Technischer Bericht, University of Kaiserslautern, 1988

Einsatzformen eines Global Storage bei DB-Sharing

Theo Härder, Erhard Rahm

Juni 1988

Abstract:

Der Global Storage (GS) stellt einen nichtflüchtigen Halbleiterspeicher mit zwei Zugriffsgranulaten (Block, Eintrag) und Zugriffszeiten im unteren Mikrosekundenbereich dar, der eine nahe Rechnerkopplung mehrerer autonomer Rechner gestattet. Untersucht wird, wie mit einem solchen Speicher die Leistungsfähigkeit eines DB-Sharing-Systems erhöht werden kann, v.a. durch eine effizientere Kooperation und Kommunikation der Rechner sowie durch ein optimiertes E/A-Verhalten. Im Mittelpunkt stehen dazu Einsatzmöglichkeiten des GS zur globalen Synchronisation und Behandlung von Pufferinvalidierungen sowie eine Verwendung als globaler Systempuffer bzw. für Logging-Zwecke. Aufgrund unterschiedlicher Realisierungsmöglichkeiten für jede der genannten Funktionen ergibt sich ein breites Spektrum alternativer Einsatzformen des GS, die es gegeneinander abzuwägen gilt.

Inhalt:

1. Einführung

An DB-gestützte Transaktionssysteme werden zunehmend steigende Anforderungen bezüglich Leistung (Durchsatz), Verfügbarkeit und Wachstumsfähigkeit ('scalability') gestellt, die den Einsatz von Mehrrechner-DBS verlangen [HR86]. Im Bereich von typischen Anwendungen solcher Hochleistungssysteme (Großbanken, Fluggesellschaften, Versicherungen etc.) werden auch in absehbarer Zukunft trotz häufig geographisch verteilter Organisationsstruktur die gemeinsamen Datenbanken vielfach in zentralen Rechenzentren verwaltet werden [Tr83] (einfachere Administration, 'gewachsene' Strukturen, langsame Übertragung nur für Ein-/Ausgabenachrichten und nicht für DB-Operationen innerhalb einer Transaktion [DYB87]). Lokal gekoppelte Mehrrechner-DBS, welche aus einer homogenen Menge räumlich benachbarter, autonomer Rechnerknoten (oder sogenannter *Computing Modules, CM*) bestehen, stellen daher eine geeignete Grundlage zur Realisierung von Hochleistungs-Transaktionssystemen dar. Die lokale Rechneranordnung erlaubt dabei eine effiziente Kopplung der einzelnen Knoten, welche entweder nachrichtenbasiert ('lose Kopplung') oder unter Verwendung gemeinsamer Speicherbereiche erfolgen kann. Von einer sogenannten 'nahen Rechnerkopplung' (closely coupled systems) über gemeinsame (Halbleiter-) Speicherbereiche verspricht man sich eine wesentlich effizientere Kommunikation [HR86, Ra86] zwischen den CM als bei loser Rechnerkopplung, wo aufwendige Protokolle und Prozeßwechsel typischerweise einen hohen Kommunikations-Overhead und möglicherweise signifikante Antwortzeitverschlechterungen verursachen.

DB-Sharing ('shared disk') bezeichnet eine Klasse lokal gekoppelter Mehrrechner-DBS, bei der jeder Knoten direkten Zugriff auf die gesamte materialisierte Datenbank auf den Externspeichern (Platten) besitzt [HR86]. Gegenüber sogenannten DB-Distribution-Systemen ('shared nothing') ergeben sich für DB-Sharing wesentliche Vorteile hinsichtlich Flexibilität der Lastverteilung, Last-Balancierung, Datenverfügbarkeit sowie Erweiterbarkeit des Systems. Darüber hinaus kann DB-Sharing als evolutionäre Weiterentwicklung aus zentralisierten DBS aufgefaßt werden, bei der im Gegensatz zu DB-Distribution oder verteilten DBS i.a. keine Änderungen an bestehenden Anwendungsprogrammen oder Datenbanken erforderlich werden. So eignet sich der DB-Sharing-Ansatz auch für nicht-relationale DBS, bei denen eine Partitionierung der Datenbestände oft nur sehr eingeschränkt möglich ist. Auf der anderen Seite können aufgrund der direkten Plattenanbindung aller Rechner durch die Architektur des E/A-Subsystem (z.B. bei Multiports) Beschränkungen hinsichtlich der Anzahl einsetzbarer CMs auftreten (häufig nur bis zu acht CM). Desweiteren müssen für eine Reihe von DB/DC-Komponenten neue und aufeinander abgestimmte Lösungen entwickelt werden (Synchronisation und Behandlung des Pufferinvalidierungs- problems, Lastverteilung und -Balancierung, Logging und Recovery), um das Leistungspotential der DB-Sharing-Architektur nutzen zu können.

Bei den bisherigen Untersuchung möglicher Lösungsstrategien für DB-Sharing [Ra88a] wurde meist eine lose Rechnerkopplung unterstellt, welche prinzipiell die beste Verfügbarkeit und Erweiterbarkeit verspricht. Gegenstand dieses Berichtes ist die Betrachtung möglicher Einsatzformen einer nahen Rechnerkopplung bei DB-Sharing über einen speziellen Halbleiterspeicherbereich, den sogenannten *Global Storage (GS).* Allgemeinere Überlegungen bezüglich einer nahen Speicherkopplung bei DB-Sharing wurden bereits in [Ra86] vorgestellt. Im Amoeba-Projekt [Sh85] soll ein solcher Speicherbereich zum beschleunigten Ausschreiben geänderter Seiten sowie deren Austausch zwischen den einzelnen Rechnerknoten genutzt werden; ähnliche Verwendungsformen werden in [DIRY87] vorgeschlagen

.

Bild 1: Stellung des GS in einem DB-Sharing-System

Bei der Betrachtung möglicher Verwendungsformen eines GS bei DB-Sharing sollen hier v.a. einige der dabei zu behandelnden Probleme aufgezeigt und erste Lösungsansätze diskutiert werden. Der vorläufige Charakter der Überlegungen wird schon dadurch hervorgerufen, daß ein solcher GS zur Zeit noch nicht verfügbar ist, sondern sich erst im Planungsstadium befindet. Die im folgenden zusammengestellten **GS-Eigenschaften** aus heutiger Sicht, auf denen die danach folgenden Ausführungen basieren, können daher durchaus noch Änderungen unterworfen sein:

- Der GS erlaubt eine nahe (Speicher-) Kopplung von bis zu vier CM (z.B. vier Quadro-prozessoren)
- Es werden zwei Zugriffsgranulate unterstützt:
  4 KB-Seiten: Zugriffsgeschwindigkeit 15 Mikrosek. (entspricht einer Netoübertragungs-
  leistung von mind. 270 MB/s)
  8 B-Einträge: Zugriffszeit 1 Mikrosekunde
  Die Zugriffe erfolgen synchron, d.h. ohne Prozessorwechsel [GS86].
- Es liegt keine Instruktionsadressierbarkeit vor, d.h. das Manipulieren von Seiten- oder Eintragsinhalten erfordert i.a. ein Einlesen in den Hauptspeicher sowie ein sich anschließendes Zurückschreiben in den GS.
- Nichtflüchtigkeit
- Zur Erhöhung der Fehlertoleranz können Teile des GS doppelt geführt werden (analog zu Spiegelplatten).
- Der Nachrichtenaustausch zwischen den CM (Interprozessor-Kommunikation) kann durch einen GS beschleunigt werden. In [Kö88a] wurden für das Senden und Empfangen einer Nachricht zusammen 2500 Instruktionen angesetzt. Da die Nachrichten auch entweder in 8 B- oder 4 KB-Einheiten abgelegt werden müssen, sind die entsprechenden Verzögerungen für das Ausschreiben der Nachricht in den GS und das spätere Einlesen aus dem GS (sowie für mögliche Verwaltungsoperationen bezüglich des GS) hinzuzählen. Genauere Einzelheiten sind hier noch nicht festgelegt (siehe auch 4.2).

