STORAGE INDUSTRY
SUMMIT

Convergence of
Storage and Memory
Developing the Needed
Ecosystem

JANUARY 20, 2016, SAN JOSE, CA

Cristian Diaconu

Microsoft

Microsoft SQL Hekaton – Towards Large Scale Use of PM for In-memory Databases

SNIA SSSI | SOLID STATE STORAGE

What this talk is about

- Trying to answer the question: What does it take for an in-memory database to out-compete a disk based database on all storage-related metrics?
- Our experience with Persistent Main Memory for In-Memory Databases

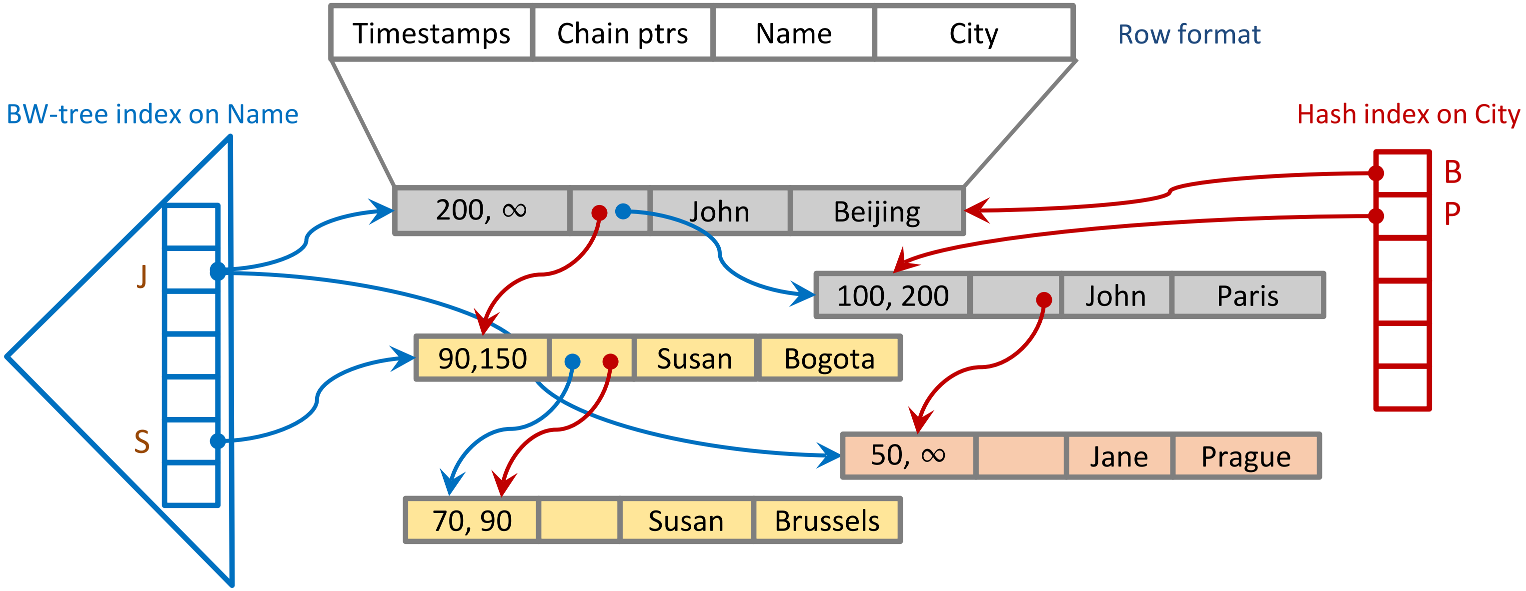
- **Talk Outline:**
- Definitions and terms
- Transaction log in persistent main memory
- Impact on database high availability
- Checkpoint, recovery and very large databases

In-memory databases

- Database engines that take advantage of large memory
- Store all/most of the data in DRAM – basis for perf gains

