Lockless in Seattle – Using In-Memory OLTP for Transaction Processing – Mark Broadbent - Supporting Notes

Contents

Slide Notes

Slide – In Memory OLTP to the rescue

In-memory data structures are optimised for memory so that the row format is has no verbose or unnecessary fields and the row header for instance will not take up any more space than is necessary – for instance if there less than 8 indexes then the row header will use less than 8 fields.

Data is persisted to disk through highly efficient asynchronous streaming to disk structures known as checkpoint file pairs, and committed transactions are written to the log file

Since transactions are still committed to disk, in order to avoid log bottlenecks, various performance improvements have been made to the log performance.

Unlike On-disk Optimistic concurrency (under SNAPSHOT ISOLATION and READ COMMITTED SNAPSHOT ISOLATION), there is no tempdb overhead from the row versioning. All versioning is performed in memory.

With On-disk tables, a lock enforces logical consistency and a latch enforces physical consistency these can be detected through high pagelatch waits in **sys.dm_os_latch_stats**

There are no in-memory fragmentation concerns for the table row data. One memory address is as quick to address as another.

The existing tooling can be used to manage In-Memory OLTP and it is built into the SQL Server product.

Slide – In-Memory Data Structures (Tables and Indexes)

All data for In-Memory OLTP tables and indexes are fully housed in memory.
When tables created a C file first generated, it is compiled into a DLL and then linked in process to the SQL Server process.

All row data is fully optimized to have a low memory overhead and low access time.

These DLLs are recreated, compiled and linked on database startup time.

Indexes are not persisted and instead are rebuilt at startup time.

There are two types of indexes that we can define.
Hash Indexes are used for point equality lookups and Non-Clustered indexes are used in the same way that we would use on-disk non-clustered indexes -for range and ordered scans.

Row data is not duplicated instead there are just pointers to the rows.

Index statistics are not automatically updated so when they are created, they are based on a table with zero rows.

Binary DLLs are not persisted in a dbbackup or reboot and table DLLs are recreated on start-up and native stored procedures area recompiled first execution.
*.c files are the C files
*.mat.xml files are XML representations
of Mixed Abstract Tree structure (related to query plan)
*.obj files are object files created by compiler
*.pub are compiler symbol files

Slide - In-Memory Data Structures (Natively Compiled Stored Procedures)

When a Natively Compiled Stored procedure is created, the query is first optimized and a C file is generated and compiled into a DLL and dynamically Linked into the SQL process.

Optimization is based on the index statistics -which potentially were created on a table with zero rows! Could be a problem there?!

These stored procedures are themselves recreated upon restart, but compilation is deferred until its first execution.

Native compiled stored procedures are unable to access on-disk tables.

Native Compilation Stored Procs can only access memory optimized tables

Native compilation stored procedures do not support EXECUTE AS CALLER in order to avoid overhead of per-statement permission checks.

Slide – MemOpt: Row Format

SQL instance maintains a Global Transaction Timestamp which is auto-incremented when a transaction commits and is unique for each and every transaction.

Begin Timestamp is always added at time of creation and is set from the Global Transaction Timestamp.

If a row is deleted then the EndTs is set. Otherwise if the row is current, a special Infinity value is used.

Every statement in a transaction has a unique 4 byte statementid. This is updated in the StmtId field of row header to implement Halloween protection.

IdxLinkCount shows number of index pointers referencing row and is useful for garbage collection deallocation.

Payload data is never updated. IMOLTP delete row by setting EndTs of original and inserts new data row with new BeginTs and EndTs set to infinity.

Row size limitation of 8060 bytes applies to size of column in table definition regardless of the actual row size (due to variable types).

8 * 8-byte index pointers at end of row header used to link rows together (maximum of 8 indexes).

Slide - Hash Indexes

Bucket size can reduce collisions but does not guarantee avoidance of collisions.

Rows In-Memory are NEVER updated. Updates generate a new row and the Global Transaction Timestamp is set on EndTs to mark deletion of old row. BeginTs and End Ts determine a row's visibility to a transaction based on it's start time.

Hashing reduces index key size giving performance gains and storage benefits. Function is deterministic but does not guarantee uniqueness nor sequential order. Hashing to same value is known as a hash collision.

Overestimate potential bucket count rather than underestimate. Extra overhead is less of a problem than too much

Underestimation of buckets can degrade performance through collisions and overestimation of buckets can waste system memory. Each bucket requires 8 bytes of memory allocated. Index scans will also incur unnecessary extra buckets to traverse.

For CRUD operations, the entire row chain must be read on read and updates, so collisions can cause overhead there too.

MS recommendation is bucket_count of one or two times distinct values in the index column/s. See https://msdn.microsoft.com/en-us/library/dn494956.aspx

Use **sys.dm_db_xtp_hash_index_stats**

Hash indexes are great for equality point lookups but have no SARGability for leading column searches or even range searches -since all columns are hashed as one.

SQL Server creates index and column level statistics for IMOLTP tables but does not update automatically. Therefore, these would be generally based upon an empty table!

Slide – BW - Trees

Non-clustered indexes in IMOLTP originally called range indexes.

There is no such thing as a clustered index with IMOLTP since hash indexes are also not clustered. There is no requirement either since the physical ordering of the rows in memory has no value -that is way the index pointers are enough to impose the logical ordering.

When SQL Server consolidates delta records, the newly created page has the same PID and replaces the old page. Consolidation happens on change, when a delta page already contains 16 records.

Minimal index_id value of IMOLTP indexes is 2 which is the same as on-disk indexes.

Every index is covering since the leaf level is always the data row -therefore you will not experience key or RID lookups

Non-clustered index access, BW-Tree is traversed to find the lead pages. Overhead depends on index depth and number of delta records.

