ENHANCEMENT TO SQL SERVER COLUMN STORES

Presented by Rania Ibrahim
• Motivation
• SQL Server 2012 (Background)
• Enhancements to SQL Server 2012
  • Column Store Indexes Enhancements
  • Batch Mode Enhancements
• Evaluation
• Conclusion
• Discussion
AGENDA

• Motivation
  • SQL Server 2012 (Background)
  • Enhancements to SQL Server 2012
    • Column Store Indexes Enhancements
    • Batch Mode Enhancements
• Evaluation
• Conclusion
• Discussion
MOTIVATION

• Data Warehousing
  • Accesses few columns of large number of rows
  • Few number of deletions and updates

• SQL Server 2012 introduced the following:
  • Column store indexes
  • Batch processing mode

• Still it had some limitations, which are:
  • Can’t update column store indexes
  • Batch processing mode was not supported for outer join and scalar aggregation

• The previous limitation were solved in the current paper
AGENDA

• Motivation
• SQL Server 2012 (Background)
  • Enhancements to SQL Server 2012
    • Column Store Indexes Enhancements
    • Batch Mode Enhancements
• Evaluation
• Conclusion
• Discussion
BACKGROUND

• Column Index Storage

The figure is taken from the “Enhancements to SQL Server Column Stores” paper
• Caching and I/O
  • Column segments and dictionaries are brought to memory when needed
  • Objects are stored continuously in a cache for large objects
  • Blob storing the column segment can span multiple disk pages
• Batch Mode Processing
  • Very efficient on modern hardware, as it enables:
    • Memory prefetching
    • Minimize cache misses
    • TLB misses
    • Branch miss-prediction

The figure is taken from the “Enhancements to SQL Server Column Stores” paper
BACKGROUND

• Batch Mode Processing (Inner Hash Join)
  • Build Operation: Build hash table on the join attribute (threads)
  • Probe Operation: Probe the hash table (threads)

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Customer Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ahmed</td>
</tr>
<tr>
<td>2</td>
<td>Rania</td>
</tr>
<tr>
<td>3</td>
<td>Mike</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purchase ID</th>
<th>Customer ID</th>
<th>Purchase Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>10.00$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>20.00$</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>100.00$</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>5.00$</td>
</tr>
</tbody>
</table>

Select * from Customer, Purchase where Customer.CustomerID = Purchase.CustomerID
• Batch Mode Processing (Inner Hash Join)
  • Build Operation: Build hash table on the join attribute (threads)

<table>
<thead>
<tr>
<th>Hash Table Key</th>
<th>Hash Table Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash (1)</td>
<td>1 Ahmed</td>
</tr>
<tr>
<td>Hash (2)</td>
<td>2 Rania</td>
</tr>
<tr>
<td>Hash (3)</td>
<td>3 Mike</td>
</tr>
</tbody>
</table>

• The hash table **should fit in memory**, if not switch to row mode!
• Batch Mode Processing (Inner Hash Join)
  • Probe Operation: Probe the hash table (threads)

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Customer Name</th>
<th>Purchase ID</th>
<th>Purchase Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Rania</td>
<td>1</td>
<td>10.00$</td>
</tr>
<tr>
<td>2</td>
<td>Rania</td>
<td>2</td>
<td>20.00$</td>
</tr>
<tr>
<td>1</td>
<td>Ahmed</td>
<td>3</td>
<td>100.00$</td>
</tr>
<tr>
<td>3</td>
<td>Mike</td>
<td>4</td>
<td>5.00$</td>
</tr>
</tbody>
</table>

• Batch mode was not implemented for outer join and for scalar aggregation
• Motivation
• SQL Server 2012 (Background)
• Enhancements to SQL Sever 2012
  • Column Store Indexes Enhancements
    • Batch Mode Enhancements
• Evaluation
• Conclusion
• Discussion
Column Store Indexes Enhancements

• Improved Index Build

**Before**
Global dictionary is filled as the index is built

**Problem:** The most frequent values are not guaranteed to be in the dictionary

**Now**
1- Sample the data to decide either to build a global dictionary or not and what values to keep in
2- Build the index using the global dictionary
Column Store Indexes Enhancements

- Improved Index Build

Before

Estimate memory and set the number of threads to use at the start

Problem: Static degree of parallelism, usually memory estimation is inaccurate.

