Aspekte der Konsistenzsicherung beim Constraint-basierten Datenbank-Caching

Martin Tritschler, Volker Hudlet, Joachim Klein

AG Datenbanken und Informationssysteme, Fachbereich Informatik
Technische Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
{m_trit, v_hudlet, jklein}@informatik.uni-kl.de

Zusammenfassung: Beim Datenbank-Caching werden Satzmengen häufig angefragter Prädikate in der Nähe von Anwendungen vorgehalten, um den Zugriff auf die Daten zu beschleunigen. Um entscheiden zu können, ob eine Anfrage durch das Cache-System beantwortet werden kann, verwendet das Constraint-basierte Datenbank-Caching einfache Bedingungen (sog. Constraints), welche die Vollständigkeit einzelner Prädikatsextensionen zusichern. Da die Gültigkeit der definierten Constraints im Cache jederzeit zu gewährleisten ist, müssen die in einem Cache gelagerten Satzmengen bei DB-Änderungen, die sie betreffen, aktualisiert werden. Dieser Aufsatz diskutiert ein Verfahren, wie eine konsistente Aktualisierung gewährleistet werden kann. Für die einzelnen Teilschritte der Erfassung, Weiterleitung und Übernahme der Änderung werden verschiedene Lösungen diskutiert, die jeweiligen Vor- und Nachteile erläutert sowie einige Implementierungskonzepte vorgestellt.

1 Motivation

Das Datenbank-Caching ermöglicht eine Beschleunigung des Zugriffs auf Daten evtl. weit entfernter Datenbanksysteme, wobei häufig zugegriffene Datenelemente in der Nähe von Anwendungen bereitgestellt werden. Im Gegensatz zu Ansätzen, die auf der Verwendung materialisierter Sichten [LGZ04] basieren, hält das Constraint-basierte Datenbank-Caching (CbDBC) Satzmengen für einzelne Prädikate vor, deren Konsistenz mithilfe von Constraints kontrolliert und eingehalten wird (ähnlich wie in Ansätzen von Oracle TimesTen [The02] und IBM [BAM+04]). Dies ermöglicht eine dynamische und adaptive Anpassung des Cache-Inhaltes an die auftretende Anfragelast [BHM06], was durch einfaches Kopieren ganzer Tabellen (Full-Table-Caching) nicht gewährleistet werden kann.

Das DBMS der zentralen Primärkopie (Backend), welche beim CbDBC typischerweise einen oder mehrere Caches verwaltet, muss diesen laufend die Änderungen von DB-Operationen übermitteln. Aufgrund der durch die Constraints entstehenden Abhängigkeitsketten wird die Aktualisierung der jeweiligen Cache-Inhalte jedoch erschwert. Die korrekte und performante Propagierung von Änderungen ist jedoch eine wesentliche Grundlage der Implementierung einer globalen Transaktionskontrolle. Voraussetzung für die Propagierung der Änderungen ist deren Erfassung, welche bereits vor dem Com-

mit durchzuführen ist. In Abschnitt 3 werden dazu gängige Methoden und das Schema des ermittelten *Write Set* (WS) vorgestellt. Die in einem WS protokollierten Änderungen müssen nur an diejenigen Caches weitergeleitet werden, die auch zu ändernde Datensätze vorhalten. Die dazu notwendige Verteilungsentscheidung wird in Abschnitt 4 vorgestellt, wobei verschiedene Wissensstände (Umfang an Metainformation über den Zustand eines Caches) für das Backend unterschieden werden. Innerhalb der Cache-Instanzen ist bei den propagierten Änderungen zu überprüfen, ob diese bestehende Constraints verletzen und daher zusätzliche Ladevorgänge (oder evtl. Entladevorgänge) zur Sicherung der Konsistenz nötig werden (vgl. Abschnitt 5). Die dafür benötigten Grundlagen werden im folgenden Abschnitt eingeführt.

2 Grundlagen des Constraint-basierten Datenbank-Caching

Ein CbDBC-System hält die Sätze einzelner Prädikatsextensionen aus Backend-Tabellen in entsprechenden Cache-Tabellen vor. Dabei wird einer Backend-Tabelle T die korrespondierende Cache-Tabelle T_C zugeordnet, wobei deren Definition mit Ausnahme der Fremdschlüsselbeziehungen vollständig übernommen wird. Zusätzlich zu den Tabellen verwaltet der Cache zwei Arten von Constraints: *Füllspalten* (filling column, FC) und *referenzielle Cache Constraints* (RCC). In beiden Constraint-Definitionen wird der Begriff der *Wertvollständigkeit* verwendet: Der Wert v einer Spalte $T_C.a$ wird genau dann als wertvollständig bezeichnet, wenn alle Sätze der zugehörigen Backend-Tabelle T, die den Spaltenwert v enthalten, im Cache verfügbar sind [HB07]. Die Menge aller Cache-Tabellen wird zusammen mit der Menge aller definierten Constraints als *Cache Group* bezeichnet.