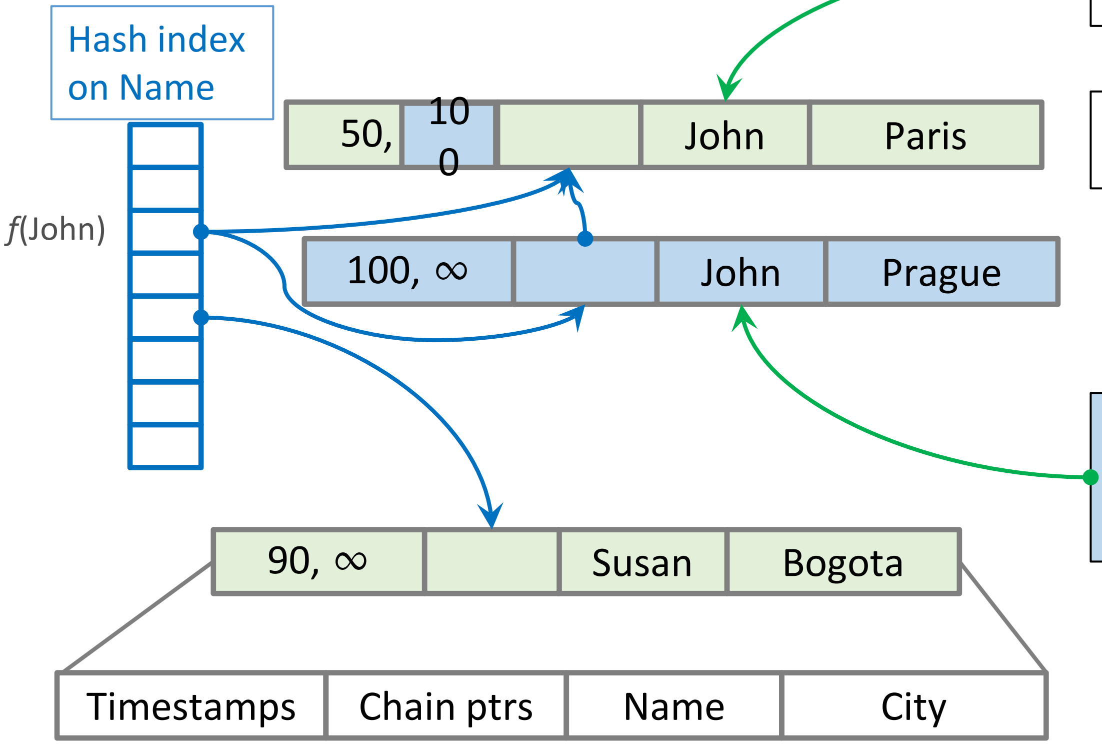
- In-memory data representation != on disk representation
- Fully integrated with Sql Server (transactions, **logging**)
- No buffer pool, no dirty writes, no locks/latches/blocking
- Does not compromise on ACID properties:
  - Atomicity
  - Consistency
  - Isolation
  - **Durability (logging and checkpoint)**
- **Up to 30X performance gains on important workloads**

Memory-Optimized Tables



- Row can be part of multiple indexes, but there is only a single copy
- Each row version has a valid time range indicated by two timestamps
- A version is visible if transaction read time falls within the version's valid time
- Garbage collection of versions: incremental, parallel, non-blocking, cooperative

Direct Access, Multi-Version, Lock-Free Transactions



Transaction **99**: Running compiled query
SELECT City WHERE Name = ‘John’
Simple hash lookup returns direct pointer to ‘John’ row

Background operation will unlink and deallocate the old ‘John’ row after transaction 99 completes.

Transaction **100**:
UPDATE City = ‘Prague’ where Name = ‘John’
No locks of any kind, no interference with transaction 99

