Langbeitrag für GI-Workshop "Datenbanken und Betriebssysteme", Heidelberg, April 1993,
in: DB-Rundbrief Nr. 11 (Mai 1993) der GI-Fachgruppe 2.5 (Datenbanksysteme)

Kohärenzkontrolle in verteilten Systemen

Erhard Rahm

Fachbereich Informatik, Universität Kaiserslautern
E-Mail: rahm@informatik.uni-kl.de

1. Einleitung

Die replizierte Pufferung von Datenobjekten in mehreren Prozessoren bzw. Rechnern führt zum Problem der Kohärenzkontrolle. Diese Problematik wurde zuerst im Bereich von Multiprozessoren bezüglich der Pufferung innerhalb von Prozessor-Caches untersucht [St90]. Im Datenbank- und Betriebssystembereich werden seit ca. 10 Jahren ähnliche Kohärenzprobleme hinsichtlich Pufferung im Hauptspeicher betrachtet, z.B. im Rahmen von Shared-Disk-Datenbanksystemen (DB-Sharing), Workstation-/Server-DBS, Netzwerk-Dateisystemen oder DSM-Systemen (Distributed Shared Memory). Trotz großer Ähnlichkeit in der Problemstellung wurden jedoch in jedem dieser Bereiche die Lösungen zur Kohärenzkontrolle weitgehend isoliert entwickelt, was dazu führte, daß sehr ähnliche Ansätze oft mehrfach "neu erfunden" wurden. Dieser Beitrag versucht, wesentliche Gemeinsamkeiten und Unterschiede bei der Kohärenzkontrolle in den verschiedenen Bereichen herauszustellen, um eine bessere gegenseitige Verständigung zu fördern. Eine stärkere gegenseitige Berücksichtigung der Forschungsergebnisse führt idealerweise auch zur Verbesserung verschiedener Lösungen, indem man z.B. für einen Bereich wirksame Optimierungen auf einen anderen Bereich überträgt.

Im nächsten Kapitel präzisieren wir zunächst die Aufgabenstellung der Kohärenzkontrolle und geben einen Überblick über die wichtigsten Lösungsansätze. Die angesprochenen Lösungsmöglichkeiten sind dabei in mehreren (bzw. allen) der angesprochenen Bereiche einsetzbar, wenngleich sie teilweise unabhängig voneinander (unter verschiedenen Bezeichnungen) vorgeschlagen wurden. In Kap. 3 und 4 gehen wir dann näher auf die einzelnen DBS- bzw. BS-Bereiche ein, in denen Kohärenzprotokolle untersucht werden, um Besonderheiten bzw. Unterschiede aufzuzeigen.

2. Kohärenzkontrolle: Generelle Aufgaben und Lösungsalternativen

Die Pufferung (Caching) sowie die replizierte Speicherung von Datenobjekten sind zwei weitverbreitete Techniken zur Leistungssteigerung in Computer-Systemen. Innerhalb von Speicherhierarchien erlaubt die Pufferung von Objekten in einem schnellen Speicher durch Nutzung von Lokalität die Anzahl von Zugriffen auf langsamere Speicher zu reduzieren; durch die Pufferung kommt es zu einer **vertikalen Replikation** von Objekten über zwei oder mehr Speicherebenen. In verteilten Systemen kann durch replizierte Speicherung von Datenobjekten auf einer bestimmten Ebene der Speicherhierarchie (**horizontale Replikation**) Kommunikation mit anderen Rechnern eingespart werden. Für beide Arten der Replikation führen Änderungen zu Konsistenz- bzw. Kohärenzproblemen, da sie zur Veralterung bzw. Invalidierung der betreffenden Datenobjekte auf tieferen Speicherebenen (bei vertikaler Replikation) bzw. in anderen Rechnern (bei horizontaler Replikation) führen.
Unser Augenmerk hier gilt vor allem der Hauptspeicherpufferung von Objekten (i.a. Seiten fester Größe) innerhalb von Datenbanksystemen (DBS) und Betriebssystemen (Dateiverwaltung). Bei

