Hekaton: SQL Server’s Memory-Optimized OLTP Engine

Cristian Diaconu, Craig Freedman, Erik Ismert, Per-Åke Larson, Pravin Mittal, Ryan Stonecipher, Nitin Verma, Mike Zwilling
Microsoft
{cdiaconu, craigfr, eriki, palarson, pravinm, ryanston, nitinver, mikezw}@microsoft.com

Presented by: Prateek Gulati
Agenda

- Why do you need in-memory processing?
- Hekaton engine overview
- How it is done
- Benefits
- Limitations
Industry Trends: CPU

- Computing power holds Moore Law due to parallelism
- CPU clock frequency stalled
- Parallel processing has its limits due to lock contention
Industry Trends: RAM

- RAM prices continue to fall
- Servers have HUGE memory
- DDR4 expected to hit mainstream in 2014-2015
- Traditional page based architecture has limitations, even when all pages are in memory
Hekaton-In-memory OLTP engine Architecture

**Architectural Pillars**

<table>
<thead>
<tr>
<th>Main-Memory Optimized</th>
<th>T-SQL Compiled to Machine Code</th>
<th>High Concurrency</th>
<th>SQL Server Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Optimized for in-memory data</td>
<td>• T-SQL compiled to machine code via C code generator and VC</td>
<td>• Multi-version optimistic concurrency control (MVCC) with full ACID support</td>
<td>• Same manageability, administration &amp; development experience</td>
</tr>
<tr>
<td>• Memory optimized Indexes (hash and range) exist only in memory</td>
<td>• Invoking a procedure is just a DLL entry-point</td>
<td>• Core engine uses non blocking lock-free algorithms</td>
<td>• Integrated queries &amp; transactions</td>
</tr>
<tr>
<td>• No buffer pool, B-trees</td>
<td>• Memory optimized Indexes (hash and range) exist only in memory</td>
<td>• No lock manager, latches or spinlocks</td>
<td>• Integrated backup/restore</td>
</tr>
<tr>
<td>• Stream-based storage</td>
<td>• Transaction log optimization (block writes, no undo)</td>
<td>• No TempDB</td>
<td>• If SQL Server crashes data is fully recoverable.</td>
</tr>
</tbody>
</table>

- T-SQL Compiled to Machine Code
  - T-SQL compiled to machine code via C code generator and VC
  - Invoking a procedure is just a DLL entry-point
  - Aggressive optimizations at compile-time

- High Concurrency
  - Multi-version optimistic concurrency control (MVCC) with full ACID support
  - Core engine uses non blocking lock-free algorithms
  - No lock manager, latches or spinlocks
  - No TempDB

- SQL Server Integration
  - Same manageability, administration & development experience
  - Integrated queries & transactions
  - Integrated backup/restore
  - If SQL Server crashes data is fully recoverable.
Hekaton Integration with SQL Server
Native Compilation Process

Compile T-SQL statements and table data access logic into machine code.
## Native Compiled Stored Procedures

<table>
<thead>
<tr>
<th>Interpreted T-SQL Access</th>
<th>Native Compiled Proc</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Access both memory- and disk-based tables</td>
<td>• Access only memory optimized tables</td>
<td></td>
</tr>
<tr>
<td>• Less performant</td>
<td>• Maximum performance</td>
<td></td>
</tr>
<tr>
<td>• Virtually all T-SQL functions supported</td>
<td>• Limited T-SQL functions supported</td>
<td></td>
</tr>
<tr>
<td><strong>When to use</strong></td>
<td><strong>When to use</strong></td>
<td></td>
</tr>
<tr>
<td>• Ad hoc queries</td>
<td>• OLTP-style operations</td>
<td></td>
</tr>
<tr>
<td>• Reporting-style queries</td>
<td>• Optimize performance critical business logic</td>
<td></td>
</tr>
<tr>
<td>• Speeding up app migration</td>
<td>• More the logic embedded, better the performance improvement</td>
<td></td>
</tr>
</tbody>
</table>
In-Memory OLTP Structures summary

Rows
• Row structure is optimized for memory access
• There are no Pages
• Rows are versioned and there are no in-place updates
• Fully durable by default (but they don’t have to be)

Indexes
• There is no clustered index, only non-clustered indexes
• Indexes point to rows, access to rows is via an index
• Indexes do not exist on disk, only in memory, recreated during recovery
• Hash indexes for point lookups
• Range indexes for ordered scans and Range Scans
In-Memory Row Format