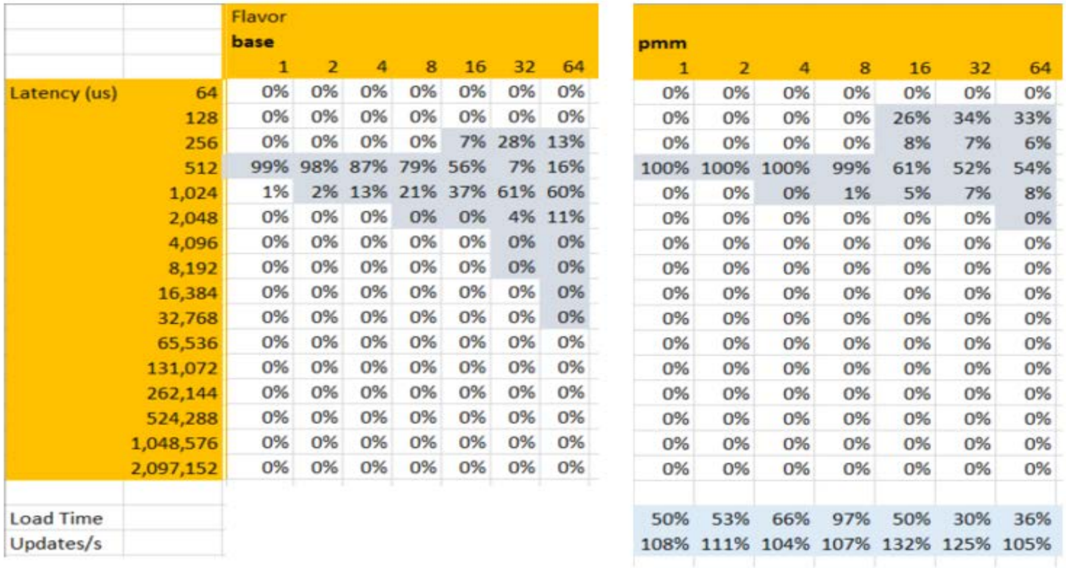
What this talk is about

- Trying to answer the question: What does it take for an in-memory database to out-compete a disk based database on all storage-related metrics?
- Experience with Persistent Main Memory for In-Memory Databases

- **Talk Outline:**
- ~~Definitions and terms~~
- Transaction log in persistent main memory
- Impact on database high availability
- Checkpoint, recovery and very large databases

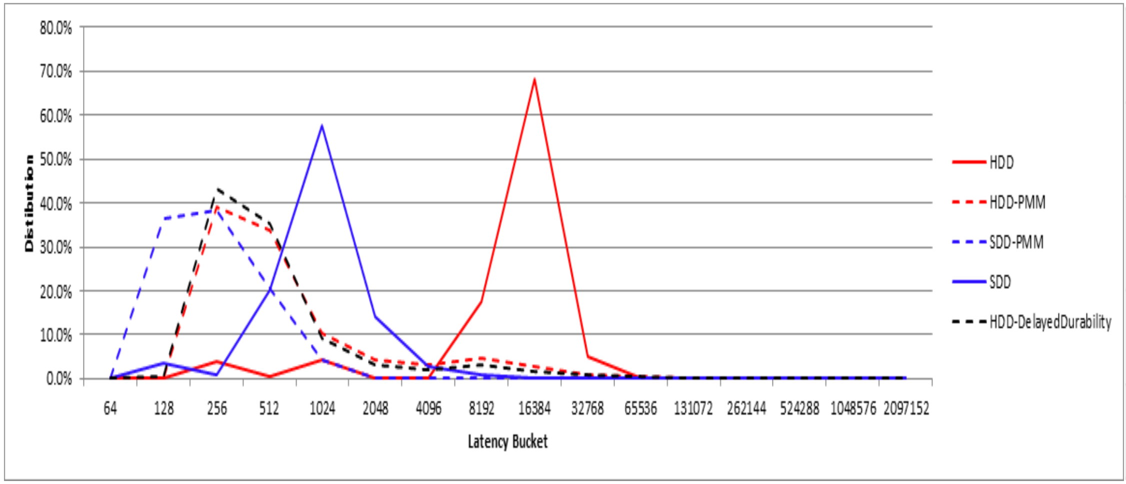
PMM Log

- Use byte-addressable log implementation
- Using Windows DirectAccess filesystem capability
- Compared with the ideal block mode access device

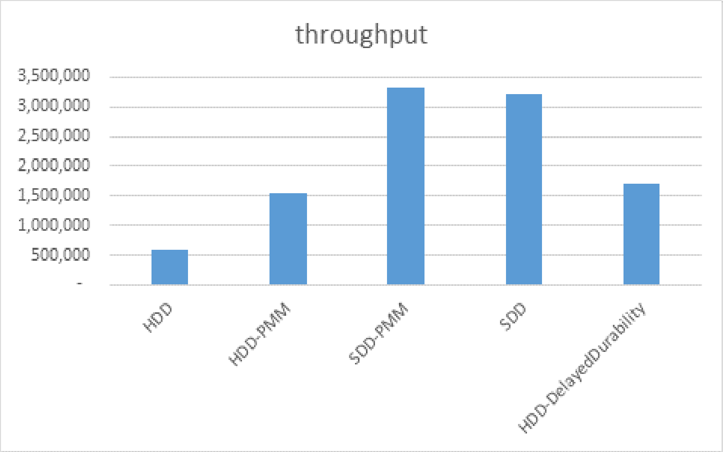


		Flavor													
		base							pmm						
		1	2	4	8	16	32	64	1	2	4	8	16	32	64
Latency (us)	64	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	128	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	26%	34%	33%
	256	0%	0%	0%	0%	7%	28%	13%	0%	0%	0%	0%	8%	7%	6%
	512	99%	98%	87%	79%	56%	7%	16%	100%	100%	100%	99%	61%	52%	54%
	1,024	1%	2%	13%	21%	37%	61%	60%	0%	0%	0%	1%	5%	7%	8%
	2,048	0%	0%	0%	0%	0%	4%	11%	0%	0%	0%	0%	0%	0%	0%
	4,096	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	8,192	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	16,384	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	32,768	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	65,536	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	131,072	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	262,144	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	524,288	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	1,048,576	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	2,097,152	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
Load Time									50%	53%	66%	97%	50%	30%	36%
Updates/s									108%	111%	104%	107%	132%	125%	105%