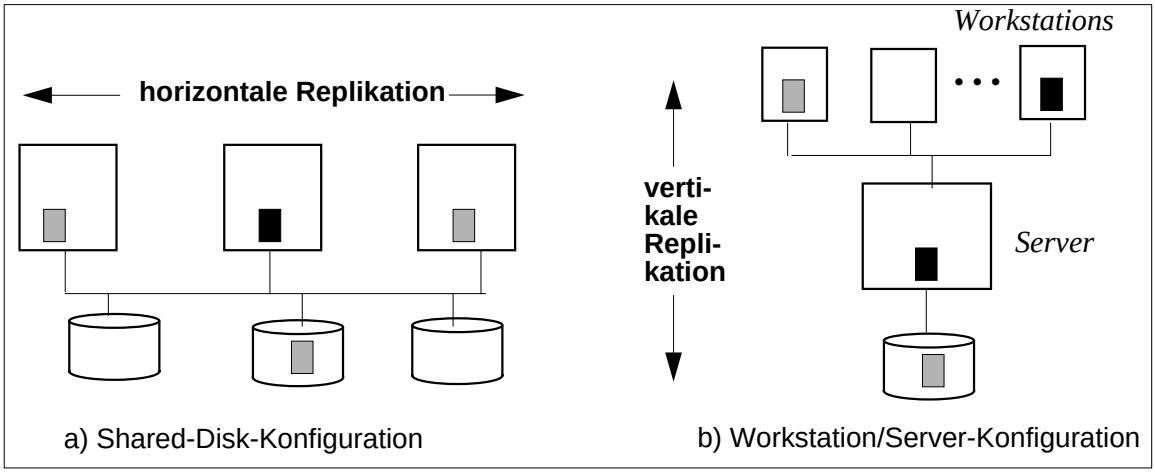

Bild 1: Kohärenzprobleme durch vertikale und horizontale Replikation

der Nutzung verteilter Systeme liegt dabei i.a. sowohl eine vertikale als auch eine horizontale Replikation von Objekten vor (Bild 1). Im Falle von Workstation/Server-Konfigurationen (Bild 1b) besteht eine horizontale Replikation bezüglich der Workstations, da dasselbe Objekt in mehreren Workstations gepuffert sein kann. Der Server (-Hauptspeicher) entspricht quasi einer weiteren Schicht in der Speicherhierarchie zwischen Workstation-Hauptspeicher und den (Server-)Platten, so daß zwischen Workstations und (DB-/Datei-) Server eine vertikale Replikation von Objekten vorliegt.

Die Kohärenzkontrolle umfaßt zwei wesentliche Teilaufgaben:

1. Der Zugriff auf invalidierte Objekte muß verhindert werden und
2. Propagierung von Änderungen, um die aktuelle Objektversionen für den Zugriff bereitzustellen.

Zur Lösung des ersten Teilproblems können Vermeidungs- oder Erkennungsstrategien eingesetzt werden, bei denen die Entstehung von Invalidierungen von vorneherein vermieden wird bzw. bei denen invalidierte Objekte vor dem Zugriff erkannt und ggf. aus dem Puffer entfernt werden. Eine Vermeidung von Pufferinvalidierungen wird z.B. durch Einschränkung der Pufferung auf nur lesbare Objekte erreicht oder indem man gepufferte Objekte durch Zusatzmechanismen wie Sperren vor einer Invalidierung schützt. Ein einfacher Erkennungsansatz besteht in einer *Broadcast* (bzw. *Multicast)-Invalidierung*, bei der jede Änderung allen Rechnern (bzw. allen Rechnern, die eine Kopie des Objektes gepuffert haben) gemeldet wird, damit veraltete Kopien aus den Puffern eliminiert werden können. Eine meist effizientere Alternative (weniger Kommunikationsaufwand) verwendet Zeitstempel oder Änderungszähler an den Objekten, um festzustellen, ob eine bestimmte Version aktuell oder veraltet ist.

Aufgrund der vertikalen und horizontalen Replikation ergibt sich die Notwendigkeit sowohl einer vertikalen als auch horizontalen Propagierung von Änderungen. Bezüglich der **vertikalen Änderungspropagierung** sind vor allem die beiden in Speicherhierarchien üblichen Alternativen zum Schreiben geänderter Objekte zu unterscheiden: Write-Through vs. Write-Back (sofortiges vs. verzögertes Durchschreiben von Änderungen). Im Datenbankbereich beziehen sich die Schreibvorgänge auf Transaktionen; hier spricht man von FORCE (write-through) vs. NOFORCE (write-back). Der Write-Through-Ansatz (FORCE) weist i.a. eine geringere Leistungsfähigkeit als Write-Back (NOFORCE) auf, dafür sind jedoch die Daten auf der Ebene, in die das Durchschreiben erfolgt (z.B. Platte), stets auf dem aktuellen Stand. Bei Write-Through entfällt

daher auch die Notwendigkeit einer horizontalen Propagierung von Änderungen, falls alle Rechner auf die "nächsttiefere Speicherebene" zugreifen können, welche die aktuellen Objektversionen hält. Dies ist der Fall bei Shared-Disk-DBS sowie Workstation/Server-Systemen (Bild 1).

