$ (1 SSD write/update)
    • MaSM – M : $\alpha = 1$ (1.75 SSD writes/update)
Testing
All Schemes for Handling Online Updates

- In-place Updates:
  - 1.7 – 3.7X slowdowns

- Indexed Updates:
  - 1.1 – 3.8X slowdowns

- MaSM w/ coarse-grain index:
  - incurs little overhead for 100MB to 100GB ranges
  - Bigger under 10mb

- MaSM w/ fine-grain index:
  - 4% overhead even at 4KB ranges

Synthetic Data
**TPCH – Replay Experiment**

- **In place updates:**
  - 1.6 – 2.2X worse than no updates
- **MaSM updates:**
  - Less than 1% overhead

*Figure 14: Replaying I/O traces of TPC-H queries on a real machine with online updates.*
Sustained Update Rate

• 2 Main Points:
  • Comparison:
    • MaSM schemes achieve orders of magnitude higher sustained update rates
  • Scalability:
    • Doubling the flash space will roughly double the sustained update rate
Conclusion
System Goals

1. Low Query Overhead & Small Memory Footprint:
   - SSD reads can be completely overlapped with Disk reads

2. No Random SSD writes:
   - As described in the algorithm

3. Low total SSD writes per update:
   - Between 1 and 2 writes per update!

4. Efficient in place migration:
   - Thanks to large SSD size (1-10% of disk) we have low frequency and will likely affect all pages of disk.

5. ACID:
   - Timestamps enable serializability
   - Locking is supported
   - Crash Recovery: only in-memory buffer needs recovery.
Paper Did Well:

• Analyzed business needs

• Thoroughly discussed previous attempts

• Aimed to reduce the implementation impact

• Considered alternative or additional implementations
  • And showed why they may or may not work
I Wish the Paper had:

• Considered Full usage of SSDs:
  • Lower energy consumption
  • Leveraging main data storage as a cache extension

• Considered Additional Costs of SSDs Cache:
  • Power?
  • Investment?