In-Memory Technologie Hekaton

Benjamin Stahl 13 INM

11. Juli 2014

Inhaltsverzeichnis I

1. Einleitende Worte
2. Architekturprinzipien
3. Hekaton Integration im SQL Server
4. Storage and Indexing
5. BW-Tree
6. Quellen

Hekaton... I

- griech:**Hekaton** 100 (Zahlwort)
- Einsatz für OLTP (Echtzeit-Transaktionsverarbeitung)
- Moorsches Gesetz kommt an ein Limit
- Anpassung an neue Hardware $\rightarrow$ Flash Speicher & Multicore CPUs & massig Speicher
- Hekaton Engine ist kein separates DBMS, sondern Teil des SQL Server

Hekaton... II

1. Verbesserung der Skalierbarkeit
2. Verbesserung der CPI (cycles per instruction)
3. Reduzierung der auszuführenden Anweisungen pro Anfrage

Hekaton... III

- Die Punkte 1 & 2 können nur eine Verbesserung von Faktor 3-4 erreichen
- Der größte Faktor kann mit der Reduzierung der auszuführenden Anweisungen pro Anfrage erreicht werden

Optimierte Indizes I

- Hekaton Indizes sind für Speicherresidente Daten optimiert
- Beständigkeit der Indexe wird sichergestellt durch „logging“ und „checkpointing“ von Datensätzen auf externen Speicher
- Hekaton Tabellen und deren Indexe können komplett vom letzten Checkpoint bzw. den Logs hergestellt werden
- Skalierbarkeit darf nicht durch Shared Memory negativ beeinflusst werden (Latches und Locks)

Latch & Lock-Freiheit I

- Hekatons interne Datenstrukturen wie z.B. die Hash und Range Indizes und die „Transaction map“ sind komplett Latch-frei (lock-free)
- Die Kombination aus Latch-freien Datenstrukturen, einer optimistischen „concurrency control“ und dem „multiversioning“, führt dazu, dass Threads nicht warten müssen oder diese gar aushungern werden

Anfragen → Maschinencode I

- Anfragen und Stored Procedures die in T-SQL geschrieben sind, werden in effizienten Maschinencode überführt
- Der so generierte Code enthält nur die Anweisungen die er wirklich für seine Anfrage brauch
- Alle Datentypen sind zum Compile-Zeitpunkt bekannt und sorgen für effiziente Code-Generierung