Bezüglich der **horizontalen Propagierung** von Änderungen besteht ein extremer Ansatz in einem *Write-Broadcast* (bzw. *Write-Multicast*), bei dem jede Änderung an alle (kopienhaltenden) Rechner geschickt wird, um die Kopien sofort zu aktualisieren. Dieser Ansatz scheidet jedoch aufgrund des hohen Kommunikationsaufwandes in der Regel aus, vor allem bei höherer Rechneranzahl. Die Alternative sieht vor, daß pro Objekt einer der Rechner als "Owner" fungiert, der andere Rechner mit der aktuellen Version des Objektes versorgt. Die Owner-Funktion kann für alle Objekte einem Rechner zugewiesen werden, was jedoch leicht zu einem Leistungsengpaß (sowie zu Verfügbarkeitsproblemen) führen kann. Daher ist i.a. eine verteilte Owner-Zuordnung vorzusehen, die dynamisch oder fest erfolgen kann. Bei der dynamischen Owner-Zuordnung wird i.a. der Rechner, der das Objekt ändert, zum Owner des Objektes. Damit besitzt der Owner (per Definition) ohne Kommunikationsverzögerung stets die aktuelle Objektversion. Kommunikation ist erforderlich, um den aktuellen Owner zu bestimmen und um das Objekt sowie ggf. die Owner-Funktion zu transferieren. Eine feste Owner-Zuordnung unter mehrere Rechner (z.B. über eine Hash-Funktion) vermeidet Kommunikation zur Bestimmung des aktuellen Owners sowie für Owner-Transfers; dafür sind Änderungen an den Owner zu übertragen, falls sie in einem anderen Rechner durchgeführt wurden.

3. Kohärenzkontrolle im Datenbankbereich

3.1 Shared-Disk-DBS

Hierbei handelt es sich um homogene Mehrrechner-DBS, bei denen in jedem Rechner eine identische Kopie des DBVS (Datenbankverwaltungssystem) läuft und jedes DBVS direkt auf alle Externspeicher (die gesamte Datenbank) zugreifen kann. Die Rechner sind i.a. in räumlicher Nachbarschaft angeordnet und über ein Hochgeschwindigkeitsnetzwerk verbunden. Der Shared-Disk-Ansatz (SD, DB-Sharing) wird vor allem im Hinblick auf die Realisierung von Hochleistungs-DBS verfolgt, welche hohe Transaktionsraten, kurze Antwortzeiten, hohe Verfügbarkeit, modulare Erweiterbarkeit und hohe Kosteneffektivität anstreben. Kohärenzkontrolle wird notwendig, da jedes DBVS Seiten der gemeinsamen DB in seinem Hauptspeicher puffert, um Externspeicherzugriffe einzusparen. Neben der globalen Synchronisation, die i.a. über ein Sperrprotokoll realisiert wird, kommt der Kohärenzkontrolle eine leistungsbestimmende Rolle zu, da diese beiden Funktionen den Kommunikationsbedarf zur Transaktionsbearbeitung weitgehend bestimmen. Bezüglich der Kohärenzkontrolle bestehen hohe Konsistenzforderungen: i.a. wird verlangt, daß jeder Zugriff auf die aktuellste, transaktionskonsistente Objektversion erfolgt.

Zur Kohärenzkontrolle bei SD konnten sehr effiziente Verfahren entwickelt werden, da sich diese Funktion nahezu vollständig in das globale Sperrprotokoll integrieren läßt. Bei einer Sperrverwaltung auf Seitenebene können so zusätzliche Nachrichten zur Kohärenzkontrolle weitgehend vermieden werden. Dies gilt vor allem für die Erkennung von Pufferinvalidierungen, die im Rahmen eines sogenannten *On-Request-Invalidierung*sverfahren stets zusammen mit der Bearbeitung von Sperranforderungen möglich ist, in dem die globalen Sperrinformationen entsprechend erweitert werden (Änderungszähler, Invalidierungsvektoren, u.ä. [Ra86]). Auch der Vermeidungsansatz läßt sich effizient nutzen, in dem gepufferte Seiten durch spezielle Haltesperren (retention locks) vor einer Invalidierung geschützt werden. Diese Sperren haben zudem den Vorteil,

daß sie eine lokale Sperrbehandlung unterstützen und somit den Kommunikationsbedarf zur globalen Synchronisation reduzieren [Ra91].