RCC. Ein RCC $S_C.s \rightarrow T_C.d$ ist eine Beziehung innerhalb einer Cache Group zwischen zwei Cache-Tabellen-Spalten, die den gleichen Wertebereich haben. Die Spalte $S_C.s$ wird Quellspalte und die Spalte $T_C.d$ Zielspalte genannt. Ein RCC ist genau dann erfüllt, wenn alle Werte v, die in der Quellspalte des RCC enthalten sind, in der Zielspalte wertvollständig vorliegen. Die RCC-Quellspalte erzwingt also für alle in ihr enthaltenen Werte die Wertvollständigkeit in der RCC-Zielspalte [Bra08].

FC. Die Füllspalten steuern den Ladeprozess von Werten im Cache. Mit jeder Füllspalte $T_C.f$ wird eine Menge aller ladbaren Füllspaltenwerte, die Kandidatenmenge K, assoziiert. Wenn eine Anfrage einen Füllwert $v \in K$ einer Füllspalte $T_C.f$ explizit referenziert, kann die Anfrageauswertung v nur benutzen, falls er in $T_C.f$ wertvollständig vorliegt. Ist der Wert noch nicht wertvollständig im Cache vorhanden, so ist er nachzuladen. Hat T_C ausgehende RCCs, setzt sich die Forderung nach Wertvollständigkeit, und somit das Laden weiterer Sätze, über die RCC-Ketten zu anderen Cache-Tabellen fort. Die gesamte Satzmenge, die aufgrund eines Füllwertes v in den Cache geladen werden muss, wird hierbei als *Cache Unit* von v bezeichnet [Bra08]. Auf die konkrete Vorgehensweise beim Laden und Entladen von Werten wird in [BHM06, KBM09] genauer eingegangen.

In Abbildung 1 ist eine Cache Group für die Backend-Tabellen S und T dargestellt. Über die Füllspalte $S_C.s3$ ist momentan nur der Satz (1, a, 10000) mit dem Wert 10000 geladen. Aufgrund des RCC $S_C.s2 \rightarrow T_C.t2$ muss der Wert a in der Zielspalte $T_C.t2$ wertvollständig sein. Daher müssen in T_C alle Sätze mit $T.t2 = a$ vorgehalten werden, wodurch z. B. auf dem Cache die Anfrage $\sigma_{t2=a}T$ beantwortet werden kann.

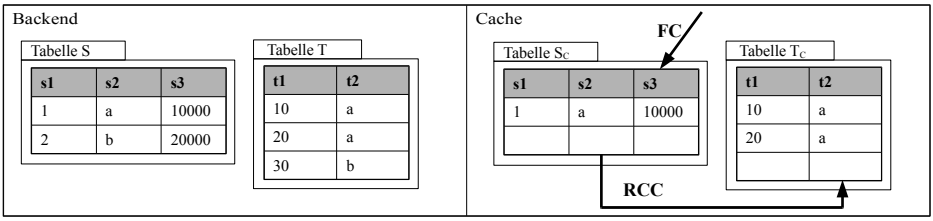

Abbildung 1: Backend-Tabellen *S* und *T* sowie Beispiel einer entsprechenden Cache Group

3 Erfassung der Änderungen

Das CbDBC-System des ACCache-Projekts [BHM06] ist als Middleware realisiert und setzt daher direkt auf SQL auf. Im Rahmen einer prototypisch implementierten globalen Transaktionskontrolle kommt eine Primary-Copy-Strategie zum Einsatz, wobei alle schreibenden Änderungen nur auf dem Backend stattfinden. Die Änderungen werden im Backend vor Commit durch den *WS-Manager* erfasst, der die geänderten Datensätze in Form eines WS aufbereitet. Zur Ermittlung dieser Änderungsinformationen haben sich zwei Vorgehensweisen etabliert [Ora07]. Beim Log-Sniffing wird asynchron auf die Log-Informationen des DB-Systems zugegriffen, um die relevanten Informationen zu extrahieren. Trigger-basierte Verfahren greifen die Änderungsinformationen hingegen direkt nach der Durchführung der Operation ab.