- Begin/End timestamp determines row’s version validity and visibility
- No concept of data pages, only rows exist
- Row size limited to 8060 bytes (@table create time) to allow data to be moved to disk-based tables
- Not every SQL table schema is supported (Ex: LOB and SqlVariant)
Hash Indexes

Hash index on Name

Ordered index on City

B-tree

10 20 John London 100

15 inf Jane Paris 150

20 Tx75 John London 110

Tx75 Inf John London 130

30 Tx75 Larry Rome 170

Tx75 inf Larry Rome 150

J

L

Old

New
Non Clustered (Range) Index

- No latch for page updates
- No in-place updates on index pages
- Page size - up to 8K. Sized to the row
- Sibling pages linked one direction
- No covering columns (only the key is stored)
Memory Optimized Table Insert

T100: INSERT (John, Prague)
Memory Optimized Table Update

T200: UPDATE (John, Prague) to (John, Beijing)
## Memory Optimized Table Delete

<table>
<thead>
<tr>
<th>Timestamps</th>
<th>Chain ptrs</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>50, ∞</td>
<td>Jane</td>
<td>Prague</td>
<td></td>
</tr>
<tr>
<td>100, ∞</td>
<td>John</td>
<td>Prague</td>
<td></td>
</tr>
<tr>
<td>90, 150</td>
<td>Susan</td>
<td>Bogota</td>
<td></td>
</tr>
</tbody>
</table>

T150: DELETE (Susan, Bogota)
Transaction Durability

• Transaction durability is ensured to allows system to recover memory-optimized table after a failure.
• Log streams contain the effects of committed transactions logged as insertion and deletion of row versions
• Checkpoint streams come in two forms:
  • a) data streams which contain all inserted versions during a timestamp interval,
  • b) delta streams, each of which is associated with a particular data stream and contains a dense list of integers identifying deleted versions for its corresponding data stream
• Hekaton table can be durable or non-durable
• Stored in a single memory-optimized FILEGROUP based on FILESTREAM implementation
• Sequential IO pattern (no random IO)
Transaction Logging

- Uses database’s transaction log to store content

- Each Hekaton log record contains a transaction log record header, followed by Hekaton-specific log content

- All logging in Hekaton is logical
  - No physical log records for physical structure modifications
  - No index-specific / index-maintenance log records
  - Redo-only log records in transaction log
Checkpoints

Hekaton Checkpoint
• Not tied to recovery interval or SQL checkpoint. Has its own log truncation
• Gets triggered when generated log exceeds a threshold (1GB) or internal min time-threshold has crossed since last checkpoint or manual checkpoint
• Checkpoint is a “set of {Data, Delta} files and checkpoint file inventory to apply transaction log from”
Populating Data / Delta files

Data files:
- Pre-allocated size (128 MB)
- Hekaton Engine switches to new data file when it estimates that current set of log records will fill the file
- Stores only the inserted rows
- Indexes exist only in memory, not on disk
- Once a data file is closed, it becomes read-only

Delta files:
- File size is not constant, write 4KB pages over time
- Stores IDs of deleted rows
Data / Delta Files

Data file contains rows inserted within a given transaction range

<table>
<thead>
<tr>
<th>Transaction Timestamp</th>
<th>TableID</th>
<th>RowID</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp (INSERT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timestamp (INSERT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timestamp (INSERT)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Delta file contains deleted rows within a given transaction range

<table>
<thead>
<tr>
<th>Transaction Timestamp</th>
<th>TableID</th>
<th>RowID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp (INSERT)</td>
<td>Timestamp (DELETE)</td>
<td>RowID</td>
</tr>
<tr>
<td>Timestamp (INSERT)</td>
<td>Timestamp (DELETE)</td>
<td>RowID</td>
</tr>
<tr>
<td>Timestamp (INSERT)</td>
<td>Timestamp (DELETE)</td>
<td>RowID</td>
</tr>
</tbody>
</table>
Merge Operation

• Merges 2+ adjacent data / delta files pairs into 1 pair
• Need for merge - Deleting rows causes data files to have stale rows
• Manual checkpoints closes file before it is “full”
• Reduces storage required to “store” active data rows
• Improves the recovery time
• Stored Procedure provided to invoke merge manually
Merge Operation

