Dynamische Konfiguration von Cache Groups beim Constraint-basierten Datenbank-Caching

Harald Wonneberger, Joachim Klein

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

Art der Arbeit: Studienarbeit
Betreuer: Prof. Dr. Theo Härder
GI-Fachbereich: Datenbanken und Informationssysteme

Abstract: Beim Datenbank-Caching werden Teilmengen relationaler Daten eines zentralen Datenbankservers in der Nähe von Anwendungen vorgehalten. Das Constraint-basierte Datenbank-Caching (CbDBC) hält dabei in den Cache-Instanzen Satzmengen von Prädikaten vor, deren Vollständigkeit mithilfe von Constraints kontrolliert und eingehalten wird. Hierbei spielt die Konfiguration einer Cache-Instanz eine wichtige Rolle für die Leistung des Gesamtsystems. In dieser Arbeit wird gezeigt, wie die bisher statische und manuell vorzunehmende Konfiguration der Cache-Instanzen dynamisch während der Laufzeit angepasst werden kann.

1 Motivation

Das Datenbank-Caching versucht den Zugriff auf meist weit entfernte Daten zu beschleunigen, wobei häufig zugegriffene Datenelemente in der Nähe von Anwendungen bereitgestellt werden. Während Konkurrenzansätze auf der Verwendung materialisierter Sichten [LGZ04] basieren, hält das Constraint-basierte Datenbank-Caching (CbDBC) Satzmengen für einzelne Prädikate vor, deren Vollständigkeit mithilfe von Constraints kontrolliert und eingehalten wird. Somit können Cache-Inhalte dynamisch an die auftretende Anfragelast angepasst werden [BHM06], was durch das Vorhalten kompletter Tabelleninhalte (Full-Table-Caching [Ora08]) nicht gewährleistet werden kann.

Während Cache-Inhalte bereits dynamisch geladen und entladen werden, muss die Cache-Konfiguration manuell vorgenommen werden. Eine dynamische Anpassung der Konfiguration würde auf sich ändernde Anfrageschemata ohne manuellen Eingriff reagieren und dadurch versuchen, die Leistung des Gesamtsystems zu verbessern. Ziel ist es daher, das Gesamtsystem zu beobachten, um so gezielt Entscheidungen über eine Anpassung der Cache-Konfiguration treffen zu können. Da im CbDBC eine der wichtigsten Anpassungen das Hinzufügen und Entfernen der dort verwendeten referentiellen Cache Constraints

(RCCs) ist, beschränken wir uns in dieser Arbeit darauf, die Überlegungen hierzu darzustellen. Die theoretischen Grundlagen dafür werden im nächsten Kapitel kurz beschrieben.

2 Grundlagen

Beim CbDBC werden die Konfigurationen innerhalb einer sog. *Cache Group* abgelegt. Diese beschreibt die Menge einzuhaltender Constraints sowie die Cache-Tabellen. Die Definition einer Cache-Tabelle T_{C} entspricht dabei der ihr zugeordneten Backend-Tabelle T bis auf Fremdschlüsseldefinitionen, die nicht übernommen werden. Bisher werden zwei verschiedene Constraint-Typen unterschieden: der *referentielle Cache Constraint (RCC)* und die *Füllspalte (filling column, FC)*. Über einen RCC (wie z. B. $T_{\mathrm{C}}.b \rightarrow S_{\mathrm{C}}.d$, vgl. Abb. 1) wird entschieden, welche Sätze in der jeweiligen Zieltabelle (hier S_{C}) vollständig vorzuhalten sind. Mit Hilfe von FCs wird entschieden, wann Sätze in den Cache zu laden sind. Dies ist immer dann der Fall, wenn z. B. ein bisher nicht vorhandener Wert v in einer FC $T_{\mathrm{C}}.a$ durch eine Anfrage explizit referenziert wird. Ist v Element einer der Füllspalte zugeordneten Kandidatenmenge, werden die Sätze $\sigma_{a=v}T$ *vollständig* in den Cache geladen. Ein entsprechender Ladevorgang setzt sich rekursiv über ausgehende RCCs fort. Hierbei garantieren diese, dass alle geladenen RCC-Quellspaltenwerte *wertvollständig* in der RCC-Zielspalte vorgehalten werden (vgl. Abb. 1) [HB07]. Durch die Vollständigkeit der Prädikatextensionen kann so entschieden werden, welche Anfragen durch den Cache beantwortbar sind.