Now

Dynamically vary the number of threads

Continually monitor the available memory and calculate the optimal number of threads
Column Store Indexes Enhancements

• Sampling Support

Before
No sampling support for column store indexes

Now
Two types of sampling support:
1- Cluster Sampling
2- Random Sampling
Column Store Indexes Enhancements

• Bookmark Support

Before
No bookmark support for column store indexes

Now
Bookmark support using:
1. Row group ID
2. Tuple ID (sequence number within the row)
Column Store Indexes
Enhancements

- Update Handling

**Before**
No updates allowed for column store indexes

**Now**
Updates allowed for column store indexes using

1. Delete bitmap
2. Delta Stores
Column Store Indexes Enhancements

• Update Handling
  • Insertion: Insert is done in delta store
    • TupleMover: converts full delta store to columnar format
    • Bulk insertion: convert data directly to column format
  • Deletion:
    • Row is in delta store: delete it normally (B-tree supports deletion)
    • Row is in column store indexes: use delete bitmap
• Update
• Merge

The figure is taken from the “Enhancements to SQL Server Column Stores” paper
• Update Handling
  • Access method layer handle the delta stores and the deleted bitmap
  • Access method layer includes delta store content in the scans
  • Access method layer skips rows that included in deleted bitmap in the scans

The figure is taken from the “Enhancements to SQL Server Column Stores” paper
Column Store Indexes Enhancements

- Update Handling
- BAD Approach for large number of deletion ?!

WHY ?!

The figure is taken from the “Enhancements to SQL Server Column Stores” paper
Column Store Indexes Enhancements

- Update Handling
  - BAD Approach for large number of deletion ?!
- Waste of memory: Many records are stored while they should be deleted
- Overhead of looking at the delete bitmap
- Column store targets application with low update rate

The figure is taken from the “Enhancements to SQL Server Column Stores” paper
AGENDA

• Motivation
• SQL Server 2012 (Background)
• Enhancements to SQL Sever 2012
  • Column Store Indexes Enhancements
  • Batch Mode Enhancements
• Evaluation
• Conclusion
• Discussion
• Mixed Execution Mode

Before
Transition between batch and row processing mode is limited

Now
Possible to transition from batch to row mode at any point in the execution plan
Batch Mode Enhancements

• Hash Join
  • Solve the problem if the hash table doesn’t fit into memory
  • Allow spilling to disk when needed

• Build Operation
  • Divide the data into buckets where each bucket has the same hash value
Batch Mode Enhancements

- **Hash Join**
  - Solve the problem if the hash table doesn’t fit into memory
  - Allow spilling to desk when needed

- **Build Operation**
  - Divide the data into buckets where each bucket has the same hash value
  - When we need memory, choose a data bucket to spill to the desk
Batch Mode Enhancements

- **Hash Join**
  - Solve the problem if the hash table doesn’t fit into memory
  - Allow spilling to desk when needed

- **Build Operation**
  - Divide the data into buckets where each bucket has the same hash value
  - When we need memory, choose a data bucket to spill to the desk
  - Write the bucket to a temporary file that is assigned to this hash value
Batch Mode Enhancements

- Probe Operation
  - Check if the incoming row cross-ponding bucket is in memory or in desk
  - If in memory, process the row normally
  - If in desk, write the row to a temporary file
• Probe Operation

- After finishing in memory probe side, release the hash table
- Load the pairing files and repeat the process
- If more than one hash table exist, then spill from larger one!
Batch Mode Enhancements

The figure is taken from the “Enhancements to SQL Server Column Stores” paper
Bitmap Filters

- Used to reject rows on probe side ≠ rows on build side
- Two types:
  - Simple bitmap
  - Complex bitmap (Bloom Filters)
- Bloom Filters may return false positives
  - Improved its false positive rate in the enhancements
- Based on input size and other statistics, choose which bitmap filter to apply
- In SQL server 2012: No memory, No bitmap.
- Now: No memory, Yes bitmap as it has a spill option.
Archival Compression

