TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung
***Einsatz und Realisierung von Datenbanksystemen* im SoSe16**
Moritz Kaufmann (moritz.kaufmann@tum.de)
http://db.in.tum.de/teaching/ss16/impldb/

Blatt Nr. 08

Hausaufgabe 1

In traditionellen Datenbanksystemen sind die Festplatte und der Buffermanager oft der Hauptgrund für Performanceengpässe. Wie ändert sich dies in Hauptspeicherdatenbanken, wo sind die neuen Flaschenhälse? Unterscheiden Sie auch zwischen Analytischen und Transaktionalen Workloads.

Der Unterschied zwischen traditionellen Datenbanksystemen und Hauptspeicherdatenbanke ist, dass wir in der Speicherhierarchie ein paar Stufen nach oben gehen. Hauptspeicher ist teurer, aber gleichzeitig auch schneller und hat eine geringere Latenz. Genauso wichtig ist aber auch, dass die Daten nun alle in einem Adressraum liegen. Bei der Nutzung von Festplatten muss das Datenbanksystem explizit die Daten von der Festplatte in den Speicher laden. Beim Hauptspeicher ist der Wechsel zwisch RAM, L3 und L1 Cache transparent für das System. Das heißt ein Buffermanager wird nicht mehr benötigt. Auch wenn der Wechsel zwischen den verschiedenen Hauptspeicherhierarchien für das System nicht explizit sichtbar ist, so ist es doch in der Performance bemerkbar. Der Latenzunterschied zwischen RAM und L3 ist ähnlich groß wie zwischen Festplatten und RAM. Die Datenbank muss nun so strukturiert werden, dass möglichst viele Operationen auf Daten in den schnelleren Speicherschichten ausgeführt werden. Ein weiterer neuerer Flaschenhals sind die Lockingverfahren. Im Vergleich zu einfachen Operationen wie Lesen, Schreiben, Addition, etc. ist ein Lock zu erstellen viel teurer. Viele Hauptspeichersystem versuchen daher Locks zu vermeiden. Ein Problem, dass hauptsächlich nur Transaktionale Workloads betrifft ist, dass beim Ändern von Daten die Änderung immer noch persistiert werden muss (Logging). Es reicht nicht aus, die Daten nur im Hauptspeicher zu halten, da diese dann nach einem Systemabsturz verloren wären. Das Schreiben auf Festplatte ist selbst mit SSDs noch wesentlich teurer als Änderungen im Hauptspeicher vorzunehmen. Auch ein Problem in Transaktionale Workloads ist die Annahme der Anfragen. Transaktionale Anfragen sind typischerweiße sehr schnell. Ein einzelner Rechner kann sehr einfach mehrere Hunderttausend Anfragen verarbeiten. Hier ist die Netzwerklatenz auch wesentlich größer als die Verarbeitungszeit der Anfragen. Daher kann ein einzelner Client die Datenbank garnicht auslasten wenn er jede Anfrage einzeln abschickt.

Hausaufgabe 2

Wie ändert sich die Bedeutung des Redo-Log und Undo-Log in Hauptspeicherdatenbanken im Vergleich zu klassischen Datenbanken? Wo werden sie gespeichert?

Da die Daten nicht mehr auf der Festplatte gespeichert werden, müssen sie (falls es keine Snapshots gibt) bei einem Neustart komplett auf Basis des Redo-Logs wiederhergestellt werden. Da nie die Daten einer nicht comitteten Transaktion auf der Platte landen wird die Undo-Log nur benötigt um im Fall eines Abort die Ã„nderungen der Transaktion zurückzusetzen. Außerdem reicht es aus die Redo Log Daten erst beim Commit zu schreiben. Aus diesem Grund sind die Daten der Undo-Log nur zur Laufzeit der Transaktion notwendig und müssen nicht auf die Festplatte geschrieben werden. Während der Wiederherstellung der Datenbank können dann einfach alle Redo Log Einträge wiederhergestellt werden.