Die horizontale Propagierung von Änderungen ist nur bei NOFORCE erforderlich; die Objekte können hierzu direkt über das Kommunikationsnetz übertragen werden oder aber auch über die gemeinsamen Externspeicher (Platten oder gemeinsame Zwischenspeicher). Letzterer Ansatz ist zwar i.a. langsamer, erfordert i.a. jedoch einen geringeren CPU-Aufwand und bringt Recovery-Vorteile mit sich. Bei einer dynamischen (Page-)Owner-Zuordnung kann der aktuelle Owner zu einem Objekt (Seite) durch Erweiterung der globalen Sperrtabelle bei der Sperranforderung ohne weitere Kommunikation ermittelt werden; die Anforderung einer Seite selbst führt jedoch zu einer zusätzlichen Verzögerung. Selbst diese Nachrichten zur Anforderung von Seiten beim aktuellen Owner lassen sich bei einer festen Owner-Zuordnung vermeiden, wenn auch die globale Sperrverantwortung in gleicher Weise unter die Rechner aufgeteilt wird. Denn in diesem Fall können Sperr- und Seitenanforderungen kombiniert werden; der Owner-Rechner überträgt die jeweilige Seite zusammen mit der Gewährung einer Sperre. Änderungen werden an den Owner übertragen, was mit der Freigabe der für die Änderung erforderlichen Schreibsperre kombiniert werden kann.

Die wichtigsten Ansätze zur Kohärenzkontrolle bei SD waren etwa 1985/86 bekannt und publiziert. Eine Übersicht findet sich in [Ra89, Ra91].

3.2 Workstation/Server-DBS

In Workstation/Server-DBS erfolgt eine Aufteilung der DBVS-Funktionalität zwischen Workstation-DBVS und Server-DBVS [DFMV90]. Solche Architekturen werden v.a. im Kontext objekt-orientierter DBS bzw. Non-Standard-DBS eingesetzt, um die Grafikeigenschaften sowie Verarbeitungs- und Hauptspeicherkapazität leistungsfähiger Workstations für komplexe DB-Anwendungen zu nutzen. Das Server-DBVS verwaltet die Externspeicher und realisiert globale Dienste wie Synchronisation und Logging. Durch Pufferung von DB-Objekten in den Workstations ("Objektpuffer") kann die Kommunikationshäufigkeit mit dem Server signifikant reduziert werden; es wird jedoch eine Kohärenzkontrolle erforderlich (Bild 1b).

Die vorgeschlagenen Ansätze zur Kohärenzkontrolle in Workstation/Server-DBS sind weitgehend identisch mit denen für SD, wenngleich die bereits vorliegenden SD-Protokolle meist ignoriert wurden. Es wird i.d.R. unterstellt, daß die globale Sperrtabelle des Server-DBVS mit Angaben zur Kohärenzkontrolle erweitert wird. Die Erkennung von Pufferinvalidierungen nach einem On-Request-Invalidierungsansatz erfolgt z.B. in Orion [KGBW90], während ObjectStore einen Vermeidungsansatz über Haltesperren ("callback locks" genannt) verfolgt [LLOW91]. Diese beiden Ansätze wurden auch in einigen weiteren Veröffentlichungen vorgeschlagen bzw. analysiert (z.B. [WN90, CFLS91, WR91, FCL92]). Meist erfolgt nur eine vertikale Propagierung von Änderungen im Rahmen eines FORCE-ähnlichen Ansatzes, bei dem am Transaktionsende sämtliche Änderungen von der Workstation zum Server übertragen werden. Die aktuelle Version eines Objektes kann daher stets beim Server angefordert werden (i.a. zusammen mit der Sperre). Eine begrenzte Form der horizontalen Propagierung von Objekten wurde in [FCL92] vorgeschlagen. Dabei wird ein Objekt jedoch weiterhin stets beim Server angefordert; nur wenn dieser das Objekt nicht im Hauptspeicher gepuffert hat, jedoch eine der Workstations, wird das betreffende Workstation-DBVS beauftragt, das Objekt direkt an die anfordernde Workstation zu übertragen. Damit soll eine kürzere Verzögerung als durch einen Plattenzugriff beim Server erreicht werden.