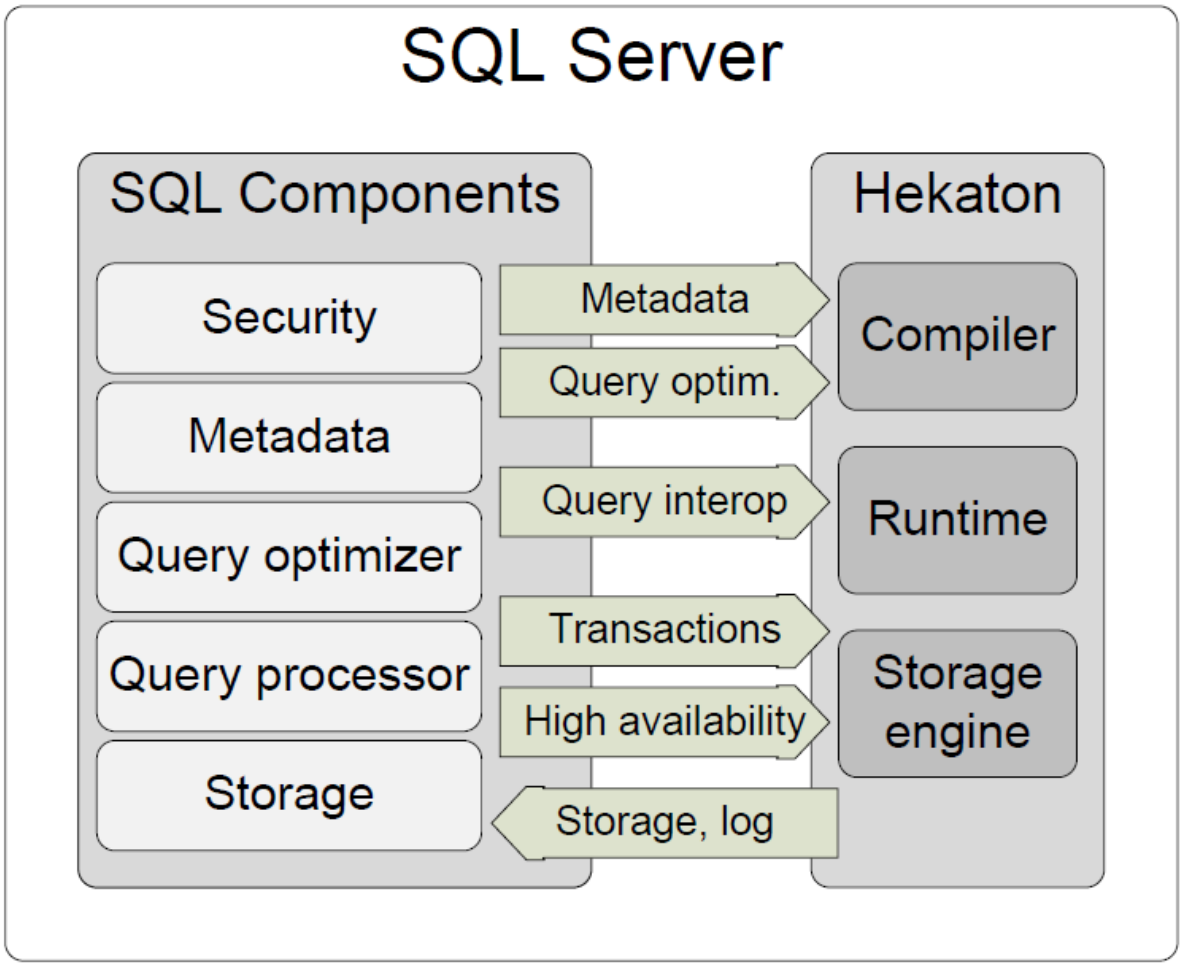
Keine Partitionierung I

- Andere Systeme wie HyPer, Dora, H-store und VoltDB die für OLTP designt wurden, partitionieren ihre Datenbank nach Kernen und geben diesen dann exklusiven Zugriff auf die Partition

Gutes Prinzip...ABER!

Funktioniert nur wenn auch der Workload partitionierbar ist!

High-Level Architektur



High-Level Architektur I

Hekaton Storage Engine

- Organisiert Nutzerdaten und Indizes
- Stellt Transaktionsoperationen für Datentabellen, Hash und Range Indizes bereit
- Weiterhin werden Basismechanismen für Speicherung, Checkpointing und Recovery bereitgestellt

Hekaton Compiler

- T-SQL Prozeduren werden mitsamt ihrer Queries, Tabellen und Indexmetadaten in eine abstrakte Baum Repräsentation überführt und anschließend in Maschinencode übersetzt

High-Level Architektur II

Hekaton Runtime System

- Kleine Komponente die zur Integration von SQL Server Ressourcen benötigt wird

Metadata

- Metadaten über Hekaton Tabellen, Indizes etc. werden im regulären SQL Server Katalog gespeichert

Query Optimization

- Eingebettete Queries werden regulär vom SQL Server Optimierer optimiert

High-Level Architektur III

Query interop

- Hekaton stellt Operatoren zum zugreifen auf Daten in Hekaton Tabellen bereit, welche im SQL Server Ausführungsplan verwendet werden können

Transactions

- SQL Server Transaktionen können sowohl Hekaton als auch normale Tabellen manipulieren