Hausaufgabe 3

Sie sollen für ein Versandhaus die Datenbank für ein Hauptspeicherdatenbanksystem optimieren. In dem System sind die Daten der letzten *drei Jahre* gespeichert. Das Schema der verschiedenen Relation ist unten beschrieben.Wählen sie jeweils eine Repräsentation der Daten für die Relationen (z.B. Spalten- oder Zeilenorientert), so dass zur Beantwortung der unten beschriebenen Anfragen möglichst wenig Daten in den CPU-Cache geladen werden müssen. Es existieren Indexe auf VerkaufsId in Verkauf, (KundenId, VerkaufsId) in KundenKäufe und KundenId in Kunde. Es können aber keine neuen Indexe definiert werden. Text wird direkt innerhalb der jeweiligen Spalte / Zeile gespeichert. '

Relationen

Verkauf

VerkaufsId (8 byte), Datum (16 byte), Uhrzeit (16 byte), IP (16 byte), Betrag (16 byte), Versandart (8 byte), Kommentar (48 byte)

KundenKäufe

KundenId (8 byte), VerkaufsId (8 byte)

Kunde

KundenId (8 byte), Anrede(8 byte), Vorname(8 byte), Nachname(8 byte), Einstufung(8 Byte), Anschrift(256 byte), Land(64 byte), Email (16 Byte), Facebook(32 byte), GPlus(32 Byte), WerbungsFrequenz(8 byte)

Anfragen mit Ausführungshäufigkeit:

1. 10000x select * from Verkauf
   where Datum > '%d'::date and Datum < '%d'::date + interval '1' month;
2. 100x select * from Verkauf;
3. 100x select count(*) from KundenKäufe group by KundenId;
4. 10000x select Anrede, Vorname, Nachname, Einstufung, Email, WerbungsFrequenz from Kunde where KundenId = '%id';

Würden Sie ihre Entscheidung ändern, wenn zusätzlich noch die folgenden Anfragen ausgeführt werden müssten?

1. 5000x insert into Verkauf VALUES (...);
2. 5000x insert into KundenKäufe VALUES (...);
3. 100x insert into Kunde VALUES (...);

Zur Lösung der Aufgabe berechnen wir für jeden Anfragetyp die Auswertungskosten als Anzahl der Cachelines die in den CPU-Cache geladen werden müssen. Der allgemeine Ansatz dabei ist, für jeden Anfragetyp zu analysieren wie die Anfrage vom Datenbanksystem ausgeführt werden kann (z.B. muss die ganze Tabelle betrachtet werden oder kann ein Index genutzt werden?). Für diese Ausführungspläne müssen dann die Kosten der verschiedenen Datenorganisationsmöglichkeiten berechnet werden.

Verkaufs Relation

Diese Relation betreffen drei Anfragen. Die Anfragen 1 und 2 und die Insert Anfrage 1. Eine Tupel benötigt 128 byte Speicherplatz, somit 2 Cachelines.

1. Anfrage: Keiner der existierenden Indexe hilft, um die *where* Bedingung auszuwerten. Daher müssen alle Einträge in der Tabelle überprüft werden (Tablescan). Die Selektivität des Prädikats kann mit $\frac{1}{AnzahlMonateinDatenbank} = \frac{1}{36}$ abeschätzt werden.

Row store Zur Auswertung des Prädikats muss für jedes Tupel die Cacheline mit der Datumsinformation geladen werden. Für alle zutreffenden Tupel müssen zur Ausgabe auch die restlichen Felder und somit auch die zweite Cacheline geladen werden. Daraus ergibt sich folgende Formel ($|V|$ = Größe der Verkaufsrelation).