- Die Kapazität des GS beträgt wahrscheinlich mehrere GB.
- Bei dem GS handelt es sich (zunächst) nicht um einen 'echten' Zwischenspeicher zwischen Hauptspeicher (HS) und Platten-/Externspeicher (ES), sondern alle Datentransfers GS <-> ES gehen über die HS (siehe Bild 1).
- Der GS ist eine 'passive' Speichereinheit, für den die Kontrolle und Verwaltung weitgehend durch SW-Maßnahmen der CM geschieht (im Gegensatz etwa zu Disk Caches, welche vom Platten-Kontroller verwaltet werden). Unterstützende Operationen (Compare and Swap) zur Zugriffssynchronisation werden angeboten.

Neben der Nichtflüchtigkeit liegt die Hauptattraktivität des GS in den schnellen Zugriffsgeschwindigkeiten im unteren Mikrosekundenbereich, die eine effektivere Kooperation und Kommunikation zwischen den Rechnern verspricht als ausschließlich über Nachrichten. Im Vergleich zu Platten erlaubt der GS einen um rund drei Größenordnungen (Faktor 10**3) schnelleren Zugriff; gegenüber Hauptspeichern jedoch ist der Zugriff immer noch um etwa den Faktor 10**2 langsamer. Diese Größenordnungen werden die Einsatzformen des GS bei DB-Sharing sowie die Algorithmen maßgeblich beeinflußen.

Als generelle Probleme bzw. Nachteile sind zu nennen:

- begrenzte Erweiterbarkeit (Anzahl anschließbarer Knoten)
- hohe Kosten (wahrscheinlich um ein Mehrfaches höher als für Platten vergleichbarer Kapazität)
- Verfügbarkeit nach Rechnerausfällen kann durch den GS beeinträchtigt werden (verseuchte Datenstrukturen u.ä.)
- Reduzierung der effektiven CPU-Leistung durch synchrone Zugriffe (bei 25 MIPS-Prozessoren entspricht jeder Blockzugriff im GS bereits 375 Instruktionen).

Kap. 2 gibt zunächst einen Überblick über mögliche Einsatzformen eines GS bei DB-Sharing sowie zu beachtende Alternativen, welche dann in den darauffolgenden Kapiteln näher diskutiert werden.

2. Generelle Verwendungsmöglichkeiten eines GS bei DB-Sharing

Die beiden Zugriffsgranulate 'Eintrag' und 'Seite' bestimmen im wesentlichen mögliche Verwendungsformen des GS. So bieten sich *GS-Einträge* zur Speicherung globaler Datenstrukturen (Tabellen) an, z.B. zur systemweiten Synchronisation, zur Behandlung des Veralterungsproblems oder aber zur Verwaltung des GS selbst. Demgegenüber eignet sich der *GS-Seitenbereich* vor allem zur Aufnahme von DB-Seiten (*) - entweder zum Austausch von geänderten Seiten zwischen den CM oder infolge einer Verdrängung aus einem lokalen Systempuffer - und Logging-Daten. Die Realisierung einer beschleunigten *Interprozessor-Kommunikation über den GS* kann entweder mit GS-Einträgen oder GS-Seiten realisiert werden, abhängig von der Größe der Nachricht. So kann z.B. der Austausch einer geänderten DB-Seite über den GS als beschleunigtes Senden einer 'großen Nachricht' aufgefaßt werden.

Bei DB-Sharing kommt der GS v.a. zur Realisierung eines globalen Systempuffers sowie zur Unterstützung der globalen Synchronisation, der Behandlung von Pufferinvalidierungen, des Logging sowie der Lastkontrolle in Betracht:

Synchronisation

Eine Möglichkeit besteht darin, zur Synchronisation eines der *nachrichtenbasierten Protokolle*, die für lose gekoppelte DB-Sharing-Systeme vorgeschlagen wurden [Ra88a,b], einzusetzen und den GS nur zur beschleunigten Interprozessor-Kommunikation zu verwenden. Von der Vielzahl der denkbaren Protokolle kommt dabei v.a. das *Primary-Copy-Sperrverfahren* (mit Leseoptimierung und integrierter Behandlung des Veralterungsproblems) in Frage, daß sich bei Trace-getriebenen Simulationen als am leistungsfähigsten erwiesen hat [Ra88a]. Neben den angesprochenen Optimierungen trug dazu v.a. bei, daß bei dem Verfahren Last- und PCA-Verteilung gezielt (und relativ einfach) aufeinander abgestimmt werden können, so daß die Transaktionen (TA) weitgehend lokal synchronisierbar sind und ein hohes Maß an rechnerspezifischer Lokalität entsteht (soweit es die Lastcharakteristika bzw. die Anzahl der Rechner zulassen). Von Vorteil ist ferner, daß mit dem Primary-Copy-Verfahren eine Synchronisation auf feinen Granulaten (Satz, Eintrag) sowie eine Spezialbehandlung für sogenannte High-Traffic-Objekte noch am ehesten möglich ist [Ra88a]. Von Nachteil ist die aufwendige und komplexe Crash-Recovery (REDO für gescheiterten Rechner sowie dessen gesamter DB-Partition, Rekonstruktion verlorengegangener Sperrtabellen, PCA-Umverteilung, Umstellung der Routing-Strategie [Ra88a]) sowie die Abhängigkeiten des Verfahrens zur 'Gutmütigkeit' der TA-Last (hoher Anteil von Lesezugriffen bzw. gute 'Partitionierbarkeit', so daß verschiedene TA-Typen möglichst auf disjunkten DB-Teilen operieren), so daß ein lineares Durchsatzwachstum oft nur bis zu einer bestimmten Rechneranzahl erreichbar ist. Letzterer Aspekt stellt hier jedoch keinen entscheidenden Nachteil dar, da mit einem GS die mögliche Anzahl von Knoten ohnehin stark begrenzt ist.

Eine weitergehende Verwendung des GS ergibt sich, wenn *im GS globale Datenstrukturen zur Synchronisation* abgelegt werden, welche von allen Knoten benutzt werden. Eine derartige Nutzung des GS zur Synchronisation läuft - im Gegensatz zum Primary-Copy-Verfahren - auf ein zentrales Sperrverfahren (**) hinaus, mit dem es aufgrund der hohen GS-Zugriffsgeschwindig-keit möglich sein

---

* Die 2 KB-Seiten der UDS-Datenbanken müssen dabei auf die 4 KB-Seiten abgebildet werden.

** Optimistische Protokolle werden hier nicht weiter diskutiert, obwohl sie prinzipiell auch über zentrale Datenstrukturen im GS realisierbar wären.

soll, Sperren wesentlich schneller als über ein nachrichtenbasiertes Protokoll zu erwerben und freizugegeben. Wie in Kap. 3 noch näher gezeigt wird, lassen sich zwischen dem Extremfall, bei dem die globale Sperrtabelle komplett im GS abgelegt wird, sowie einer vollkommen verteilten Sperrverwaltung ohne Verwendung gemeinsamer Sperrinformationen im GS (wie beim Primary-Copy-Verfahren) noch eine Reihe von Zwischenformen finden, welche unter verschiedenen Gesichtspunkten (z.B. Reduzierung von GS-Zugriffen, Nutzung von Lokalität, Fehlertoleranz) von Interesse sein können.