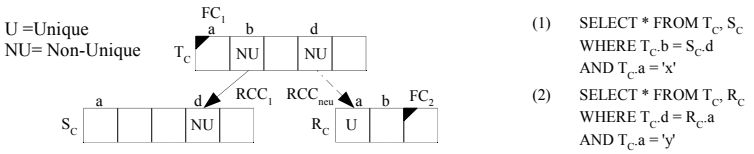

Abbildung 1: Schematische Darstellung einer Cache-Group-Konfiguration

3 Hinzufügen und Entfernen von RCCs

Hinzufügen von RCCs. In einem ersten Beispiel soll auf einer Cache-Instanz mit der bestehenden Cache Group aus Abb. 1 die Anfrage (1) ausgeführt werden. Dies ist ohne Probleme möglich, wenn der Wert x durch die FC $T_{\mathrm{C}}.a$ vorgehalten wird. Der vorzunehmende Gleichverbund $T_{\mathrm{C}}.b = S_{\mathrm{C}}.d$ wird dabei durch RCC_1 ermöglicht. Wird durch mehrfaches Anfordern der Anfrage (2) aus Abb. 1 ein Schwellwert überschritten, so könnte man entscheiden, einen neuen RCC (hier: RCC_{neu}) zur Cache Group hinzuzufügen, um so die Anfragebeantwortung zu beschleunigen. RCC_{neu} erweitert die Menge der Constraints und erzwingt so die Wertvollständigkeit in $R_{\mathrm{C}}.a$ für die in $T_{\mathrm{C}}.d$ enthaltenen RCC-Quellspaltenwerte. Diese muss jedoch zunächst durch das Nachladen fehlender Sätze her-

gestellt werden, bevor RCC_{neu} für die Anfragebeantwortung sichtbar wird. Der Ladeprozess startet dabei nicht wie bisher ausgehend von einer FC, sondern wird in einer RCC-Zielspalte für die fehlenden Quellspaltenwerte ausgelöst. Um initial zu entscheiden, welche RCC-Quellspaltenwerte wertvollständig nachzuladen sind, kann evtl. vorhandenes Wissen genutzt werden. Die nachfolgenden Überlegungen lassen auch auf den Fall übertragen, dass durch das Einfügen eines neuen RCCs Zyklen entstehen, wobei die Struktur der sog. *atomaren Zonen* entsprechend anzupassen ist [HB07]:

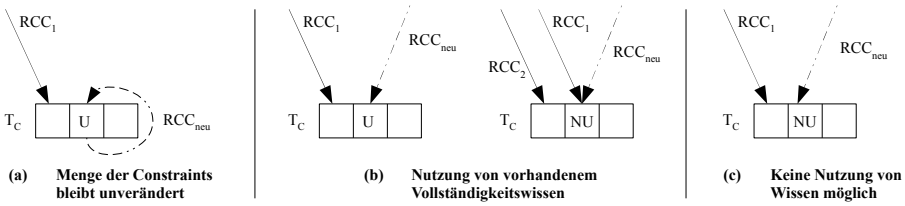

Abbildung 2: Nutzung von bestehendem Vollständigkeitwissen zur Initiierung des Nachladevorgangs

- Die Einfügung eines redundanten RCC [BH07] (wie RCC_{neu} in Abb. 2a) wird ausgeschlossen, da hierbei kein neues Vollständigkeitswissen entsteht. Somit muss auch kein Nachladevorgang ausgelöst werden, um Wertvollständigkeit zu garantieren.
- Ist die Zielspalte eines neu hinzugefügten RCC als Unique (U) markiert (RCC_{neu} in Abb. 2a), sind bereits alle vorhandenen Werte implizit wertvollständig, sodass nur noch nicht vorhandene RCC-Quellspaltenwerte nachzuladen sind. Bei Non-Unique-Spalten (NU) (vgl. Abb. 2b) lässt sich die Wertvollständigkeit nur über bereits vorhandene RCCs, die auf dieselbe Spalte verweisen, ableiten. RCC-Quellspaltenwerte, die über entsprechende RCCs bereits geladen sind, müssen somit nicht nochmals geladen werden.
- Ist die Zielspalte eines hinzugefügten RCC eine NU-Spalte und kein anderer RCC verweist ebenfalls auf dessen Zielspalte (wie bei RCC_{neu} in Abb. 2c), so kann kein bestehendes Vollständigkeitswissen genutzt werden, da nicht auf die Wertvollständigkeit einiger vorhandener Werte geschlossen werden kann. Daher müssen sämtliche Werte der Quellspalte des neu eingefügten RCC wertvollständig nachgeladen werden.