Timestamp: 500

Timestamp: 600

Memory Optimized Table Filegroup

Range: 100 - 199
Range: 200 - 299
Range: 300 - 399
Range: 400 - 499

Merge 200-399
### Garbage Collection

<table>
<thead>
<tr>
<th>Timestamps</th>
<th>Chain ptrs</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>50, ∞</td>
<td>Jane</td>
<td>Prague</td>
<td></td>
</tr>
<tr>
<td>100, 200</td>
<td>John</td>
<td>Prague</td>
<td></td>
</tr>
<tr>
<td>200, ∞</td>
<td>John</td>
<td>Beijing</td>
<td></td>
</tr>
<tr>
<td>90, 150</td>
<td>Susan</td>
<td>Bogota</td>
<td></td>
</tr>
</tbody>
</table>

**T250:** Garbage collection
Cooperative Garbage Collection

- Scanners can remove expired rows when they come across them
- Offloads work from GC thread
- Ensures that frequently visited areas of the index are clean of expired rows
Performance Gains

- No improvements in communication stack, parameter passing, result set generation
- 10-30x more efficient
- Reduced log bandwidth & contention. Log latency remains
- Checkpoints are background sequential IO
Hekaton Engine’s Scalability

System throughput

<table>
<thead>
<tr>
<th>Transactions Per Second</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cores</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>SQL with contention</td>
<td>984</td>
<td>1,363</td>
<td>1,645</td>
<td>1,876</td>
<td>2,118</td>
<td>2,312</td>
</tr>
<tr>
<td>SQL without contention</td>
<td>1,153</td>
<td>2,157</td>
<td>3,161</td>
<td>4,211</td>
<td>5,093</td>
<td>5,834</td>
</tr>
<tr>
<td>Interop</td>
<td>1,518</td>
<td>2,936</td>
<td>4,273</td>
<td>5,459</td>
<td>6,701</td>
<td>7,709</td>
</tr>
<tr>
<td>Native</td>
<td>7,078</td>
<td>13,892</td>
<td>20,919</td>
<td>26,721</td>
<td>32,507</td>
<td>36,375</td>
</tr>
</tbody>
</table>
Memory Optimized Table Limitations

Optimized for high-throughput OLTP
• No XML and no CLR data types

Optimized for in-memory
• Rows are at most 8060 bytes
• No Large Object (LOB) types like varchar(max)
• Durable memory-optimized tables are limited to 512 GB. (Non-durable tables have no size limit.)

Scoping limitations
• No FOREIGN KEY and no CHECK constraints
• No schema changes (ALTER TABLE) – need to drop/recreate table
• No add/remove index – need to drop/recreate table
• No Computed Columns
• No Cross-Database Queries
Natively Compiled Procedures Restrictions

- Not all operators/TSQLs are supported
- Only Nested Loop join, no TSQL MERGE or EXISTS, cursors, nested queries
- No CASE statement, CTEs, user-defined functions, UNION statement, DISTINCT statement
- Transaction isolation level
- SNAPSHOT, REPEATABLEREAD, and SERIALIZABLE
- READ COMMITTED and READ UNCOMMITTED is not supported
- Cannot access disk-based tables
- No TEMPDB! Use In-Memory Table variables
- No automatic recompile on statistics changes
- Need to stop & start SQL or drop & create procedure
CREATE DATABASE [Hekaton]
ON PRIMARY
(NAME = N'Hekaton_data', FILENAME = N'C:\Data\Data\Hekaton_data.mdf'),
FILEGROUP [Hekaton_InMemory] CONTAINS MEMORY_OPTIMIZED_DATA
(NAME = N'Hekaton_mem', FILENAME = N'C:\Data\Mem\Hekaton_Lun1.mdf')
LOG ON
(NAME = N'Hekaton_log', FILENAME = N'C:\Data\Log\Hekaton_log.ldf')

ALTER DATABASE [Hekaton]
ADD FILE (NAME = N'Hekaton_mem', FILENAME = N'C:\Data\Mem\Hekaton_Lun2.mdf')
TO FILEGROUP [Hekaton_InMemory]
CREATE TABLE [Customer] (  [CustomerID] INT NOT NULL  PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 1000000),  [AddressID] INT NOT NULL INDEX [IxName] HASH WITH (BUCKET_COUNT = 1000000),  [LName] NVARCHAR(250) COLLATE Latin1_General_100_BIN2 NOT NULL INDEX [IXLName] NONCLUSTERED (LName)  )  WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA);
References

http://www.enterprisetech.com/2014/03/18/microsoft-turbocharges-transactions-hekaton-memory/
http://www.sqlskills.com/blogs/bobb/category/hekaton/
http://www.sqlpassion.at/archive/2013/08/12/extreme-transaction-processing-xtp-hekaton-the-solution-to-everything/
www.slideshare.net/RaviOkade/
Thank you for your time!

Q&A