Behandlung von Pufferinvalidierungen

Da jeder Knoten eines DB-Sharing-Systems einen lokalen Systempuffer (LSP) führt, kommt es zu einer (dynamischen) Datenreplikation in den Puffern und dem damit verbundenen *Veralterungsproblem* (buffer invalidation problem) [Ra88a,b]. Um unnötige Kommunikationsvorgänge zur Lösung dieses Veralterungsproblems zu vermeiden, ist eine integrierte Behandlung im Rahmen der Synchronisation unter Nutzung erweiterter Sperrinformationen (z.B. um veraltete Seiten erkennen zu können, Angaben zum Aufenthaltsort der aktuellen Seitenversion oder bzgl. der Ausschreibkoordinierung) vorzusehen; dies ist bei einer Synchronisation auf Seitenebene am einfachsten, da hierbei das parallele Ändern einer Seite in verschiedenen Rechnern ausgeschlossen wird. Wird die globale Sperrtabelle im GS abgelegt, so können nun natürlich auch für geänderte DB-Seiten die *Angaben zur Behandlung des Veralterungsproblems in Tabellen des GS* abgelegt werden. Desweiteren kann der GS zum *beschleunigten Austausch geänderter Seiten* zwischen den Rechnern verwendet werden, und zwar sowohl bei einem nachrichtenbasierten Synchronisationsprotokoll als bei einer Synchronisation über zentrale Sperrtabellen im GS. In Kap. 4 wird die Behandlung von Pufferinvalidierungen unter Nutzung des GS näher diskutiert.

Globaler Systempuffer

Eine erweiterte Nutzung des GS-Seitenbereiches ergibt sich, wenn dieser nicht nur zur zum Austausch geänderter Seiten (bzw. allgemein zur beschleunigten Interprozessor-Kommunikation) genutzt wird, sondern auch zur Aufnahme von aus den LSP verdrängten bzw. ausgeschriebenen Seiten. Hauptziele dabei sind das schnellere Ausschreiben geänderter Seiten sowie die Einsparung von Lesevorgängen von Platte. Eine wichtige Entwurfsentscheidung bei der Realisierung eines globalen Systempuffers, welche auch Auswirkungen für die Behandlung des Veralterungsproblems sowie für Logging und Recovery besitzt, ist, ob für geänderte Seiten eine sogenannte *NOFORCE- oder FORCE-Strategie* [HR83] bezüglich des GS verfolgt werden soll (s. Kap. 5). Dabei werden bei FORCE am Ende einer Update-TA sämtliche von ihr geänderte Seiten vom LSP in den GS geschrieben, während bei NOFORCE ein solches 'Hinauszwingen' der Änderungen aus den lokalen Puffern unterbleibt.

Logging

Aufgrund der Nichtflüchtigkeit des GS sowie der im Vergleich zu Platten wesentlich schnelleren Zugriffszeiten bietet es sich natürlich zur Reduzierung der Antwortzeiten auch an, Log-Daten in den GS auszuschreiben. Dazu kann der GS sowohl zur (temporären) Aufnahme lokaler Log-Daten (Terminalnachrichten, UNDO- bzw. REDO-Daten) der einzelnen CM genutzt werden, oder aber auch für globale (REDO-) Log-Daten aller Rechner. In Kap. 6 wird auf diese Verwendungsformen noch näher eingegangen.

Lastkontrolle

Wird die globale Lastkontrolle auf mehreren Front-End-Rechnern realisiert [Ra88a], dann können die einzelnen Prozessoren bei einer nahen Kopplung über einen (eigenen) GS diesen zur schnelleren Kommunikation untereinander nutzen, zur Ablage gemeinsam benutzter Datenstrukturen (z.B. Routing-Tabelle, Angaben zur aktuellen Last- und Verteilsituation, u.ä.) oder zur schnellen Sicherung von Eingabenachrichten. Eine Weiterleitung von TA-Aufträgen bzw. Antwortnachrichten zwischen den Front-End-Rechnern und den Verarbeitungsrechnern über einen GS ist dagegen wegen der starken Beschränkung hinsichtlich der Gesamtanzahl anschließbarer Knoten als weniger sinnvoll anzusehen. Andererseits erscheint mit dem GS ein Verzicht auf ein Front-End-System und eine Lastverteilung durch die Verarbeitungsrechner selbst wieder attraktiver, da Umverteilungen (Migration) von TA-Aufträgen relativ effizient vornehmbar sind. Auf nähere Ausführungen hinsichtlich der Lastkontrolle muß im folgenden jedoch verzichtet werden, da sie den Rahmen dieses Berichtes sprengen würden.

Bild 2: Einsatzalternativen eines GS bei DB-Sharing (Auswahl)

Bild 2 zeigt einen Teil der angesprochenen Einsatzalternativen eines GS bei DB-Sharing (ohne Logging und Lastkontrolle). Dabei stellt sich natürlich zunächst die Frage, ob überhaupt alle sinnvoll erscheinenden Verwendungsformen des GS angestrebt werden sollen. Dies verspricht zwar intuitiv den größten Leistungsgewinn, verursacht andererseits auch die höchsten Kosten (größter Speicherplatzbedarf für den GS), kann möglicherweise zu Engpässen bei der Gesamtzugriffshäufigkeit zum GS führen und ergibt eine starke Abhängigkeit des Systems zur Verfügbarkeit des GS bzw. birgt die höchste Gefahr einer Fehlerausbreitung über gemeinsam benutzte GS-Datenstrukturen. Desweiteren ergeben sich für die einzelnen Funktionen unterschiedliche Verwendungsarten des GS, die zum Teil in Bild 2 aufgenommen wurden und die es gegeneinander abzuwägen gilt. So ist bei der Synchronisation (und zugleich bezüglich der integrierten Lösung des Veralterungsproblems) zu entscheiden, ob ein nachrichtenbasiertes Verfahren (mit beschleunigter Kommunikation über den GS) eingesetzt werden soll, oder ob (und in welchem Ausmaß) globale Sperrinformationen im GS genutzt werden sollen. Bei der Ausschreibstrategie ist zwischen einem FORCE- und NOFORCE-Ansatz zu wählen, wobei dies das Ausschreiben bezüglich des GS betrifft, falls ein globaler Systempuffer vorgesehen wird, anderenfalls das Ausschreiben auf Platte. Im letzteren Fall kann jedoch die FORCE-Strategie wegen der hohen

Schreibverzögerungen für Update-TA von vorneherein ausgeschlossen werden; diese Möglichkeit wurde in Bild 2 lediglich aus Symmetriegründen aufgenommen. Wenn kein globaler Systempuffer vorgesehen wird, dann sollte für NOFORCE zumindest der Austausch geänderter Seiten über den GS abgewickelt werden (s chnellerer Austausch von Änderungen, Entlastung des Kommunikationssystems).

Weitere Alternativen beim Einsatz eines GS, die bisher noch nicht angeprochen wurden, ergeben sich bezüglich der *Verwaltung des GS*, insbesondere bezüglich der Verwaltung des GS-Seitenbereiches (Austausch geänderter Seiten, globaler Systempuffer, Logging). Neben einer Verwaltung über zentrale Datenstrukturen im GS selbst kommt dabei auch ein sogenannter *partitionierter Ansatz* in Frage, bei dem jeder CM eine Partition des GS kontrolliert und die dazu erforderlichen Datenstrukturen lokal verwaltet (z.B. zur Realisierung lokaler Log-Dateien im GS). In [DIRY87] wird ein solcher Ansatz vorgestellt, wobei ein CM zwar Lesezugriffe bezüglich des gesamten gemeinsamen Speicherbereiches (hier: GS) vornehmen kann, Schreibzugriffe jedoch nur bezüglich der von ihm kontrollierten Partition. Daneben sind auch Mischformen aus einem solch partitionierten Ansatz sowie der Verwendung zentraler Verwaltungsdaten im GS denkbar, wobei dies im einzelnen natürlich stark von der vorgesehenen Verwendung der zu verwaltenden GS-Daten abhängt.