High Availability

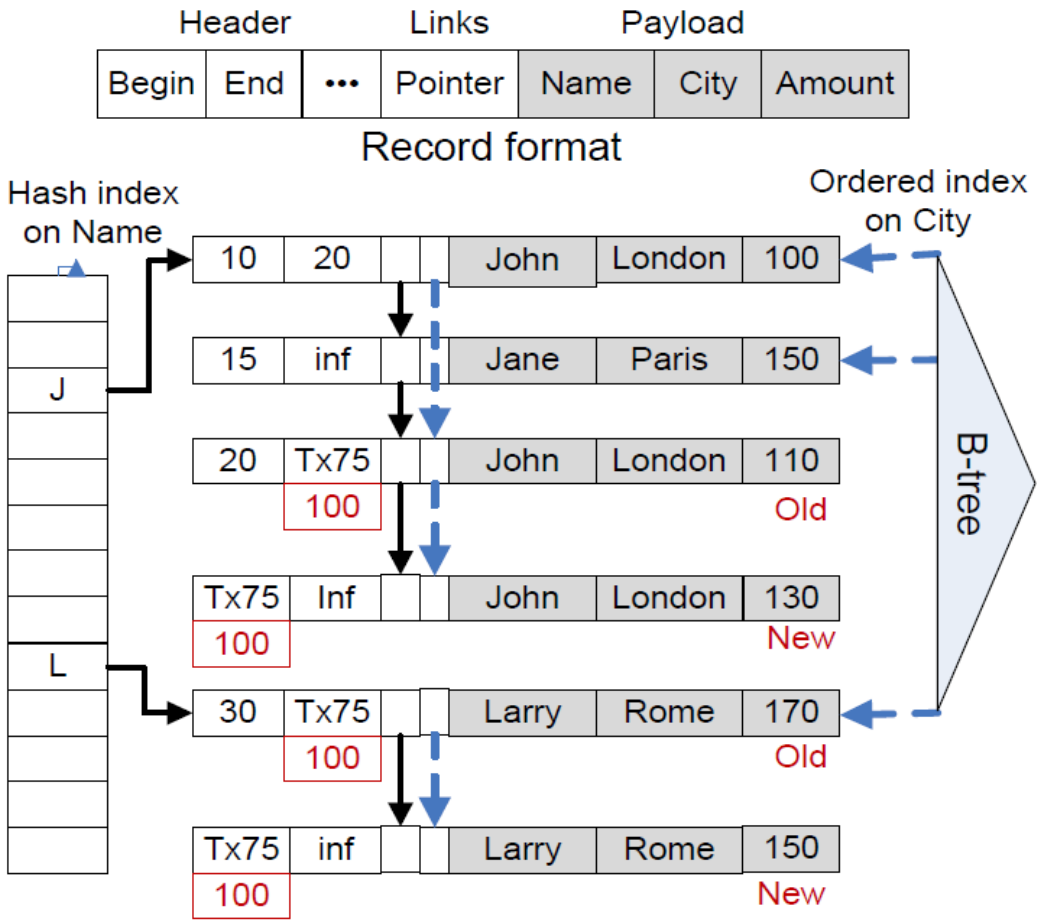
- AlwaysOn Feature auch hier verfügbar
- Hekaton Tabellen sind auch auf Zweitservern lesbar

High-Level Architektur IV

Storage, log:

- Hekaton logt seine Updates im normalen SQL Server Transaktionslog

Reads & Updates



BW-Tree I

Bw = „Buzz Word“

„A Latch-free, Log-structured B-tree for Multi-core Machines with Large Main Memories and Flash Storage“

Architektur I

- Der Bw-Tree atomic record store (ARS) ist ein klassischer B^+ - Baum
- Anstatt Latches zu verwenden, wurde für Zustandsänderungen ein atomarer „compare and swap“ (CAS) Befehl eingeführt
- Es wird nur geblockt, wenn der Bw-Tree Seiten aus dem stabilen Speicher holen muss (LSS), was relativ selten passiert bei einem großen Arbeitsspeicher