- Data not frequently accessed can be further compressed
- That will be on the cost of slower query performance
- Xpress 8 compression library routine is used

<table>
<thead>
<tr>
<th>Database Name</th>
<th>Raw data size (GB)</th>
<th>Compression ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Archival compression?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>EDW</td>
<td>95.4</td>
<td>5.84</td>
</tr>
<tr>
<td>Sim</td>
<td>41.3</td>
<td>2.2</td>
</tr>
<tr>
<td>Telco</td>
<td>47.1</td>
<td>3.0</td>
</tr>
<tr>
<td>SQM</td>
<td>1.3</td>
<td>5.41</td>
</tr>
<tr>
<td>MS Sales</td>
<td>14.7</td>
<td>6.92</td>
</tr>
<tr>
<td>Hospitality</td>
<td>1.0</td>
<td>23.8</td>
</tr>
</tbody>
</table>

The figure is taken from the “Enhancements to SQL Server Column Stores” paper
AGENDA

• Motivation
• SQL Server 2012 (Background)
• Enhancements to SQL Server 2012
  • Column Store Indexes Enhancements
  • Batch Mode Enhancements
• Evaluation
  • Conclusion
• Discussion
The figure is taken from the “Enhancements to SQL Server Column Stores” paper

<table>
<thead>
<tr>
<th>Query</th>
<th>Rowstore</th>
<th>Columnstore</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cold</td>
<td>Warm</td>
<td>Cold</td>
</tr>
<tr>
<td>Q_count</td>
<td>13.0</td>
<td>4.33</td>
<td>0.309</td>
</tr>
<tr>
<td>Q_outer</td>
<td>263</td>
<td>1.03</td>
<td>4.1</td>
</tr>
<tr>
<td>Q_union_all</td>
<td>20.8</td>
<td>19.0</td>
<td>3.0</td>
</tr>
<tr>
<td>Q_count_in</td>
<td>62.5</td>
<td>24.0</td>
<td>2.29</td>
</tr>
<tr>
<td>Q_not_in</td>
<td>12.0</td>
<td>10.2</td>
<td>6.95</td>
</tr>
</tbody>
</table>
Evaluation

• Storage Requirement
  • Column store store_sales table = 13.2 GB
  • Row store store_sales table = 43.5 GB

• Delete Performance: Delete 5.5% of 101 million row table
  • Column store took 57 seconds (insert in delete bitmap)
  • Row store took 239 seconds

• Bulk and Trickle Load Rates
  • Trickle insert: 3.93 million rows, 22 minutes and 16 seconds
  • Bulk load of 20 million rows in batches of 1000 rows, 9 minutes and 46 seconds (faster)
AGENDA

• Motivation
• SQL Server 2012 (Background)
• Enhancements to SQL Server 2012
  • Column Store Indexes Enhancements
  • Batch Mode Enhancements
• Evaluation
• Conclusion
• Discussion
Conclusion

- New enhancement to SQL sever 2012 were introduced
  - Making column store indexes updatable
  - Enhancing column store index building
  - Extending batch mode processing to outer join and scalar aggregation
  - Enhancing batch mode for hash join
  - Introducing archival compression
  - …..

- Evaluate the efficiency of the newly introduced extensions
AGENDA

• Motivation
• SQL Server 2012 (Background)
• Enhancements to SQL Server 2012
  • Column Store Indexes Enhancements
  • Batch Mode Enhancements
• Evaluation
• Conclusion
• Discussion
Discussion

Any questions 😊

• If more than one hash table exist in memory and memory is filled, then the system spills from larger one! As that will minimize the expected spill on the probe size, is that the best option?

• Random Sampling, is it the best ? It is known that random sample is not good for skewed data distributions ? Use stratified sampling ?

• How to extend the model to work with high deletion\update operations or long time with small rate of deletion\update ?
  • Garbage collection process, operate when load is low, that read the compressed row group, delete the rows that in the deleted bitmap and re-write ?
  • Other ideas ?

• Deciding when to use archival compression ? Is there an automatic way to decide ?