3. Synchronisation bei DB-Sharing unter Nutzung eines GS

Da sich für nachrichtenbasierte Protokolle nur bezüglich der Realisierung des Nachrichtenaustauschs über den GS neue Aspekte ergeben (s. 4.2), betrachten wir in diesem Kapitel lediglich diejenigen Alternativen, bei denen der GS zur Ablage globaler Sperrinformationen genutzt wird. Zur Verdeutlichung grundlegender Fragestellungen bei einer derartigen Verwendung des GS für Synchronisationszwecke wird zunächst der einfachste Fall, bei dem die globale Sperrtabelle vollkommen im GS residiert, diskutiert. In 3.2 stehen dann Verfahren im Mittelpunkt, bei dem im GS eine reduzierte Sperrinformation abgelegt wird und die CM verstärkt zur Synchronisation beitragen. In 3.3 folgt eine abschließende Wertung der diskutierten Lösungsalternativen.

3.1 Globale Sperrtabelle vollkommen im GS

In zentralisierten DBS wird die Sperrinformation im Hauptspeicher typischerweise durch eine variable Anzahl TA- und objektspezifischer Kontrollblöcke repräsentiert, die wiederum eine variable Anzahl von Untereinträgen (z.B. gehaltene Sperren einer TA; gewährte oder wartende Sperranforderungen eines Objektes) enthalten. Wird eine maximale Parallelität pro Rechner festgelegt, so ist die Anzahl von TA-Einträgen begrenzt und relativ klein. Bei den Objekten ist dies jedoch i.a. nicht der Fall; die objektspezifischen Kontrollblöcke werden daher i.d.R. in Hash-Tabellen verwaltet. Zur Kollisionsbehandlung (mehrere Objekteinträge pro Hash-Klasse) werden entweder die Kontrollblöcke einer Hash-Klasse verkettet oder aber es wird für die Überläufer ein eigener Überlaufbereich verwaltet [Pe86]. Semaphore (z.B. pro Hash-Klasse) verhindern den gleichzeitigen Zugriff auf die Datenstrukturen durch verschiedene DBVS-Prozesse (Subtasks). Die Erkennung von Deadlocks kann durch Zyklensuche bei jedem Sperrkonflikt erfolgen.

Diese variabel langen Datenstrukturen, die einer hohen Lese- und Änderungsfrequenz unterworfen sind, sollen nun im GS verwaltet werden, wobei im Prinzip lediglich die 8 B-Einträge zur Verfügung stehen. Um die Diskussion überschaubar zu halten, konzentrieren wir uns hier auf die Speicherung der *objekt-*

spezifischen Sperrinformationen (*) (globale Sperrtabelle), wobei für jedes sperrbare Datenobjekt sowohl Informationen fester als auch variabler Länge zu führen sind. Zu den *Angaben fester Länge* zählen v.a. die Objektidentifikation (i.a. > 4 B) sowie höchster gewährter Sperrmodus oder Anzahl wartender Sperranforderungen; damit kann dann bereits entschieden werden, ob eine Sperranforderung gewährbar ist oder nicht. Die *variablen Angaben* bestehen v.a. aus Listen von gewährten und wartenden Sperranforderungen, die prinzipiell zu einer Liste zusammengelegt werden können, wenn die gewährten Sperren stets am Anfang der Liste liegen [Pe86]. Pro Sperranforderung ist dabei eine TA-Kennung (z.B. aus CM-ID und relativer TA-Nummer) sowie der Sperrmodus zu führen.

Zur Realisierung der Listenstruktur(en) im GS ist offensichtlich eine explizite *Verzeigerung von GS-Einträgen*, welche Sperrinformationen desselben Objektes führen, erforderlich, wobei die Verweise zu den Listen im festen Teil der Sperrangaben abzulegen sind. Wenn unterstellt wird, daß die GS-Adresse eines Eintrages 4 B umfaßt, kann pro Eintrag höchstens ein Pointer vorgesehen werden (es ist so also keine Mehrfachverzeigerung realisierbar, wie z.B. für einige Listenoperationen sinnvoll; außerdem können keine getrennte Listen für gewährte oder wartende Sperranforderungen geführt werden). Mehr Freiheitsgrade ergeben sich bei einer relativen GS-Adressierung der Listenelemente, wenn diese aus einem Eintrags-Pool mit fester GS-Basis-adresse allokiert werden; in diesem Fall dürften z.B. 2 B-Adressen ausreichen (Pool-Größe von 64 K Einträgen). Trotzdem reicht auch in diesem Fall ein einziger 8 B-Eintrag für den festen Teil der Sperrinformationen i.a. nicht aus, da bereits die Objektidentifikation typischerweise 8 B lang ist.

Zur Umgehung dieses Problems, das auch bei der Realisierung anderer Tabellen (z.B. zur Speicherung von Informationen bezüglich des Veralterungsproblems oder zur Verwaltung eines globalen Systempuffers) auftritt, kommen im wesentlichen zwei Alternativen in Betracht, die in 3.1.1 bzw. 3.1.2 für die Realisierung einer globalen Sperrtabelle näher diskutiert werden:

- Die erste Möglichkeit ist eine *starre Organisation* der Sperrtabelle, bei der auf eine explizite Speicherung des Objektnamens verzichtet wird und stattdessen für jedes sperrbare Objekt (DB-Seite) von vorneherein ein Kontrolleintrag vorgesehen wird.
- Eine flexiblere Organisation ergibt sich, wenn die *Objektidentifikation explizit* in den Kontrolleinträgen *gespeichert* wird; dies erfordert jedoch größere Einträge als 8 B (z.B. 16 B) zur Verwaltung der Sperrinformation fester Länge. Eine Alternative dazu ist natürlich, für jedes zu verwaltende Objekt stets mehrere 8 B-Einträge vorzusehen; es ergibt sich dann aber auch ein Mehrfaches der Zugriffskosten.

3.1.1 Starre Sperrverwaltung mit impliziter Objektidentifikation

Bild 3 zeigt eine mögliche Realisierung der globalen Sperrtabelle mit impliziter Objektidentifizierung, ähnlich wie in [Kö88a] unterstellt. Dabei wird in einem festen Teil der Sperrtabelle für jedes der maximal K sperrbaren Objekte ein sogenannter *Primäreintrag* von 8 B vorgesehen, unabhängig davon, ob eine Sperre für das Objekt angefordert wurde oder nicht. Daneben besteht ein Pool von Einträgen, der zur Realisierung der Listenstrukturen für gewährte und wartende Sperranforderungen dient, welche den

---

* TA-spezifische Angaben wie Menge gewährter Sperren einer TA werden sinnvollerweise in den CM gehalten, um lokal feststellen zu können, ob eine TA bereits eine ausreichende Sperre für ein zu referenzierendes Objekt besitzt, oder um die bei EOT freizugebenden Sperren zu kennen.

variablen Teil der globalen Sperrtabelle repräsentieren (*). Zur Verzeigerung der GS-Einträge wurde in Bild 3 in jedem Eintrag ein Pointer fester Länge (z.B. 2 bis 4 B) vorgesehen. Der Verweis aus einem Primäreintrag zeigt dabei immer auf das erste Element der gemeinsamen Liste gewährter und wartender Sperranforderungen (*GWS-Liste*), während die Pointer von Einträgen des variablen Teils auf Folgeeinträge dieser objektspezifischen Liste verweisen. Die Elemente der GWS-Liste wollen wir auch als *Sekundäreinträge* bezeichnen. So zeigt das Beispiel in Bild 3 an, daß für Objekt O den TA T1 und T2 eine Lesesperre (R-Sperre) gewährt wurde, während T3 auf eine X-Sperre wartet.