Nonclustered indexes are great for range scans and seek operations on multiple predicates. SARGability rules for in-memory and on-disk nonclustered indexes are almost identical. Biggest difference is that in-memory indexes are uni-directional and are therefore incapable of providing SQL Server the ability to scan the index keys in the opposite sort order.

Page splitting operations can still occur (when the page does not have enough PFS to accommodate a new data row. Page merging occurs when delete operation leaves an index page less than 10% from its maximum page size (8KB) or contains a single row.

Update the stats after data is loaded and if data in memory is volatile, you should manually update statistics on a regular basis.

Use UPDATE STATISTICS command to update individual statistics or sp_updatestats to update all statistics on tables.

A full scan is always performed when updating stats (on-disk samples by default). Must specify NORECOMPUTE when CREATE STATISTICS or UPDATE STATISTICS.

Missing stats can result in suboptimal execution plans.

More on In-Memory OLTP statistics at http://msdn.microsoft.com/en-us/library/dn232522.aspx.

Deallocation of rows will not happen instantly by the garbage collector (even if you dropped a table), so ensure that your system has at least enough memory to accommodate two extra copies of table data if possible.

Slide - Data Persistence

Persistence to disk is achieved via streaming utilization customized FILESTREAM technology. In-Memory OLTP and FILESTREAM store data completely separately from each other and would require seperate filegroups.

On disk tables store most recent version of the row, IMOLTP persists multiple versions of the row on disk (as and when these changes are generated) into the data and delta files which make up a Checkpoint File Pair (CFP). Changes to rows may be store in any of the containers.

Inserts stored in data file, deletes stored in delta file/s, updates create new record for data and delta file.

Every database has at least 8 Checkpoint File Pairs in various states.

Since no data is overwritten on disk, and all writes to data and delta files are append-only and therefore sequential, this avoids random IO}

Make sure to house containers on drives optimized for sequential IO.

Log writes are still the most important thing but log truncation will still depend upon being able to persist that data.

A separate Checkpoint file per is created per scheduler starting at minimum of eight. Initial size of files depends upon server memory.

Checkpoints reduce recovery time

IMOLTP performs continuous asynchronous checkpoints and continuously scans transaction log, streaming changes to checkpoint file pairs in UNDER CONSTRUCTION state.

Sequential IO is still significantly faster than random IO on SSD -some metrics suggest 5 times slower.

Even though there is are continuous checkpoints running in IMOLTP, there is still a checkpoint event which hardens all remaining log records to checkpoint file pairs and transitions all UNDER CONSTRUCTION CFP to the ACTIVE state. It creates a checkpoint inventory of all the files added by current checkpoint.

Active CFPs become read-only although delta files can store outstanding Deletes (and deletes from updates).

IMOLTP checkpoint can be invoked manually with CHECKPOINT command or when a transaction log usage has grown by more than 512MB regardless of whether the transaction log data is IMOLTP related or not. In the latter case, this could return a sub-optimal performance if high volume of on-disk logging is occurring.

Recovery of database will start from last CHECKPOINT event (using those ACTIVE CFPs) plus any addition transaction in transaction log.

Disk placement of In-Memory filegroups on different storage arrays can help improve I/O and performance but they should be optimized for sequential IO.

Slide – Querying Data

Native has significant performance gain but limited surface area support compared to Interop

SNAPSHOT isolation is on transaction not statement.

Cross container transactions can have

READ UNCOMMITTED/READ COMMITTED/READ_COMMITTED_SNAPSHOT + in-memory SNAPSHOT/ REPEATABLE READ/ SERIALIZABLE

REPEATABLE READ/ SERIALIZABLE + in-memory SNAPSHOT

Cross container transactions therefore are really two internal transactions that are synchronized together.

Note you cannot use SNAPSHOT on disk with in-memory since there are synchronization issues

An exception is raised and transaction is rolled back if REPEATABLE READ or SERIALIZABLE data consistency rules are violated.

Auto committed IMOLTP READ COMMITTED transaction would not see changes committed after a transaction has started (unlike disk-table READ COMMITTED). {must try and perhaps put into my isolation demo}

NOLOCK hint ignored on IMOLTP tables, READ UNCOMMITTED unsupported and errors.

Slide - In-Memory OLTP Limitations

N*BIN2 collation requirement (in 2014) for Index columns could be deal breaker for existing migration. {Also how does this effect cross container transactions? Would you need explicit collate clause}

Slide - Troubleshooting and Reporting on In-Memory OLTP

Use **sys.dm_db_xtp_index_stats_view** to show information on hash and non-clustered index access methods and ghost rows.
scans_started - shows frequency that a row chains in index were scanned.
rows_returned - shows cumulative number of rows returned to a next plan operator.
rows_touched - shows cumulative number of rows accessed in the index.
rows_expired - shows detected stale rows.
rows_expired_removed - shows unlinked (from index row chain) stale rows.
{perhaps need a few DMVs on a slide with important columns?}

More on sys.dm_db_xtp_index_stats view at http://msdn.microsoft.com/en-us/library/dn133081.aspx.

The **sys.dm_db_xtp_nonclustered_index_stats** view nonclustered index stats.

more on **sys.dm_db_xtp_nonclustered_index_stats** view at https://msdn.microsoft.com/en-us/library/dn645468.aspx.

Other

IMOLTP not supported for database mirroring sessions but is supported in Availability Groups (only durable tables) and Failover Clusters -the latter of which it could affect failover time. Cannot replicate IMOLTP tables in a publication, but In-Memory enabled databases can still be replicated.

DBCC CHECKDB skips IMOLTP tables.

IMOLTP requires 64 bit SQL Server only

Notice of copyright