Die derzeitigen Arbeiten zur Kohärenzkontrolle in Workstation/Server-DBS unterstellen das herkömmliche Transaktionskonzept, obwohl dieses für Non-Standard-Anwendungen ungeeignet ist. In [ABG90] wurde vorgeschlagen, für in Workstations gepufferte Objekte schwächere Konsistenzbedingungen zuzulassen, um den Aufwand zur Kohärenzkontrolle zu reduzieren. Dabei wurden jedoch Information-Retrieval-Anwendungen unterstellt, bei denen nur Lesezugriffe auf gepufferte Objekte erfolgen und für die es oft ausreicht, die Kopien nur in definierbaren Zeitabständen zu aktualisieren ("quasi copies").

3.3 Verteilte DBS

Ein der Kohärenzkontrolle verwandtes Problem ist das der Wartung replizierter Datenbanken. Dabei erfolgt jedoch die horizontale Replikation auf Ebene der Externspeicher (Platten), also nicht auf Ebene eines Pufferspeichers. Diese Form der Replikation ist somit auch statisch (Anzahl und Ort der Replikate sind fest vorbestimmt); neben der Unterstützung schneller Lesezugriffe steht vor allem eine Erhöhung der Verfügbarkeit im Vordergrund. Die Ansätze zur Wartung replizierter Datenbanken [GA87] sind daher auch nur bedingt auf die Problemstellung der Kohärenzkontrolle übertragbar. Zum Beispiel wird bzgl. der Aktualisierung replizierter Datenbanken i.a. verlangt, daß Änderungen auf alle Kopien propagiert werden, was einem Write-Broad(-Multi-)cast gleichkommt. Dies ist äußerst aufwendig (selbst bei asynchroner Propagierung) und wird deswegen im Rahmen der Kohärenzkontrolle i.a. nicht verfolgt.

Ähnliche Kohärenzprobleme wie etwa bei SD ergeben sich auch bei verteilten DBS (bzw. sogenannten Shared-Nothing-Systemen), wenn ein Rechner zur Einsparung von Kommunikation Daten eines anderen Rechners (einer externen DB-Partition) zur späteren Weiterverarbeitung puffert. Ein solcher Ansatz ("data shipping") wird jedoch meist nicht verfolgt, sondern stattdessen werden Operationen i.a. bei den datenhaltenden Rechnern ausgeführt ("function shipping"). Eine externe Pufferung wird jedoch zum Teil für Katalogdaten vorgesehen, wobei Invalidierungen meist über Zeitstempelvergleich erkannt werden. Der Owner der aktuellen (Katalog-)Daten ist durch die Datenverteilung fest bestimmt.

4. Betriebssystembereich

4.1 Netzwerk-Dateisysteme

Netzwerk-Dateisysteme bieten Benutzern innerhalb lokaler Netzwerke einen ortstransparenten Zugriff auf gemeinsame Dateien. Üblicherweise liegt eine Workstation/Server-Konfiguration vor, in der die Dateien von einem oder mehreren Datei-Servern verwaltet werden. Zur Reduzierung des Kommunikationsaufwandes mit Datei-Servern erfolgt eine Pufferung von Seiten (Sun NFS, Sprite,...) oder ganzer Dateien (Andrew) innerhalb der Workstations.

Ein wesentlicher Unterschied zum DB-Bereich liegt darin, daß Netzwerk-Dateisysteme i.a. keine Transaktionen unterstützen und damit auch nicht entsprechende Synchronisations- und Recovery-Protokolle. Die Konsistenzforderungen bezüglich Kohärenzkontrolle beziehen sich daher auch nicht auf Transaktionen, sondern Dateioperationen (Open/Close, Read/Write). Ein wünschenswertes Konsistenzkriterium ist, daß jede Leseoperation auf eine Datei sämtliche zeitlich vorhergehenden Änderungen sieht. Dies wird z.B. von Sprite unterstützt [NWO88], wobei jedoch eine Datei während einer Änderung nicht für parallele Lesezugriffe in anderen Rechnern gepuffert werden darf.

Einen Überblick zur Kohärenzkontrolle in Netzwerk-Dateisystemen findet man in [LS90]. Zur Erkennung invalidierter Seiten/Dateien kann zwischen Client- und Server-initiierenden Ansät-

zen unterschieden werden. Beim Client-initiierten Ansatz prüft der Client (das Workstation-Betriebssystem) vor dem Zugriff beim Server, ob ein gepuffertes Objekt noch gültig ist. Dies korrespondiert zu einem On-Request-Invalidierungsansatz, erfordert hier jedoch eigene Nachrichten für die Gültigkeitsprüfung. Im Server-initiierten Ansatz führt der Server darüber Buch, wo welche Objekte gepuffert sind, um im Falle einer Änderung eine Invalidierung zu veranlassen; dies entspricht einem Multicast-Invalidierungsansatz. Im Andrew-Dateisystem wird letzterer Ansatz in Kombination mit einem Callback-Mechanismus verwendet, der dem Einsatz von Haltesperren ähnelt [Ho88].