Write Set

Tabelle S

row ID	s1	s2	s3	New value	DML type
18	2	b	20000	false	delete
24	3	c	30000	true	insert
15	1	a	10000	false	update
15	1	b	10000	true	update

Abbildung 2: Write Set mit Änderungsoperationen für die Tabelle *S* aus Abbildung 1

Im Rahmen des ACCache-Projekts wurde der letztere Ansatz gewählt. Hierbei werden die Änderungsinformationen gesammelt und können vom WS-Manager jederzeit an die Caches propagiert werden, was die Realisierung einer globalen Transaktionkontrolle erleichtert. Ein WS, dessen Struktur in Abbildung 2 veranschaulicht ist, enthält pro geänderten Datensatz genau einen ihn eindeutig identifizierenden Eintrag. Das in Abbildung 2 dargestellte WS enthält Einträge zu einer Lösch-, einer Einfüge- und einer Aktualisierungsoperation, wobei für letztere sowohl die alten, als auch die neuen Werte gespeichert sind (hellgrau hinterlegt).

4 Propagierung der Änderungen

Nachdem die Änderungen erfasst und in Form eines WS bereitgestellt sind, ist zu klären, an welche Caches die Änderungen propagiert werden müssen. Hierbei ist der Wissensstand, den das Backend über die Caches hat, von zentraler Bedeutung. Es können folgende vier Fälle unterschieden werden, die jeweils aufeinander aufbauen [Thi07]:

- *Kein Wissen:* Dem Backend ist nur die Existenz der Caches bekannt.
- *Geringes Wissen:* Das Backend kennt die Cache-Tabellen der einzelnen Instanzen.
- *Strukturelles Wissen:* Die komplette Cache-Group-Definition ist bekannt.
- *Vollständiges Wissen:* Zusätzlich zur Cache-Group-Definition wird ein Index verwaltet, der herausfinden kann, ob ein Cache einen bestimmten Datensatz hält.

In den ersten beiden Fällen sind dem Backend kaum Informationen über die Caches bekannt. Deshalb werden Änderungen potenziell auch an Caches propagiert, welche die geänderten Sätze nicht vorhalten, was zu erhöhtem Kommunikationsaufwand führt. Die Caches können dann durch Abgleich mit ihren Constraints entscheiden, ob Änderungen zu verwerfen sind. Bei vollem Wissen kann das Backend durch einen Index der RCC-Quellspaltenwerte eine Vorselektion der an die jeweiligen Caches zu propagierenden Änderungen treffen. Danach werden nur diese ausgewählten Änderungen propagiert. Der Index muss jedoch mit den Caches und deren Lade- und Entladeverhalten konsistent gehalten werden. Aufgrund der vorangegangenen Überlegungen wurde entschieden, im ACCache-Projekt den Mittelweg zu gehen und das Backend anfänglich mit strukturellem Wissen zu versehen.

5 Übernahme der Änderungen

Nach Propagierung der Informationen an die entsprechenden Caches führen diese die Änderungsoperationen in ihrer Cache Group aus. Durch die eindeutige Identifikation der Sätze im WS können alte Werte blind mit neuen überschrieben werden. Besitzt eine Spalte einer vorgehaltenen Tabelle keine oder nur eingehende RCCs, so haben Änderungen ihrer Werte keine weiteren Implikationen für die Konsistenz des Caches.

Bei Spalten mit ausgehenden RCCs können hingegen Constraints verletzt werden, falls ein neuer Wert bisher nicht in der Spalte vorhanden war. Dieser Wert muss dann in allen Zielspalten der RCCs wertvollständig gemacht werden, was durch Nachladen der entsprechenden Sätze realisiert wird. Haben die Zielspalten selbst wieder ausgehende RCCs, kann es dabei zu kaskadierendem Laden über mehrere Tabellen hinweg kommen.

Falls ein vorher vorhandener Wert innerhalb einer Spalte nicht mehr existiert, können Werte in den Zielspalten unerreichbar werden. Der Cache kann die betroffenen Sätze daraufhin entladen. Da diese Entladevorgänge große Auswirkungen auf die Größe und Performanz des Caches haben, ist eine weiterführende Betrachtung in [KBM09] zu finden. Im Hinblick auf eine gloable Transaktionsverwaltung ist zu beachten, dass durch Lade- und Entladevorgänge der Commit-Zeitpunkt deutlich hinausgezögert werden kann. Durch eine temporäre Invalidierung von RCC-Quellspaltenwerten und eine Durchführung des Ladevorgangs nach dem Commit („lazy“) ließe sich das Commit-Verhalten beschleunigen.