Bild 3: Globale Sperrtabelle bei impliziter Objektidentifizierung (Beispiel)

Wie das Beispiel verdeutlicht, wird in den Sekundäreinträgen neben dem Folgeverweis im wesentlichen die Rechner- und TA-Kennung sowie der Zugriffswunsch (Sperrmodus) geführt. Im Primäreintrag können dagegen noch Angaben wie gewährter Sperrmodus oder Anzahl wartender TA gehalten werden.

Hauptproblem der skizzierten Speicherorganisation ist natürlich die Beschränkung auf eine maximale Anzahl K sperrbarer Objekte, welche zudem eindeutig auf das Intervall [1..K] abbildbar sein müssen. Eintrags- oder Satzsperren scheiden so schon von vorneherein aus. Selbst bei Seitensperren wird für große Datenbanken bereits für den festen Teil der globalen Sperrtabelle ein enormer Speicherplatzbedarf eingeführt: dieser umfaßt bei DB-Seiten von 2 KB und einer DB-Größe von 100 GB bereits 400 MB (wovon der überwiegende Teil ungenutzt bleibt); ein ebenso hoher Platzbedarf, der sich bei Spiegelung noch verdoppelt, ergibt sich für weitere Tabellen mit starrer Organisation. Der variable Teil der globalen Sperrtabelle dürfte dagegen vergleichsweise klein gehalten werden können, da typischerweise kurze TA dominieren und die Gesamtzahl parallel gestarteter TA auch begrenzt ist.

Eine Alternative zur Reduzierung des Speicherplatzbedarfes wäre, Hash-Klassen als zu sperrende Objekte vorzusehen; bei einer ausreichend hohen Anzahl von Hash-Klassen dürften damit auch kaum mehr Sperrkonflikte als bei einer Synchronisierung auf Blockebene auftreten. Der Vorteil wäre, daß die Größe der Sperrtabelle (weitgehend) unabhängig von der DB-Größe (Anzahl DB-Seiten) gehalten werden kann. Zudem könnte auch eine (näherungsweise) Synchronisierung auf Satz- oder Eintragsebene unterstützt werden, indem die (eindeutige) Adresse des Satzes/Eintrages anstelle der Seitennummer auf eine der Hash-Klassen abgebildet wird. Allerdings wird es nun schwer möglich, die Lösung des Veralterung-

* Zur Allokation von Elementen aus diesem Pool kann durch ein Bit pro Eintrag angezeigt werden, ob dieser 'frei' oder 'belegt' ist. Zum Auffinden eines freien Eintrages sind mehrere Realisierungsformen denkbar, angefangen von der sequentiellen Suche bis zur Selektion über eine Hash-Funktion.

sproblems (welches die Hauptschwierigkeit bei einer Synchronisation auf Satz-/Eintragsebene darstellt) zusammen mit dem Sperrprotokoll zu lösen, da hierzu eine genaue Identifizierung der betroffenen Seite (bzw. des Eintrages) erforderlich wird, welche jedoch aus der Hash-Klasse allein nicht ableitbar ist.

Im folgenden soll nun kurz diskutiert werden, welche Aktionen bezüglich des GS für die oben skizzierte Realisierung der globalen Sperrtabelle beim Setzen und Freigeben von Sperren anfallen.

LOCK:

Zunächst ist der für das zu sperrende Objekt zugehörige Primäreintrag im festen Teil der Sperrtabelle in den Hauptspeicher des zugreifenden CM einzulesen. Der Primäreintrag erlaubt eine Entscheidung darüber, ob die Sperre gewährbar ist oder nicht (z.B. über gewährten Sperrmodus und Anzahl wartender Anforderungen). Kann die Sperre gewährt werden, ist ein Sekundäreintrag im variablen Teil der Sperrtabelle zu belegen und vorne in die GWS-Liste einzuhängen. Insgesamt fallen somit *mindestens vier GS-Zugriffe pro Sperranforderung* an (Lesen und Zurückschreiben des geänderten Primär- sowie des Sekundäreintrages). Noch mehr Zugriffe entstehen bei abgelehnter Sperranforderung, da hierbei ein Sekundäreintrag an das Ende der GWS-Liste anzufügen ist. Dies erfordert jedoch ein sequentielles Lesen der gesamten Liste, wenn nicht im Primäreintrag das aktuelle Listenende abgelegt wird. Zusätzliche GS-Zugriffe ergeben sich natürlich auch, wenn beim Zurückschreiben eines Eintrages im Rahmen eines Compare&Swap-Befehles festgestellt wird, daß der Eintrag zwischenzeitlich von einem anderen CM verändert wurde. In diesem Fall muß die LOCK-Bearbeitung i.a. wiederholt werden.

UNLOCK:

Es wird davon ausgegangen, daß für jede TA im HS des ausführenden CM sämtliche Sperren (mit GS-Adresse des zugehörigen Sekundäreintrages), welche die TA hält, geführt werden, damit bekannt ist, welche Sperren beim Commit bzw. Abort der TA freizugeben sind. Für jede Sperrfreigabe ist der Sekundäreintrag aus der GWS-Liste zu entfernen (Lesen und Zurückschreiben des Sekundäreintrages sowie dessen Vorgänger in der GWS-Liste). *Im günstigsten Fall* (wenn der Sekundäreintrag das erste Element der GWS-Liste ist) entstehen so *vier GS-Zugriffe pro Sperrfreigabe*. Weitere Zugriffe fallen an, wenn durch die Sperrfreigabe wartende Sperranforderungen gewährbar werden. In diesem Fall müssen mittels Durchlaufen der GWS-Liste die gewährbaren Anforderungen bestimmt sowie die zugehörigen Sekundäreintrage ausgekettet werden. Ferner sind Benachrichtigungen an die Rechner zu schicken, um die Aktivierung der betroffenen TA zu veranlassen.

Durch die genannten Zugriffshäufigkeiten ergeben sich keine nennenswerten Antwortzeiterhöhungen für die TA. Nimmt man statt der mindestens 8 GS-Zugriffe pro Sperre 10 an, so ergibt das für eine TA mit 10 Sperren lediglich eine Antwortzeiterhöhung um 0.1 ms. Kritischer dagegen ist bereits die Gesamtzugriffshäufigkeit zum GS. Denn bei 1000 TA/s fallen dann allein zur Synchronisation bereits 100.000 GS-Zugriffe pro Sekunde an, die eine Auslastung von etwa 10 % verursachen.

Problematisch ist auch die *Behandlung von Rechnerausfällen*, wobei ähnliche Schwierigkeiten wie bei eng gekoppelten Multiprozessoren auftreten. So muß erkannt werden können, welche Einträge der globalen Sperrtabelle von dem ausgefallenen Knoten in Bearbeitung waren und durch den Ausfall möglicherweise in einem inkonsistenten Zustand hinterlassen wurden. Letzteres ist möglich, da z.B. beim Setzen einer Sperre mehrere GS-Einträge zu ändern sind, was i.a. nicht atomar (z.B. mit einem einzigen GS-Befehl) möglich sein dürfte. Die betroffenen GS-Einträge können zum Beispiel bei Doppeltführen der Sperrtabelle erkannt werden; dies impliziert jedoch nicht nur den doppelten Speicherplatzbedarf, sondern auch den doppelten Schreibaufwand (mindestens zwei zusätzliche GS-Zugriffe pro Sperre) und dementsprechend eine noch höhere GS-Auslastung (> 10 %). Zur Durchführung der Crash-Recovery müssen auch alle Sperranforderungen des betroffenen CM, die in der globalen Sperrtabelle hinterlegt sind, identifiziert werden (z.B. durch sequentielles Lesen der gesamten Tabelle), um diese entfernen und wartende TA anderer Knoten aktivieren zu können.