Zur vertikalen Propagierung von Änderungen[1] verwenden Netzwerk-Dateisysteme entweder eine Write-Through, Write-Back (delayed-write) oder Write-on-Close-Strategie [LS90]. Die erste und dritte Strategie korrespondieren grob zu einem FORCE-Ansatz bezogen auf jede Schreiboperation (Write-Through) bzw. bezogen auf die Dauer einer Dateibearbeitung (Write-on-Close), wobei Änderungen zu den jeweiligen Zeitpunkten zum Server übertragen werden. Beim Write-Back-Ansatz werden Änderungen erst nach Ablauf einer bestimmten Zeitspanne oder spätestens bei Ersetzung aus dem Workstation-Puffer zum Server übertragen, was bei mehrfacher Änderung desselben Objektes Kommunikationseinsparungen erlaubt. Bei Write-on-Close und Write-Back führt ein Workstation-Ausfall zum Verlust von Änderungen, die noch nicht zum Server übertragen werden konnten.

4.2 Distributed Shared Memory (DSM)

DSM-Systeme unterstützen die Abstraktion eines gemeinsamen Speichers in verteilten Systemen [NL91]. Durch Nutzung globaler Datenstrukturen im (logisch) gemeinsamen Speicher sollen verteilte und parallele Anwendungen einfacher erstellt werden können als durch Verwendung expliziter Operationen zum Nachrichtenaustausch (message passing). Das Betriebssystem implementiert die Abstraktion des gemeinsamen Speichers auf Basis physisch verteilter Speicher, so daß eine logische Speicherreferenz einen Kommunikationsvorgang mit einem anderen Rechner verursachen kann, um das entsprechende Objekt zu erreichen. Eine Hauptspeicherpufferung referenzierter Objekte (Seiten) soll die Kommunikationshäufigkeit reduzieren, verlangt jedoch eine Kohärenzkontrolle.

Ein wesentlicher Unterschied zur Kohärenzkontrolle im DB-Bereich liegt wiederum darin, daß DSM-Systeme üblicherweise keine Transaktionen unterstützen[2]. Die meisten DSM-Systeme verlangen eine sogenannte strikte Speicherkonsistenz, wobei jede lesende Speicherreferenz den zuletzt im System geschriebenen Wert zurückliefert. Zur Leistungsverbesserung werden teilweise aber auch schwächere Konsistenzzusicherungen unter Nutzung der Anwendungssemantik vorgesehen [NL91]. Allerdings muß der Anwendungsprogrammierer das jeweilige Konsistenzmodell kennen und berücksichtigen, um korrekte Anwendungen zu erstellen.

Einen Überblick zu vorgeschlagenen bzw. realisierten Ansätzen zur Kohärenzkontrolle in DSM-Systemen bieten [He90, SZ90, NL91]. Üblicherweise wird nur eine horizontale Migration von Objekten zwischen Rechnern betrachtet, wobei die Owner-Funktion entweder einem Rechner allein zugeordnet ist oder dynamisch bzw. fest verteilt wird [LH89]. Pufferinvalidierungen werden meist durch einen "Write-Invalidate"-Ansatz erkannt, wobei Kopien nach jeder Änderung explizit invalidiert werden. Dies entspricht wiederum einem Broadcast- bzw. Multicast-Invalidie-

1. Eine horizontale Propagierung von Objekten wird unseres Wissens derzeit nicht unterstützt.
2. Wird das Transaktionskonzept unterstützt wie z.B. in [BHT90], können die Verfahren aus dem DB-Bereich nahezu unverändert übernommen werden.

rungsansatz. Der Owner eines Objektes vermerkt dazu üblicherweise, an welchen Rechnern Kopien vorliegen, um eine Invalidierung vorzunehmen.

Im verteilten Betriebssystem Clouds wurde eine Kombination von Kohärenzkontrolle und Synchronisation vorgesehen, um die Nachrichten zur Invalidierung von veralteten Kopien zu vermeiden [RAK89]. Dabei erfolgt die Synchronisation auf einem Objekt durch den jeweiligen Owner, so daß Objekttransfers mit Nachrichten zur Sperrgewährung und -freigabe kombiniert werden können, wie bereits zuvor im DB-Bereich vorgeschlagen wurde. Der Clouds-Ansatz hat jedoch den Nachteil, daß Objekte mit Freigabe der Sperre aus dem Puffer entfernt werden müssen, was jedoch die Nutzung von Lokalität erheblich einschränkt. Eine naheliegende Verbesserung wäre der Einsatz eines On-Request-Invalidierungsansatzes, bei dem bei der Sperranforderung die Gültigkeit einer Kopie geprüft wird.