In Abbildung 3 wurden die Änderungen des WS aus Abbildung 2 berücksichtigt. Da der Cache die Datensätze der Lösch- und Einfügeoperation nicht vorhält, kann er die Änderungen verwerfen. Der neue Wert b in $S_C.s2$ erzwingt auf Grund des RCC $S_C.s2 \rightarrow T_C.t2$ in dessen Zielspalte Wertvollständigkeit, weswegen der Satz (30,b) geladen wurde. Da der Wert a in der Quellspalte nicht mehr vorhanden ist, wurde er in $S_C.s2$ unerreichbar und daraufhin entladen.

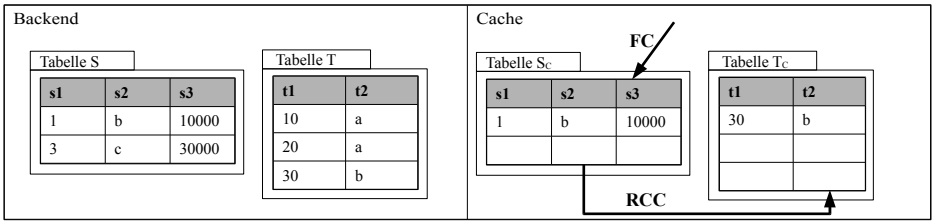

Abbildung 3: Situation in Backend und Cache nach dem Eintreffen des WS aus Abbildung 2

6 Fazit

In den vorangegangenen Abschnitten wurden wichtige Aspekte der Konsistenzsicherung beim CbDBC erläutert und deren Umsetzung im ACCache-Projekt skizziert. Die Änderungsinformationen werden im Backend abgegriffen und unter Zuhilfenahme des strukturellen Wissens über die Caches an die potenziell betroffenen Cache-Instanzen propagiert. Durch Verwendung von Write Sets wurde gezeigt, wie Nachladevorgänge die Gültigkeit von definierten Constraints in den Cache Groups aufrechterhalten können. Die hier diskutierten Verfahren ermöglichen es, eine Synchronisation zwischen Backend und Caches durchzuführen und bilden weitergehend die Basis für eine globale Transaktionskontrolle, welche die Änderungsoperationen unter Transaktionsschutz stellt.

Literatur

[BAM+04] Christof Bornhövd, Mehmet Altinel, C. Mohan, Hamid Pirahesh und Berthold Reinwald. Adaptive Database Caching with DBCache. *Data Engineering Bulletin*, 27(2):11–18, 2004.

[BHM06] Andreas Bühmann, Theo Härder und Christian Merker. A Middleware-Based Approach to Database Caching. In *Proc. ADBIS*, LNCS 4152, Seiten 182–199, Thessaloniki, 2006.

[Bra08] Susanne Braun. Implementierung und Analyse von Synchronisationsverfahren für das CbDBC. Diplomarbeit, Technische Universität Kaiserslautern, 2008.

[HB07] Theo Härder und Andreas Bühmann. Value Complete, Column Complete, Predicate Complete – Magic Words Driving the Design of Cache Groups. *VLDB Journal*, 17(2):805–826, 2007.

[KBM09] Joachim Klein, Susanne Braun und Gustavo Machado. Selektives Laden und Entladen von Prädikatsextensionen beim Constraint-basierten Datenbank-Caching. In *Proc. BTW*, 2009.

[LGZ04] Per-Åke Larson, Jonathan Goldstein und Jingren Zhou. MTCache: Transparent Mid-Tier Database Caching in SQL Server. In *Proc. ICDE*, Seiten 177–189. IEEE, 2004.

[Ora07] Oracle Corporation. Oracle Database Data Warehousing Guide, 11g Release 1. http://download.oracle.com/docs/cd/B28359_01/server.111/b28313/title.htm, 2007.

[The02] The TimesTen Team. Mid-tier Caching: The TimesTen Approach. In *Proc. SIGMOD*, Seiten 588–593, 2002.

[Thi07] Julia Thiele. Grundlegende Aktualisierungsprobleme beim Constraint-basierten Datenbank-Caching. Diplomarbeit, Technische Universität Kaiserslautern, 2007.