How to reduce memory grant observed during ALTER INDEX REORGANIZE on Clustered Columnstore Indexes

Last updated by | Radhika Shah | Sep 6, 2022 at 8:39 PM PDT

Contents

Scenario

Customer observes very high memory grant request while trying to do the ALTER INDEX REORGANIZE for various Clustered Columnstore Indexes. Customer is using below to reorganize their indexes after large insert operation on their managed instance and observing huge memory grant wait:

```
ALTER INDEX <indexname> ON <tablename> REORGANIZE WITH (COMPRESS_ALL_ROW_GROUPS = ON);
```

Analysis / Recommendations

Columnstore index, both clustered and nonclustered, can get fragmented like any other index. SQL Server allows customer to defragment this index using the familiar ALTER INDEX <index-name> REORGANIZE command instead of using a heavy hammer approach of rebuilding the index.

A columnstore index is considered fragmented if it has:

- Multiple delta rowgroups: The delta rowgroups are not necessarily a negative as they can be used to minimize the impact on transactional workload in real-time operational analytics. If your goal is to speed up the analytics queries to what is possible, then you can get the best performance when all the rows are in the compressed rowgroups. Besides speeding up the queries, compressed rowgroups typically take 10x less storage than delta rowgroup and proportionally less memory when they are scanned.
- Deleted rows: When a rowgroup is compressed, it is marked read-only. Unlike when the rows are stored in rowstore in pages, there is no easy way to remove compressed row. I am sure you don't want uncompress a rowgroup, delete the row and then re-compress it as this will be prohibitively expensive. What SQL Server does instead it store the information to identify the row in an internal btree structure referred to as delete-bitmap and delete-buffer. While this is more efficient for the delete operation, all analytic queries must filter the deleted rows out by internally executing 'anti-semijoin' before the query results can be returned. Ideally, you don't want any deleted rows.

- Ideal compressed rowgroup has a size of 1 million rows (1024*1024 = 1048576) and most compressed rowgroups are of this size but they can be smaller for multiple reasons. For example, if you are bulk importing data and your batchsize is set to 102400, it will generate compress rowgroup of size 102400. You could choose a batchsize in the multiples of 1 million but more often than not, you have little control on how the source files are generated. There are also situations such as hitting the dictionary size limit where you can't really much to change it. Please refer to this blog for some examples.

Delta rowgroup: *A delta rowgroup is a clustered B-tree index that's used only with columnstore indexes. It improves columnstore compression and performance by storing rows until the number of rows reaches a threshold (1,048,576 rows) and are then moved into the columnstore.*

When a delta rowgroup reaches the maximum number of rows, it transitions from an OPEN to CLOSED state. A background process named the tuple-mover checks for closed row groups. If the process finds a closed rowgroup, it compresses the delta rowgroup and stores it into the columnstore as a COMPRESSED rowgroup.

When a delta rowgroup has been compressed, the existing delta rowgroup transitions into TOMBSTONE state to be removed later by the tuple-mover when there is no reference to it. The tuple-mover is helped by a background merge task that automatically compresses smaller OPEN delta rowgroups that have existed for some time as determined by an internal threshold, or merges COMPRESSED rowgroups from where a large number of rows has been deleted. This improves the columnstore index quality over time.

Deltastore: *A columnstore index can have more than one delta rowgroup. All of the delta rowgroups are collectively called the deltastore.*

During a large bulk load, most of the rows go directly to the columnstore without passing through the deltastore. Some rows at the end of the bulk load might be too few in number to meet the minimum size of a rowgroup, which is 102,400 rows. As a result, the final rows go to the deltastore instead of the columnstore. For small bulk loads with less than 102,400 rows, all of the rows go directly to the deltastore.

Based on the scenario mentioned above, customer is performing REORGANIZE index after large dataset is inserted. In general, the command `REORGANIZE WITH (COMPRESS _ALL ROW _GROUPS = ON)` is useful especially after performing large/bulk insert operations.

When using columnstore indexes, the delta store may end up with multiple small row groups after inserting, updating, and deleting data over time. Reorganizing a columnstore index forces delta store row groups into compressed row groups in columnstore, and combines smaller compressed row groups into larger row groups. The reorganize operation also physically removes rows that have been marked as deleted in the columnstore. Reorganizing a columnstore index may require additional CPU resources to compress data, which may slow the overall system performance while the operation is running. However, once data is compressed, query performance improves. For syntax examples, see Examples - Columnstore reorganize.

To use less memory during index `REORGANIZE`, customer need to reduce the max memory grant in the workload group or the resource pool that they use. See ALTER WORKLOAD GROUP and ALTER RESOURCE POOL.

Public Doc Reference

Columnstore indexes: Overview

Columnstore indexes - Query performance recommendations

Perform index maintenance using REORGANIZE and REBUILD

Reorganize an index

Examples - Columnstore reorganize

Columnstore Index Defragmentation using REORGANIZE Command

Understanding SQL server memory grant

Columnstore Memory Grant Issue

Memory Grants: The mysterious SQL Server memory consumer with Many Names

Internal Reference

ICM 272868757

How good have you found this content?