5. Ausblick

Es wurde gezeigt, daß zur Kohärenzkontrolle im Datenbank- und Betriebssystembereich sehr ähnliche Lösungsmöglichkeiten bestehen, dies in vorliegenden Forschungsarbeiten jedoch meist nicht beachtet wurde. Ein wesentlicher Unterschied liegt darin, daß im DB-Bereich das Transaktionskonzept unterstützt wird, wodurch auch die Konsistenzforderungen an die Kohärenzkontrolle bestimmt sind. Im Betriebssystembereich dagegen fehlt derzeit meist die Transaktionsunterstützung, so daß sich andere (jedoch verwandte) Konsistenzforderungen ergeben, die an die jeweiligen Verarbeitungseinheiten angelehnt sind (z.B. Datei- oder Seitenzugriffe in Netzwerk-Dateisystemen, Speicherreferenzen in DSM-Systemen). Soll, wie vielfach wünschenswert, bereits eine Transaktionsunterstützung durch das Betriebssystem erfolgen, dann können für den DB-Bereich bereits vorgeschlagene Optimierungen zur Kohärenzkontrolle unmittelbar übernommen werden. Insbesondere kann eine weitgehende Integration in ein Synchronisationsprotokoll vorgenommen werden, um Nachrichten einzusparen. Auch bezüglich Recovery ergeben sich Auswirkungen auf die Kohärenzkontrolle (Propagierung von Änderungen), die im DB-Bereich bereits untersucht wurden [Ra91].

Die meisten Vorschläge zur Kohärenzkontrolle unterstellen Seiten fester Größe als Einheiten der Pufferung und Kohärenzkontrolle. Dies ist vielfach sinnvoll, da für Seiten eine einfache und effiziente Pufferverwaltung realisierbar ist und Seiten die Zugriffseinheiten zu Externspeichern darstellen. Eine integrierte Behandlung von Kohärenzkontrolle und Synchronisation verlangt damit jedoch auch eine Sperrvergabe auf Seitenebene, was für einige Objekttypen eine unzulässig hohe Konfliktrate verursachen kann. Verfahrenserweiterungen für feinere Sperrgranulate sind bereits vorgeschlagen worden, können jedoch wiederum vermehrte Kommunikation mit sich bringen [Ra91][1]. Auch in objekt-orientierten DBS und DSM-Systemen sind anwendungsbezogene Puffergranulate oft wünschenswert, um unnötige Interferenzen mit anderen Transaktionen bzw. Anwendungsvorgängen zu reduzieren.

Die Kohärenzkontrolle kann die Skalierbarkeit auf eine große Anzahl von Rechnern beeinträchtigen, da die Anzahl der Invalidierungen und Objekttransfers potentiell mehr als linear mit der Rechneranzahl zunimmt. Zudem steigt der Aufwand einiger Kohärenzprotokolle (z.B. Broadcast-/Multicast-Invalidierung) mehr als linear mit der Rechneranzahl. Letzterer Nachteil läßt sich durch optimierte Protokolle umgehen, insbesondere bei Integration in das Sperrverfahren (sofern

1. Die Unterstützung größerer Puffer- und Sperrgranulate ist relativ einfach (-> hierarchisches Sperrverfahren) und kann i.a. zur Kommunikationseinsparung genutzt werden.

der Kommunikationsaufwand zur Synchronisation begrenzt werden kann). Der Umfang an Invalidierungen hängt daneben natürlich von der Änderungshäufigkeit ab und inwieweit Zugriffe auf dieselben Objekte an verschiedenen Rechnern erfolgen. Im DB-Bereich ist es z.B. vielfach möglich, aufgrund bekannter Referenzmerkmale von Transaktionstypen die Last so unter die Rechner aufzuteilen, daß eine hohe rechnerspezifische Lokalität erreicht wird, so daß sich auch die Anzahl an Pufferinvalidierungen entscheidend reduziert (affinitätsbasierte Lastverteilung). In Netzwerk-Dateisystemen oder Workstation-/Server-DBS ist ohnehin mit einer moderaten Invalidierungshäufigkeit zu rechnen, da der (nahezu) gleichzeitige Zugriff auf dieselben Dateien/Objekte durch eine große Anzahl von Benutzern eher die Ausnahme sein dürfte. In DSM-Systemen wurde versucht, durch Abschwächung der Konsistenzzusicherungen die Skalierbarkeit zu verbessern.