3.1.2 Flexible Sperrverwaltung mit expliziter Objektidentifizierung

Die explizite Speicherung der Objektnamen in der Sperrtabelle bringt den großen Vorteil mit sich, das nur noch für tatsächlich zu sperrende Objekte Kontrollinformationen im GS zu hinterlegen sind, wodurch i.a. eine drastische Reduzierung des Speicherplatzbedarfes im Vergleich zur starren Tabellenorganisation möglich wird. Allerdings reicht nun, wie erwähnt, ein 8 B-Eintrag nicht mehr aus, um den festen (primären) Teil der Kontrollinformationen zu einem Objekt zu speichern, da oft die Objektidentifikation bereits 8 B umfaßt. Wünschenswert wäre daher, größere GS-Einträge als 8 B bereitzustellen (*), da die Verwendung mehrerer 8 B-Einträge für den festen Teil der Sperrinformation zu einer entsprechenden Erhöhung der GS-Zugriffsfrequenz führt und (wahrscheinlich) keine atomare Änderung mehrerer Einträge möglich ist.

Eine mögliche Realisierungsform einer flexiblen Tabellenorganisation mit 8 B-Einträgen ist in Bild 4 dargestellt. Es handelt sich dabei um eine Erweiterung der in 3.1.1 vorgestellten Tabellenorganisation, wobei v.a. zwei neue Elemente hinzukommen:

- Zum einen soll die feste Sperrinformation eines zu verwaltendes Objektes nicht mehr in einem einzigen Primäreintrag von 8 B abgelegt werden, sondern in zwei im GS aufeinanderfolgenden 8 B-Einträgen. Im ersten dieser zwei Einträge wird dabei der Name des Objektes (z.B. Seitennummer) gespeichert, während der zweite Eintrag die restliche Sperrinformation fester Länge (höchster gewährter Sperrmodus, Anzahl wartender Sperranforderungen, etc.) sowie zwei Pointer (von je 2 B) enthält. Einer dieser Verweise zeigt dabei, wie schon in 3.1.1, auf den Beginn der GWS-Liste, welche aus einer variablen Anzahl von Sekundäreinträgen (von je 8 B) besteht und gewährte sowie wartende Sperranforderungen enthält. Der zweite Pointer dient zur Verkettung innerhalb des festen Teiles der globalen Sperrtabelle (s.u.).

Die Sperrkontrollblöcke zu den Objekten (bestehend aus je zwei 8 B-Einträgen) werden zur schnellen Lokalisierung im Rahmen einer *Hash-Tabelle mit Überlaufbereich* abgelegt. Wie Bild 4 zeigt, besteht diese Hash-Tabelle aus einem sogenannten Primärbereich mit H1 Hash-Klassen sowie einem Sekundär- oder Überlaufbereich von H2 Hash-Klassen, wobei pro Hash-Klasse jeweils Angaben zu höchstens einem Objekt abgelegt werden (d.h., pro Hash-Klasse werden genau zwei 8 B-Einträge benötigt). Ist die einem Objekt über die Hash-Funktion zugeordnete Hash-Klasse im Primärbereich bereits von einem anderen Objekt beansprucht, wird für die Sperrangaben, wiederum über eine Hash-Funktion, ein freier Platz im Überlaufbereich belegt (Kollisionsbehandlung). Da nun nur für tatsächlich benutzte DB-Objekte Sperrinformationen geführt werden, ergibt sich selbst bei einer hohen Anzahl von Hash-Klassen ein weit geringerer Speicherplatzbedarf für die globale Sperrtabelle als bei impliziter

---

* Die Ablage der globalen Sperrtabelle in 4 KB-Einheiten scheitert an der 'langsamen' Zugriffsgeschwindigkeit von 15 Mikrosekunden, die bei höheren TA-Raten zu Engpässen führen würde.

Objektidentifizierung (bei 200 K Hash-Klassen werden z.B. nur 3.2 MB für die Hash-Tabelle benötigt).

Bild 4: Aufbau der globalen Sperrtabelle bei expliziter Objektidentifizierung

Beim Zugriff auf die globale Sperrtabelle wird zunächst immer die einem Objekt zugeordnete Hash-Klassen-Information des Primärbereiches inspiziert. Tritt dabei eine Namenskollision auf (wenn die beiden Einträge zu der Hash-Klasse also bereits mit Sperrinformationen eines anderen Objektes belegt sind), so wird über eine zweite Hash-Funktion eine Hash-Klasse des Überlaufbereiches bestimmt, wo Angaben zu dem Objekt gespeichert bzw. gefunden werden können (kommt es dabei erneut zu einer Kollision, so wird über eine erneute Prozedur ein anderer Platz im Überlaufbereich ermittelt usw.). Ein Problem entsteht nun durch das Löschen von nicht mehr benötigten Sperreinträgen für ein Objekt O im Primärbereich, wenn aufgrund von Namenskollisionen zum Löschzeitpunkt noch Sperrangaben im Überlaufbereich stehen, die zu Objekten gehören, die in die gleiche Hash-Klasse des Primärbereiches wie O fallen. Die Angaben im Überlaufbereich werden dann nämlich nicht mehr gefunden, wenn die Hash-Klasse im Primär-Bereich freigegeben und die ursprüngliche Namenskollision damit aufgehoben wird.

Zur Behandlung dieser Schwierigkeit wird für jede Hash-Klasse (im zweiten Eintrag) ein Pointer geführt, der im Falle eines Namenskonfliktes auf die Hash-Klasse des Überlaufbereiches verweist, in dem die Sperrangaben des betroffenen Objektes abgelegt wurden. Bei einer relativen Adressierung innerhalb des Überlaufbereiches dürften dazu 2 B ausreichen, wodurch bis zu 64 K Hash-Klassen im Überlaufbereich zugelassen werden. Soll die Sperrinformation zu einem Objekt im GS nun freigegeben werden, so wird durch den Pointer angezeigt, ob (und wo) für die Hash-Klasse ein Überläufer vorliegt. Ist dies der Fall, so sind in der globalen Sperrtabelle die Angaben zu dem Überläuferobjekt in den frei werdenden Platz zu speichern, wobei (bei mehreren Namenskollisionen) ein solches Verschieben der Sperrinformationen für mehrere Objekte erforderlich werden kann.

Die Verwendung von zwei Einträgen für die feste Sperrinformation eines Objektes sowie die deutlich aufwendigere Tabellenverwaltung im Vergleich zur starren Organisation (Anlegen, Auffinden und Löschen von Kontrollblöcken) impliziert natürlich einen höhere GS-Zugriffshäufigkeit, auch wenn Namenskollisionen typischerweise selten vorkommen sollten.