PMM Log – *Not* TPCC, 2S



HDD	601K
HDD-PMM	1,555K
SDD-PMM	3,321K (109.6%)
SDD	3,214K (100%)
HDD-Lazy	1,720K



		HDD	HDD-PMM	SDD-PMM	SDD	HDD-DelayedDurability
64	us]	0.0%	0.0%	0.0%	0.0%	0.0%
128	us]	0.0%	0.1%	**36.3%**	3.5%	0.7%
256	us]	3.9%	**39.2%**	**38.3%**	0.9%	**43.1%**
512	us]	0.4%	**34.0%**	**20.7%**	**20.1%**	**35.5%**
1024	us]	4.4%	**10.4%**	4.5%	**57.6%**	**9.3%**
2048	us]	0.0%	4.2%	0.1%	**14.2%**	3.3%
4096	us]	0.1%	3.0%	0.0%	2.6%	2.1%
8192	us]	**17.6%**	4.7%	0.0%	0.9%	3.2%
16384	us]	**68.2%**	2.6%	0.0%	0.2%	1.6%
32768	us]	**5.0%**	1.0%	0.0%	0.0%	0.7%
65536	us]	0.3%	0.6%	0.0%	0.0%	0.5%
131072	us]	0.1%	0.1%	0.0%	0.0%	0.1%
262144	us]	0.0%	0.0%	0.0%	0.0%	0.0%
524288	us]	0.0%	0.0%	0.0%	0.0%	0.0%
1048576	us]	0.0%	0.0%	0.0%	0.0%	0.0%
2097152	us]	0.0%	0.0%	0.0%	0.0%	0.0%

PMM Log - Implementation

- Add-on over the Sql Log Manager
- 1:1 relationship between *existing* log staging area (LC) and *new* PMM log store memory
- Adding a log record (algorithm outline):
  - Obtain LSN
  - Copy to existing LC slot and then copy to PMM
  - Atomically attach PMM log record to PMM log store
  - If log record is a transaction commit, cl-flush PMM log records
- Surprise: Increased throughput with longer code path!
  - IO path is shorter; IO blocks are larger; lazy commit behavior
- No free lunch: Recovery needs to account for “holes”
  - Still very simple – just move content from PMM to regular log

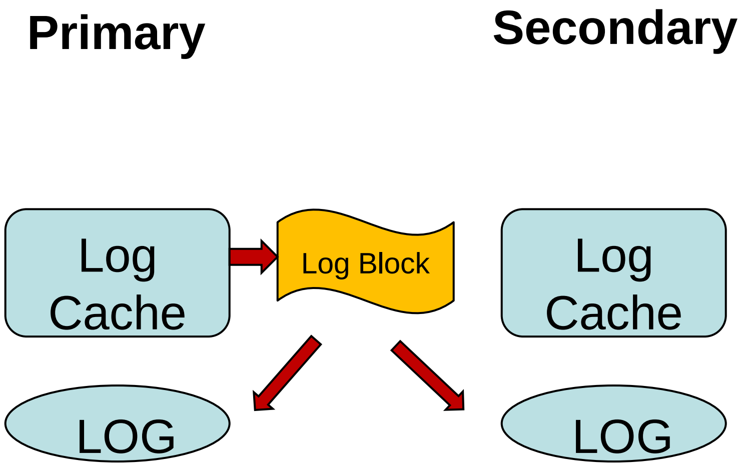
What this talk is about

- Trying to answer the question: What does it take for an in-memory database to out-compete a disk based database on all storage-related metrics?
- Experience with Persistent Main Memory for In-Memory Databases

- **Talk Outline:**
- ~~Definitions and terms~~
- ~~Transaction log in persistent main memory~~
- Impact on database high availability
- Checkpoint, recovery and very large databases