Entfernen von RCCs. Unterschreitet die Häufigkeit der Anfrage (2) aus Abb. 1 eine zuvor zu definierenden Schwellwert, so kann der RCC_{neu} wieder aus der Cache Group entfernt werden. Dadurch würden sämtliche Sätze, die durch RCC_{neu} in R_{C} wertvollständig geladen wurden, ihre Existenzberechtigung verlieren und müssten entladen werden, sofern kein anderer RCC ihre Existenz fordert. Größere Änderungen ergeben sich, wenn RCC_1 entfernt wird, da dann die Cache-Tabelle S_{C} von keinem weiteren RCC mehr referenziert wird und daher komplett entfernt werden müsste (vgl. Abb. 1). Dieser Vorgang setzt sich rekursiv durch die Cache Group fort, wodurch weitere Sätze bzw. Cache-Tabellen wegfallen können.

Sichtbarkeit. Ohne zusätzliche Maßnahmen muss ein neu hinzugefügter RCC solange invalidiert bleiben, bis alle benötigten Sätze nachgeladen sind; erst dann kann er zur Anfragebearbeitung freigegeben werden. Beim Löschen von RCCs entstehen keine Inkonsistenzen bzgl. der Anfragebearbeitung, sodass eine entsprechende Löschung sofort wirksam werden kann (vgl. Abb. 3a). Eine weiterführende Maßnahme besteht darin, nur einzelne Werte eines RCC zu invalidieren. Dies würde es ermöglichen, bereits validierte Werte als *Einstiegspunkt* zu verwenden (vgl. [HB07]). Weiterhin könnten Equi-Joins durchgeführt werden, solange ein solcher Join keine invalidierten Werte miteinander verbindet (vgl. Abb. 3b). Eine entsprechende Evaluation des Joins ist jedoch ohne spezielle Maßnahmen schwierig [Bra08].

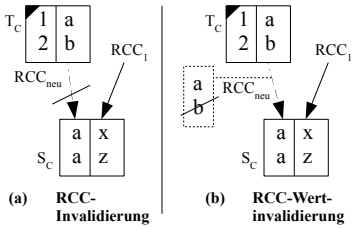

Abbildung 3: RCC Invalidierung und RCC-Wertinvalidierung

4 Fazit

In dieser Arbeit wurde dargestellt, wie eine dynamische Anpassung von Cache Groups grundsätzlich ermöglicht werden kann, wodurch sich weitere Vorteile zur Optimierung der Anfragebeantwortung bieten. Neben RCCs müssen jedoch noch detaillierte Analysen für die Anpassung von Füllspalten, Werte der Kandidatenlisten sowie Cache-Tabellen durchgeführt werden. Eine darüber hinaus wichtige Frage betrifft die Auswirkungen, die sich auf andere Komponenten des CbDBC-Systems wie z. B. der Transaktionsverwaltung ergeben. Die Freigabe von Anpassungen der Cache Group für die Anfragebeantwortung (z. B. Freischalten eines RCCs) darf die bei der Transaktionsverwaltung vorgesehene Snapshot-Isolation nicht verletzen [Bra08]. Völlig offen ist jedoch noch, anhand welcher statistisch zu ermittelnder Schwellwerte, Entscheidungen über Anpassungen an einer Cache Group getroffen werden können.

Literatur

[BH07] Andreas Bühmann und Theo Härder. Making the Most of Cache Groups. In Kotagiri Ramamohanarao, P. Radha Krishna, Mukesh K. Mohania und Ekawit Nantajeewarawat, Hrsg., *DASFAA*, Lecture Notes in Computer Science, Seiten 349–360. Springer, 2007.

[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. *VLDB Journal*, 17(2):805–826, 2007.

[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.

[Ora08] Oracle Corporation. Internet Application Server Documentation Library, 2008.