So werden beim *LOCK* mindestens 6 GS-Zugriffe erforderlich, wenn die dem zu sperrenden Objekt zugeordnete Hash-Klasse im Primärbereich noch nicht belegt ist und die Sperre damit gewährt werden kann (Lesen und Zurückschreiben von 2 Einträgen der Hash-Tabelle sowie eines Sekundäreintrages). Liegen für das Objekt bereits Kontrollinformationen im GS vor, so ergeben sich wenigstens 5 GS-Zugriffe zum Setzen der Sperre (in diesem Fall muß der Objektname nicht mehr geschrieben werden). Weitere GS-Zugriffe entstehen bei abgelehnter Sperranforderung, im Falle von Namenskollisionen oder bei Zugriffskonflikten mit anderen CM (Compare&Swap).

Beim UNLOCK ergeben sich auch wenigstens 5 GS-Zugriffe; im Falle von gewährbaren Sperranforderungen, Namenskollisionen oder wenn die GS-Informationen aufgegeben werden können jedoch mehr. Zu beachten ist, daß aufgrund von Verschiebungen der Sperrangaben, die Sperrinformationen eines Objektes beim UNLOCK an einer anderen GS-Adresse als beim LOCK vorliegen können.

Insgesamt sind also pro Sperre etwa 15 GS-Zugriffe einzukalkulieren, was für das Zahlenbeispiel aus 3.1.1 mit 1000 TA/s zu einer GS-Auslastung von 15 % führt. Diese erhöht sich auf etwa 20 %, wenn die globale Sperrtabelle doppelt geführt wird, da dann alle Schreibzugriffe doppelt auszuführen sind. Somit ergibt sich für 8 B-Einträge durch die Hash-Tabelle mit expliziter Objektidentifizierung eine um etwa 50 % höhere GS-Zugriffsfrequenz als bei starrer Tabellenorganisation.

3.2 Sperrinformationen im GS sowie in den CM

Die obige Diskussion hat gezeigt, daß die globale Synchronisation über den GS im Vergleich zu einem nachrichtenbasierten Protokoll bei loser Rechnerkopplung erheblich effizienter abgewickelt werden kann. Denn bei loser Kopplung kann eine TA nur im 'idealen Fall', wenn (nahezu) alle Sperranforderungen lokal befriedigbar sind (strikte Partitionierbarkeit der Last), effizienter synchronisiert werden; dies ist aber für die Mehrzahl der Anwendungen nicht zu erwarten. Bei nur einer globalen Sperranforderung pro TA jedoch sind die damit verbundenen Antwortzeitauswirkungen sowie Kommunikationsbelastungen i.a. bereits größer als bei einer Synchronisierung über den GS, welche üblicherweise die Antwortzeit um deutlich weniger als eine Millisekunde erhöhen sollte. Damit wird zugleich eine weitaus größere Unabhängigkeit zu den Charakteristika und Verteilbarkeit der TA-Last ermöglicht, so daß hohe TA-Raten, kurze Antwortzeiten und lineares Durchsatzwachstum eher zu erreichen sind.

Andererseits wurde deutlich, daß die Verwaltung variabler Datenstrukturen im GS nur sehr umständlich möglich ist, daß die Behandlung von Rechnerausfällen problematisch sein kann (Verfügbarkeit) sowie daß bei hohen TA-Raten (> 1000 TA/s) Engpässe im GS drohen (zumal der GS i.a. nicht nur für Synchronisationszwecke eingesetzt werden soll). Aus diesen Gründen wird hier nun untersucht, wie diese Nachteile abgeschwächt bzw. vermieden werden können, indem auch in den CM objektspezifische Sperrinformationen (innerhalb von lokalen Sperrtabellen) geführt werden. Hierzu betrachten wir im folgenden zwei allgemeinere, aufeinander aufbauende Organisationsformen. Ein erster Schritt besteht darin, dynamische Datenstrukturen weitgehend aus dem GS zu entfernen und in den CM zu verwalten. Damit können dann insbesondere GS-Zugriffe zur (umständlichen) Manipulation der Listenstrukturen eingespart werden; allerdings werden noch sämtliche Sperranforderungen über die globale Sperrtabelle im GS entschieden. Der zweite Schritt besteht dann darin, auch Sperren lokal zu vergeben (und damit weitere GS-Zugriffe einzusparen), wobei auf Techniken zurückgegriffen werden kann, die für Sperrprotokolle bei loser Rechnerkopplung vorgeschlagen wurden (z.B. Sole-Interest, Leseautorisierungen).

3.2.1 Auslagern dynamischer Sperrinformationen in die CM

Hierzu wird angenommen, daß jeder CM eine lokale Sperrtabelle mit sämtlichen gewährten Sperren sowie wartenden Sperranforderungen für lokale TA führt. Das Ziel ist nun, die vollständige Speicherung dieser dynamischen Angaben im GS zu umgehen und dennoch eine korrekte globale Synchronisierung über den GS zu ermöglichen. Die Vorschläge dazu sind sowohl bei einer globalen Sperrtabelle mit impliziter als auch expliziter Objektidentifizierung anwendbar.

Ein erster und einfach zu lösender Schritt ist, im GS die gewährten Sperren nicht mehr auf TA-Ebene, sondern auf Rechnerebene zu vermerken. Da die maximale Rechneranzahl NMAX als begrenzt vorausgesetzt werden kann, genügt es hinsichtlich gewährter Sperren in der globalen Sperrtabelle im GS folgenden Vektor zu führen:

GRANTED-MODE: array [1 .. NMAX] of [0, R, X, ...]

Damit wird für jeden der Rechner angegeben, ob Sperren und wenn ja mit welchem höchsten Sperrmodus an TA des Rechners vergeben sind (0 bedeutet, daß keine Sperren vergeben sind; R, daß nur Lesesperren gewährt wurden, u.s.w). Für die Abspeicherung dieser Information reichen zwei Bit pro Rechner (falls nicht mehr als drei Sperrmodi unterschieden werden), so daß bei NMAX=4 insgesamt 1 B pro Sperreintrag anfällt.

Offensichtlich genügt dieser Vektor (sowie die Angabe, ob bereits Sperranforderungen warten), um entscheiden zu können, ob eine Sperranforderung gewährbar ist oder nicht. Der Vorteil ist, daß die TA, denen eine Sperre gewährt wurde, nur noch in der lokalen Sperrtabelle geführt werden, so daß das Einfügen sowie spätere Ausfügen von Sekundäreinträgen für gewährte Sperren im GS entfällt. Damit werden bei starrer Organisation der globalen Sperrtabelle (3.1.1) zum Setzen einer Sperre statt mindestens 4 nur noch 2 GS-Zugriffe erforderlich. Zum Freigeben einer Sperre können sogar weniger als 2 (statt mindestens 4) GS-Zugriffe ausreichen: denn das Freigeben einer Sperre erfordert nur dann einen GS-Zugriff, wenn sich dadurch der Modus gewährter Sperren auf Rechnerebene reduziert (z.B. braucht nur das Freigeben der letzten Lesesperre mitgeteilt zu werden). Insgesamt hat sich also durch diese einfache Maßnahme die Anzahl der GS-Zugriffe bereits auf rund die Hälfte reduziert. Auch bei expliziter Objekt-identifizierung (3.1.2) werden durch den Wegfall der Sekundäreinträge pro Sperre i.a. mehr als 4 GS-Zugriffe eingespart.

Wünschenswert wäre nun natürlich auch, im GS die Abspeicherung einer Liste wartender Sperranforderungen zu umgehen. Dies ist jedoch schwieriger als bei den gewährten Sperren, da eine faire Behandlung wartender Anforderungen ohne 'Verhungern' einzelner TA (starvation) sicherzustellen ist. Eine Reduzierung von TA-spezifischen auf rechnerspezifische Angaben (ähnlich wie für gewährte Sperren) muß hier die zeitliche Reihenfolge der Anforderungen ebenso wie den Sperrmodus berücksichtigen (z.B. um TA mehrerer Rechner gleichzeitig eine Lesesperre gewähren zu können).