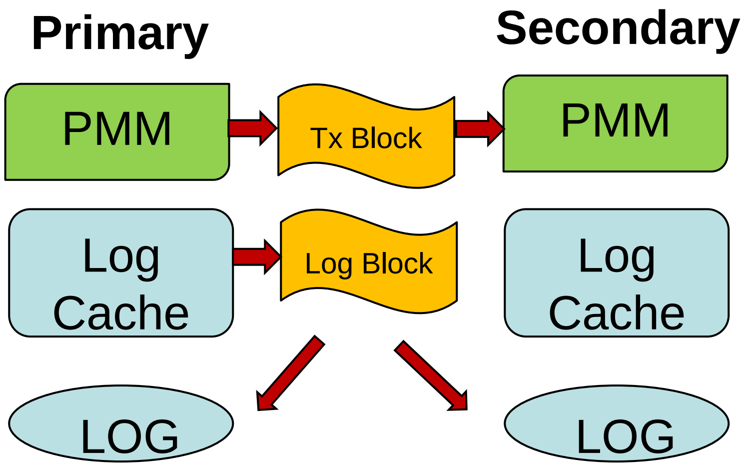
Impact on High Availability

- Sql Server HA: send log blocks to secondary replicas
- PMM log side-effect: larger (and slower) block creation
- Solution: Send content on the tx commit path **as well**



Primary

Secondary



Primary

Secondary

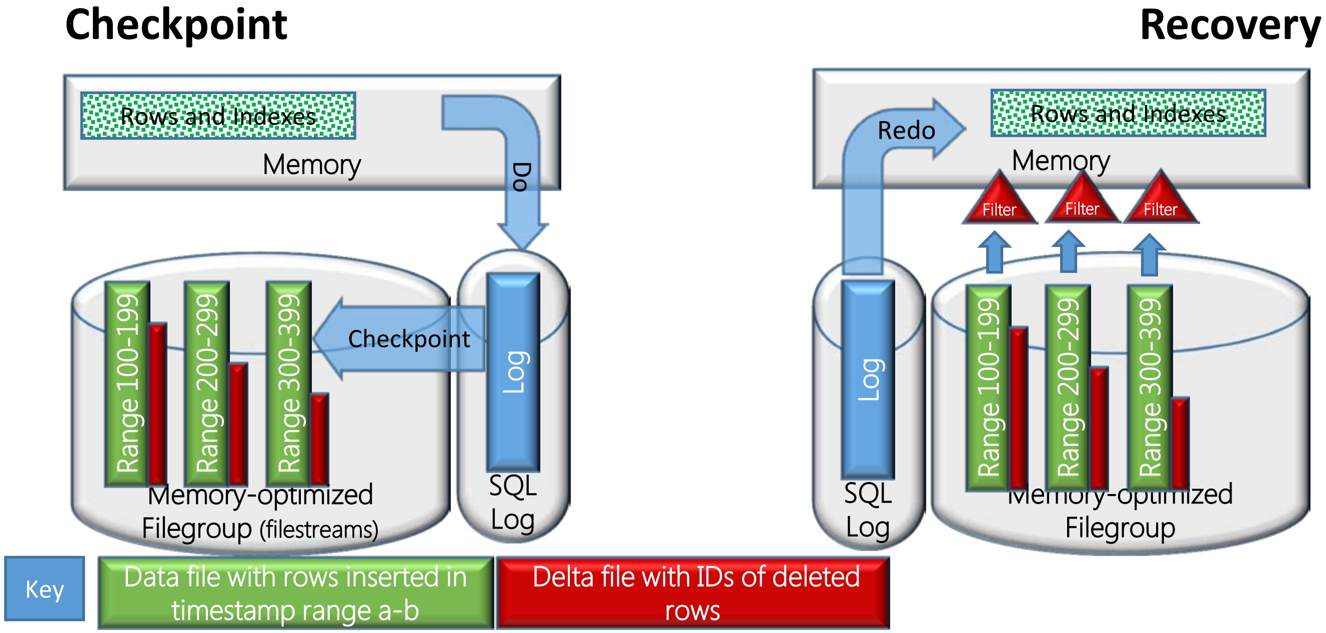
- Consequences: double the network traffic
- Add roughly 25us of latency to transaction commit
- Which does not translate into loss of throughput (context-free work is valuable)

What this talk is about

- Trying to answer the question: What does it take for an in-memory database to out-compete a disk based database on all storage-related metrics?
- Experience with Persistent Main Memory for In-Memory Databases

- **Talk Outline:**
- ~~Definitions and terms~~
- ~~Transaction log in persistent main memory~~
- ~~Impact on database high availability~~
- Checkpoint, recovery and very large databases