Literatur

ABG90 Alonso, R., Barbara, D., Garcia-Molina, H. : Data Caching Issues in an Information Retrieval System. *ACM Trans. Database Systems 15*, 3, 359-384, 1990

BHT90 Bellew, M., Hsu, M., Tam, V.: Update Propagation in Distributed Memory Hierarchies. *Proc. 6th Int. Conf. on Data Engineering*, IEEE Computer Society Press, 521-528, 1990

CFLS91 Carey, M.J., Franklin, M.J., Livny, M., Shekita, E.J.: Data Caching Tradeoffs in Client-Server DBMS Architectures. *Proc. ACM SIGMOD Conf.*, 357-366, 1991.

DFMV90 DeWitt, D.J., Futtersack, P., Maier, D., Velez, F.: A Study of Three Alternative Workstation-Server Architectures for Object Oriented Database Systems. *Proc. of the 16th Int. Conf. on Very Large Data Bases*, 107-121, 1990.

FCL92 Franklin, M.J., Carey, M.J.,Livny, M.: Global Memory Management in Client-Server DBMS Architectures. *Proc. of the 18th Int. Conf. on Very Large Data Bases*, 1992.

GA87 Garcia-Molina, H., Abbott, R.K.: Reliable Distributed Database Management. *Proc. of the IEEE 75*, 5, 601-620, 1987.

He90 Hellwagner, H.:A Survey of Virtually Shared Memory Schemes. TUM-I9056, TU München, 1990

Ho88 Howard, J.H. et al.: Scale and Performance in a Distributed File System. *ACM Trans. Comp. Systems 6*, 1, 51-81, 1988

KGBW90 Kim, W., Garza, J.F., Ballou, N. Woelk, D.: Architecture of the Orion Next-Generation Database system. *IEEE Trans. Knowledge and Data Engineering 2* ,1, 109-124, 1990

LLOW91 Lamb, C., Landis, G., Orenstein, J., Weinreb, D.: The ObjectStore Database System. *CACM 34*, 10, 50-63, Oct. 1991.

LS90 Levy, E., Silberschatz, A.: Distributed File Systems: Concepts and Examples. *ACM Comput. Surv. 22*, 4 (Dec.), 321-374, 1990.

LH89 Li, K., Hudak, P..: Memory Coherence in Shared Virtual Memory Systems. *ACM Trans. Comp. Systems 7*, 4, 321-359, 1989

NL91 Nitzberg, B., Lo, V.: Distributed Shared Memory: a Survey of Issues and Algorithms. *IEEE Computer* Aug. 1991.

NWO88 Nelson, M.N., Welch, B.B., Ousterhout, J.K.: Caching in the Sprite Network File System. *ACM Trans. Comp. Systems 6*, 1,134-154, 1988

Ra86 Rahm, E.: Primary Copy Synchronization for DB-Sharing. *Information Systems 11*, 4, 275-286, 1986

Ra89 Rahm, E: Der Database-Sharing-Ansatz zur Realisierung von Hochleistungs-Transaktionssystemen. *Informatik-Spektrum* 12 (2), 65-81, 1989

Ra91 Rahm, E.: Concurrency and Coherency Control in Database Sharing Systems, ZRI-Bericht 3/91, Univ. Kaiserslautern, Dec. 1991

RAK89 Ramachandran, U., Ahamad, M., Khalidi, M.Y.A.: Coherence of Distributed Shared Memory: Unifying Synchronization and Data Transfer. *Proc. Int. Conf. on Parallel Processing*, Vol. II, 160-169, 1989.

St90 Stenström, P.: A Survey of Cache Coherence Schemes for Multiprocessors. *IEEE Computer*, 12-24, Juni 1990

SZ90 Stumm, M., Zhou, S. : Algorithms Implementing Distributed Shared Memory. *IEEE Computer* (May), 54-64, 1990

WR91 Wang, Y., Rowe, L.A.: Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture. *Proc. ACM SIGMOD Conference* (Boulder, May), 367-376, 1991

WN90 Wilkinson, K., Neimat, M.: Maintaining Consistency of Client-cached Data. *Proc. 16th Int. Conf. on Very Large Data Bases*, 122-133, 1990