Efficient Transaction Processing in SAP HANA Database

Presented by Henggang Cui
15-799b Talk
Motivation

• OLTP
  – large number of concurrent users and transactions
  – high update load
  – very selective point queries

• OLAP
  – aggregation queries over a huge volume of data
  – compute statistical models from the data
Motivation

• Zoo of different systems with different capabilities for different application scenarios
  – OLTP: row-stores
  – OLAP: column-stores
• However, workloads usually contain both
  – transactional database needs statistical information to make on-the-fly business decisions
  – data-warehouses are required to capture transactions feeds for real-time analytics
SAP HANA

- efficient processing for both OLTP and OLAP
- achieved through a sophisticated multi-step record life cycle management approach
Outline

• Lifecycle management of records
• Merge details & optimization
• Summary & discussion
Lifecycle Management of Records

• Three stages of physical representation
  – L1-delta
  – L2-delta
  – Main

• Records are propagated through different stages in their lifetime
L1-delta Storage

• L1-delta
  – accepts all incoming data requests
  – stores records in row format (write-optimized)
    • fast insert and delete
    • fast field update
    • fast record projection
  – no data compression
  – holds 10,000 to 100,000 rows per single-node
L2-delta Storage

• L2-delta
  – the second stage of the record life cycle
  – stores records in column format
  – dictionary encoding for better memory usage
  – unsorted dictionary
    • requiring secondary index structures to optimally support point query access patterns
  – well suited to store up to 10 million rows
Main Storage

• Main
  – final data format
  – stores records in column format
  – highest compression rate
    • sorted dictionary
    • positions in dictionary stored in a bit-packed manner
    • the dictionary is also compressed
Lifecycle Management of Records

physical operators

unified table

L1-delta → L2-delta → main store

persistency Layer

write-optimized representation

read-optimized representation
Unified Table Access

• A common abstract interface to access different stores
• Records are propagated asynchronously
  – without interfering with running operations
• Two transformations (or merge steps)
  – L1-delta to L2-delta
  – L2-delta to main
Merge from L1-delta to L2-delta

• Row format to column format conversion
  – rows are split into corresponding columnar values
  – column-by-column inserted into the L2-delta
L1-delta to L2-delta Merge Steps

• Step 1 (parallel)
  – appends new entries to the dictionary

• Step 2 (parallel)
  – column values are added using the dictionary encodings

• Step 3
  – propagated entries removed from the L1-delta
L1-to-L2-delta Merge is Cheap

• Step 1 and Step 2 can be performed in parallel
  – # tuples to be moved is known in advance
• Needs no reconstruction of L2-delta structures
  – just appends entries to the unsorted dictionary
• This merge can be incremental

• Minimal influence to the running transactions
Merge from L2-delta to Main

• Resource intensive task
  – a new main structure is created out of the L2-delta and the existing main
  – should be carefully scheduled and highly optimized

• Must be a complete merge
  – the old L2-delta is closed and a new one is created
  – retries the merge on failure
Persistency Mapping

- HANA provides Full ACID guarantees
  - using REDO logs and save pointing
  - merging makes it quite complicated
Outline

• Lifecycle management of records
• Merge details & optimization
• Summary & discussion
Merge Optimization

• The classic merge needs optimization because
  – L2-delta to main merge is resource intensive
  – Main store needs high compression rate

• Optimization: Re-sorting merge
• Optimization: Partial merge
The Classic Merge

• Step1:
  – generate new dictionary

• Step2:
  – generate new indices based on the new dictionary
The Classic Merge

L2-delta

- Dictionary entries
- Delta index

- Positions not stored

```
Los Gatos: 1
...: 2
San Jose: 3
Campbell: 4
...: 5
Palo Alto: 6
```

```
3
6
3
1
4
6
```

main

- Dictionary entries
- Main index

```
Belmont: 1
Cupertino: 2
Daly City: 3
Los Altos: 4
Los Gatos: 5
Palo Alto: 6
Saratoga: 7
```

```
3
3
3
1
1
1
2
4
5
6
```

sorted
The Classic Merge

<table>
<thead>
<tr>
<th>Los Gatos</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>San Jose</td>
<td>3</td>
</tr>
<tr>
<td>Campbell</td>
<td>4</td>
</tr>
<tr>
<td>Palo Alto</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Belmont</th>
<th>1</th>
<th>main</th>
<th>1</th>
<th>delta</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cupertino</td>
<td>2</td>
<td>-1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daly City</td>
<td>3</td>
<td>2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Los Altos</td>
<td>4</td>
<td>3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Los Gatos</td>
<td>5</td>
<td>4</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Palo Alto</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>San Jose</td>
<td>8</td>
<td>-1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saratoga</td>
<td>9</td>
<td>7</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

sorted

dictionary position mapping

new

old
The Classic Merge

2nd phase: main index generation

new dictionary position mapping

main index

main entries

delta entries
Re-Sorting Merge

• Goal: higher compression rate

• Re-Sorting Merge
  – reorganizes the content of the full table to yield a data layout which provides higher compression potential
  – not easy because all records should have the same order in all columns
  – uses a scheme discussed in another paper
Partial Merge

• Goal: reduce merge overhead
• Partial Merge
  – splits the main into two independent structures

• Passive main
  – not part of the merge process

• Active main
  – takes part in the merge process with the L2-delta
Partial Merge

L2-delta

partial merge

main store

passive main

active main
Outline

• Lifecycle management of records
• Merge details & optimization
• Summary & discussion
Characteristics of Record Stages

(a) workload optimization

- L1-Delta
- L2-Delta
- Main

- read optimized
- write optimized
Characteristics of Record Stages

(b) memory consumption
Characteristics of Record Stages

![Graph showing characteristics of record stages with execution frequency on the y-axis and stages on the x-axis: L1-Delta, L2-Delta, L2-Delta + Active Main, Active Main, Passive Main. The graph indicates high and low execution frequencies for each stage.](image)
Discussion

• When to merge?
  – How do we know when the records are not likely to be updated anymore?

• Why it must be a complete merge?
  – Keep some in row-store, some in column-store?