Architektur II

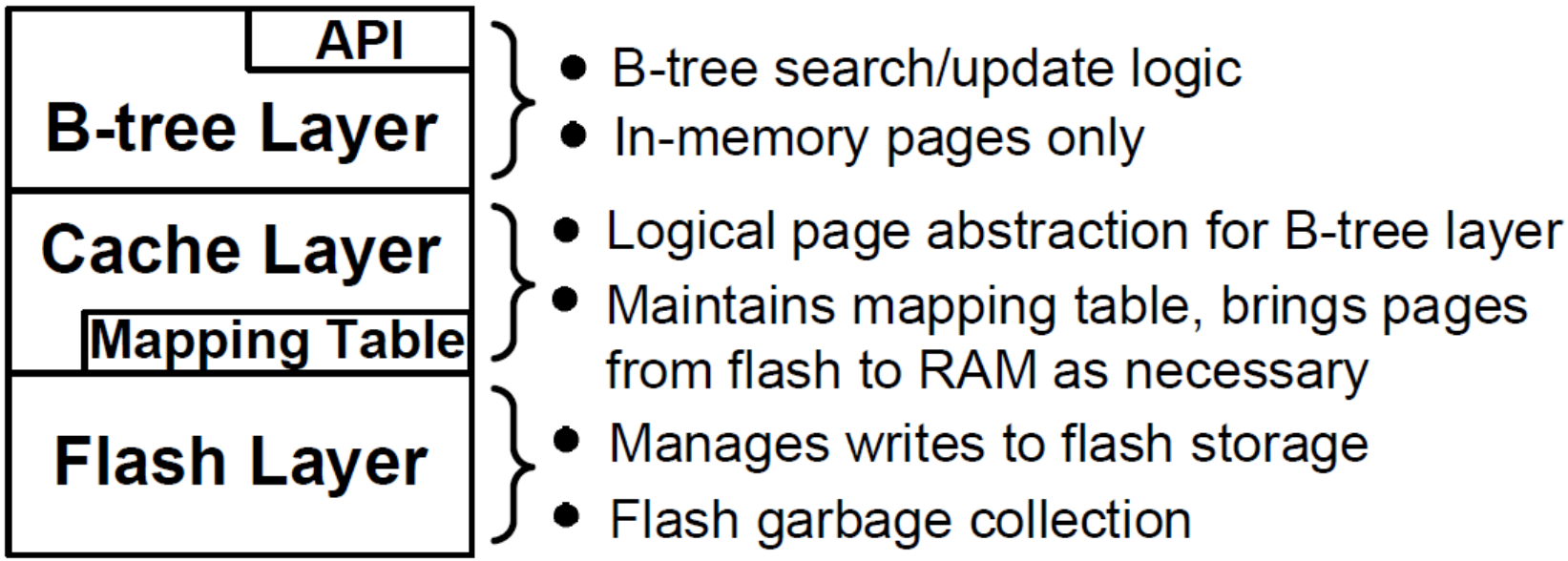


Abbildung: Architektur des Bw-tree atomic record store

Mapping Table I

- Verknüpft logische Seiten zu physikalischen Seiten
- Logische Seiten werden mit einem „page identifier“ (PID) identifiziert
- Die Mapping Table übersetzt die PID entweder in ein Flash Offset, also die Adresse einer Seite im „stable storage“
- Oder aber in einen Zeiger, der auf eine Seite im Hauptspeicher verweist
- Die Innovation: PIDs werden zum verknüpfen von Blättern im Baum verwendet

Delta Updating I

- Seiten Zustandsänderungen werden durch das Erzeugen eines „delta record“ realisiert
- Diese werden einem existierenden Seitenzustand vorangestellt
- In der Slot Seite für die physikalische Adresse der Mapping Tabelle, wird die Speicheradresse des (neuen) „delta record“ eingetragen
- Dazu wird die „compare and swap“ (CAS) Instruction verwendet
- Wenn dies erfolgreich war, wird die „delta record“ Adresse zur neuen physikalischen Adresse der Seite

Bw-tree Structure Modifications I

- Latches schützen Teile unseren Index-Baumes während einer Strukturänderung nicht
- Bsp. Pagesplit
- Um das Problem zu umgehen werden die Strukturänderungen in eine Sequenz von CAS Befehlen überführt

Quellen I

... Cristian Diaconu, Craig Freedman.
Hekaton: SQL Server's Memory-Optimized OLTP Engine.
Microsoft Research, 1:12, 2013.

Sudipta Sengupta Justin J. LEvandoski, David B. Lomet.
The Bw-Tree: A B-tree for New Hardware Platforms.
Microsoft Research, 1:7, 2013.

... Per-Ake Larson, Spyros Blanas.
High-Performance Concurrency Control Mechanisms for Main-Memory Databases .
Microsoft Research, University of Wisconsin – Madison, 1:11, 2013.