Hekaton checkpoint and recovery



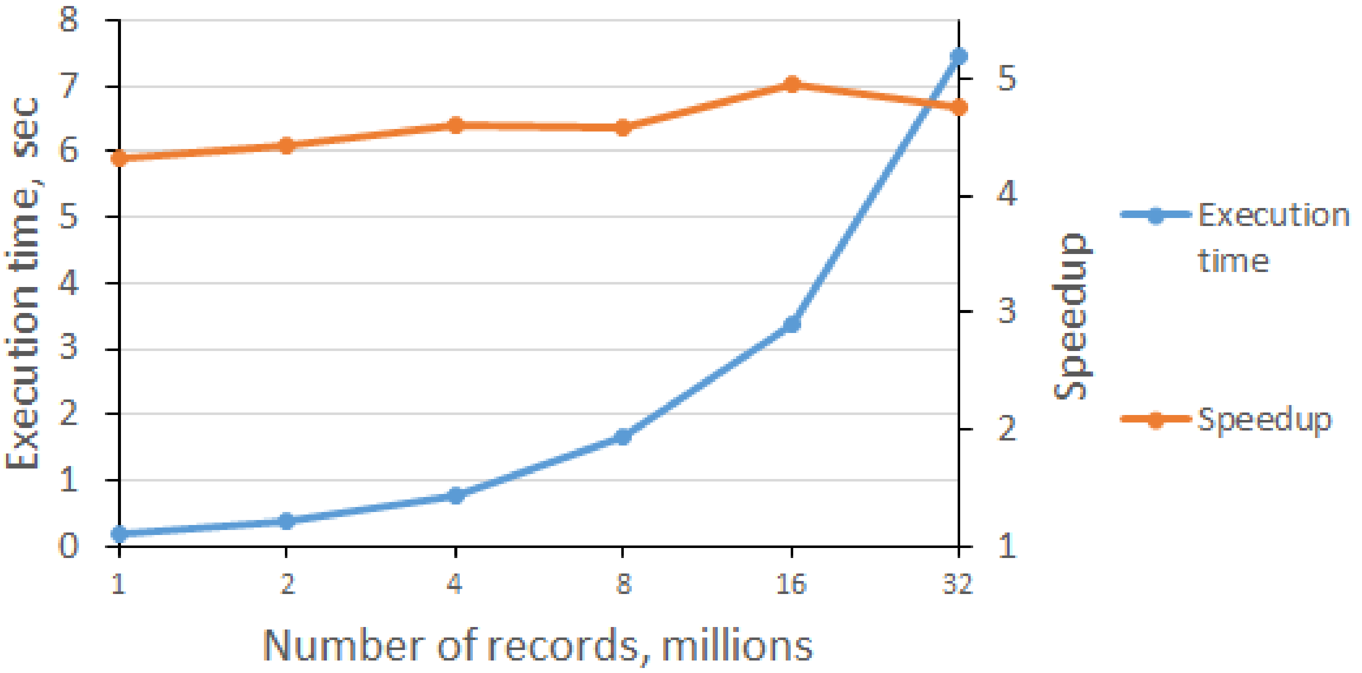
Hekaton Checkpoint

- Fully parallel at both creation and recovery time
- Scalable and high throughput: 1G/second, external limit
- Fully integrated with Sql: encryption, backup/restore, IO resource governance, space management, etc.
- Can be produced from log stream alone (remote-able)

- Loading 1TB of data is slow, regardless of parallelism
  - Assume IO at 1G/s leads to 1000s (~17min) recovery time
- Many indices and fast IO can make it appear slower
- This is where in-memory DB has had an inherent disadvantage!

What to do? O(N) attempt…

- Sort rows before loading them.



- Results in significant time reduction, but still O(N).
- We want constant time recovery instead.

What to do? O(1) attempt…

- Introduce GenerationEra – incremental value which tracks new instances of the host process lifetime.
- Every object is marked with its GenerationEra.
- Differentiate between ‘visibility’ and ‘reachability’.
- Analyze each container: heap, bw-tree index, hash index, free lists to identify object lifetime.
- Use a mark and sweep approach to move eligible old-era items to the current era’s freelist.
- Sweep happens in parallel with DB becoming available.
- Result: DB is available in small constant time.
- Insight: Requires hardware support for ‘reachability’.

Credits

- Paul Larson, Bob Fitzgerald, Ildar Absalyamov – MSR
- Sridharan Sakthivelu – Intel
- Hekaton Engine Team – Microsoft Sql Server