$\text{Kosten} = |V| + \frac{|V|}{36} = \frac{|V|*37}{36}$

Column store Zur Auswertung des *where* Prädikats reicht es nur die Datumsspalte zu laden und nur wenn dieses erfüllt ist die restlichen Prädikate. Durch das Spaltenlayout liegen 4 Datumseinträge innerhalb einer Cacheline. Da die Werte sequentiell analysiert werden können, kann mit einer Cacheline das *where* Prädikat von 4 Tupeln überprüft werden. Beim nachladen der anderen Spalten (6) bei zutreffenden Datumswerte entsteht jedes mal ein Cachemiss.

$\text{Kosten} = \frac{|V|}{4} + \frac{|V|*6}{36} = \frac{|V|*15}{36}$

2. Anfrage: Hier wird jeweils die gesamte Tabelle gelesen.

Row store $\text{Kosten} = |V| * 2$

Column store Da die Daten sequentiell ohne Einschränkung gelesen werden, können jeweils immer alle Daten aus den Cachelines genutzt werden.

$\text{Kosten} = \frac{|V|}{8} + \frac{|V|}{4} + \frac{|V|}{4} + \frac{|V|}{4} + \frac{|V|}{4} + \frac{|V|}{8} + \frac{|V|*48}{64} = |V| * 2$

1. Insert Anfrage:

Row store Kosten = 2

Column store Zum Einfügen muss für jede Spalte die entsprechende Cacheline geladen werden.

Kosten = 7

Wenn die Insertanfragen ignoriert werden, wäre ein Column store die optimale Datenstruktur. Zur Abschätzung mit Berücksichtigung der Inserts müssen die Kosten aller Anfragen mit der Ausführungshäuftigkeit gewichtet werden.

Row store Kosten = $10000 * \frac{|V|*37}{36} + 100 * |V| * 2 + 5000 * 2$

Column store Kosten = $10000 * \frac{|V|*15}{36} + 100 * |V| * 2 + 5000 * 7$

Ab $|V| > 4$ ist der Column store auch dann noch die Beste Wahl.

KundenKäufe
Diese Relation betreffen Anfragetyp 3 und Insert Anfragetyp 2.
3. Anfrage: Der Index kann hier nicht genutzt werden, da wir nur auf KundenID aggregieren. Das heißt die Datenbank muss wieder alle Werte überprüfen.

Row store Es können 4 Werte pro Cacheline geladen werden.

Kosten = $\frac{|KK|}{4}$

Column store Hier reicht es nur die KundenId Spalte zu laden.

Kosten = $\frac{|KK|}{8}$

2. Insert Anfrage:

Row store Kosten = 1

Column store Kosten = 2

Bei der Analyse unter Berücksichtigung der Insert Anfrage ergeben sich folgende Kosten:

Row store Kosten = $100 * \frac{|KK|}{4} + 5000 * 1$

Column store Kosten = $100 * \frac{|KK|}{8} + 5000 * 2$

Bei weniger als 400 Werten in KundenKäufe ist ein Row store besser, bei mehr ein Column store.

Kunde
Diese Relation wird in Anfragetyp 4 und Insert Anfrage 3 genutzt.
4. Anfragetyp: Hier können wir den Index auf KundenId nutzen und müssen somit nur für diese Zeile alle Werte laden.

Row Store Hier müssen wir für jeden Datensatz die Cachelines laden, deren Daten wir benötigen. Dies sind 3. Bonus: Die Kosten für den Rowstore lässt sich durch umordnen der Spalten innerhalb jeder Zeile auf 1 Cacheline reduzieren.

Kosten = 3

Column Store Basierend auf der Annahme, dass der Cache leer, verursacht jede Spalte einen Cache-Miss.

Kosten = 6

3. Insert Anfrage

Row Store Kosten = 7

Column Store Hier muss berücksichtigt werden, dass zum Einfügen der Anschrift 4 Cachelines angefasst werden.

Kosten = $10 + 4$

Optimales Layout ist für diese Tabelle ein Rowstore.