Eine denkbare Realisierungsform wäre pro Rechner und Sperrmodus nur einen *Warteeintrag* in der globalen Sperrtabelle vorzusehen; da die Anzahl der Sperrmodi und der Rechner begrenzt sind, ergibt sich so eine maximale Anzahl von Warteeinträgen. Ist #M die Anzahl der Sperrmodi, dann sind maximal #M*NMAX Warteeinträge zu unterscheiden. Zusätzlich ist für jeden dieser Warteeinträge die Position innerhalb der Warteliste zu verwalten. Eine Möglichkeit, wartende Sperranforderungen demgemäß im GS darzustellen, sieht folgendermaßen aus:

*#WE: 1 .. #M*NMAX; (* aktuelle Anzahl von Warteeinträgen *)*
GWL: array [1 .. #WE] of [CM-ID, Modus]; (globale Warteliste *)*

Ein Warteeintrag besteht also aus der Identifikation des CM (Rechners) sowie dem Sperrmodus; die Reihenfolge innerhalb der globalen Warteliste ist implizit gegeben. Der Speicherplatzbedarf für die gezeigte Datenstruktur ist natürlich von NMAX sowie #M abhängig. Für NMAX=4 sind für die GWL (#WE) bei zwei Sperrmodi (RX-Verfahren) nur 3 B (3 Bit) vorzusehen, bei drei Sperrmodi (z.B. RAX-Verfahren) 6 B (4 Bit). Insgesamt läßt sich also auch bei drei Sperrmodi (und maximal vier CM) die reduzierte globale

Sperrinformation bezüglich gewährten und wartenden Sperranforderungen in einem 8 B-Eintrag ablegen. Bei expliziter Objektidentifizierung wird noch (mindestens) ein weiterer 8 B-Eintrag für die Speicherung des Objektnamens erforderlich. Da außerdem gemäß dem Realisierungsvorschlag in 3.1.2 noch 2 B zur Verkettung der Hash-Klassen benötigt werden, reichen bei expliziter Objektidentifizierung nur für ein RX-Verfahren zwei Einträge pro Hash-Klasse (Objekt) aus; bei drei Sperrmodi wären dagegen schon drei Einträge vorzusehen (erhöhte Zugriffshäufigkeit).

Die Verwendung der eingeführten Datenstrukturen verdeutlicht das Beispiel in Bild 5 für drei Rechner (NMAX=4) und das RX-Sperrverfahren. Dabei ist für das betrachtete Objekt O einer TA (T1) in Rechner CM3 eine X-Sperre gewährt, während TA in allen Rechnern auf die Zuteilung einer Lesesperre warten. Darüber hinaus liegen in Rechner CM1 noch zwei X-Anforderungen für O vor (T3 und T7). Wie zu erkennen liegen auf TA-Ebene (in den lokalen Sperrtabellen) 6 wartende Sperranforderungen vor, jedoch nur 4 Einträge auf Rechnerebene innerhalb der globalen Warteliste GWL. So ist z.B. die lokale Warteliste in CM1 nur zum Teil im GS repräsentiert, da pro Rechner nur zwei Einträge in der GWL unterschieden werden. Das bedeutet einerseits, daß bei nicht leerer lokaler Warteliste eine Sperranforderung nur dann sogleich zu einem GS-Zugriff führt, wenn zu dem angeforderten Modus nicht schon ein Warteeintrag in der GWL vorliegt. Andererseits ist dafür Sorge zu tragen, daß bei Abarbeitung der lokalen Warteliste nur lokal eingetragene (wartende) Anforderungen baldmöglichst an den GS weitergereicht werden.

So wird z.B. bei FIFO-Abarbeitung der globalen Warteliste GWL nach Freigabe der gesetzten X-Sperre zunächst eine R-Sperrgewährung von CM3 nach CM2 geschickt und der GS-Eintrag entsprechend angepaßt (GRANTED-MODE := 0R00; #WE := 3; erstes GWL-Element entfernen). In CM2 werden daraufhin natürlich beide R-Anforderungen befriedigt, obwohl die zweite R-Anforderung nicht mehr im GS gespeichert wurde. Nach Freigabe der letzten R-Sperre greift CM2 auf die globale Sperrtabelle zu und veranlaßt die weitere Abarbeitung der globalen Warteliste, wobei dann die X-Anforderung aus CM1 (T3) befriedigt wird. Erst nach Entfernen des zugehörigen X-Warteeintrages für CM1 aus der GWL kann nun die bereits länger gestellte X-Anforderung von TA T7 an den GS weitergereicht werden, wobei ein erneuter X-Warteeintrag für CM1 an das aktuelle Ende der GWL angefügt wird.

Bild 5: Beispielszenarium zur Verwaltung wartender Sperranforderungen

Bei einer Implementierung könnten natürlich verschiedene Strategien zur Abarbeitung der Wartelisten verfolgt werden (z.B. Vorziehen von R-Anforderungen), die hier aber nicht näher zu diskutiert werden brauchen. Das Beispiel sollte lediglich verdeutlichen, daß auch mit der reduzierten Warteinformation im GS eine faire Bedienung wartender Sperranforderungen möglich ist. Damit konnte nun erreicht werden, daß in der globalen Sperrtabelle im GS für ein Objekt keine variablen Datenstrukturen (Listen) mehr zu verwalten sind, sondern nur noch ein einfaches Feld fester Länge (keine Verkettung). Dabei genügt bei impliziter Objektidentifizierung i.a. ein einziger 8 B-Eintrag aus, während anderenfalls größere (mehrere) Einträge erforderlich werden.

Die Führung lokaler Sperrtabellen gestattet natürlich jedem CM eine Erkennung lokaler Deadlocks, so daß nur noch globale Deadlocks einer gesonderten Behandlung bedürfen. Am einfachsten wäre für diese natürlich eine Timeout-Strategie oder eine Deadlock-Vermeidung (z.B. durch BOT-Zeitstempel, Zeitintervalle o.ä.) [Ra88a]. Eine Erkennung globaler Deadlocks kann entweder durch ein nachrichtenbasiertes Protokoll oder unter Nutzung geeigneter Datenstrukturen im GS erfolgen. Allerdings wäre ein globaler Wartegraph im GS mit 8 B-Einträgen nur äußerst umständlich zu realisieren und würde mühsame und häufige Änderungsoperationen verursachen.

Ein weiterer Vorteil der eingeführten Redundanz ist, daß die globale Sperrinformation im GS durch die Angaben in den lokalen Sperrtabellen rekonstruierbar ist (Fehlertoleranz). So können z.B. auch nach einem Rechnerausfall die GS-Sperreinträge, die der ausgefallene Rechner in Bearbeitung hatte, durch die Sperrangaben in den überlebenden CM auf einen konsistenten Zustand gebracht werden. Das eigentliche Problem, nämlich die Erkennung der betroffenen Einträge, wird damit jedoch nicht gelöst; hierzu muß i.a. ein Doppeltführen der globalen Sperrtabelle in Kauf genommen werden.

Insgesamt konnte durch den Wegfall der variablen Sperrangaben (TA-Einträge) im GS die Gesamtzugriffshäufigkeit zum GS signifikant reduziert werden. So werden bei starrer Organisation der globalen Sperrtabelle statt etwa 10 lediglich 4 bis 5 GS-Zugriffe pro Sperre erforderlich; bei doppelter Führung der Sperrtabelle fallen pro Sperre weitere 2 (statt mindestens weiteren 4) GS-Zugriffe an. Damit ergibt sich für 1000 TPS und 10 Sperren pro TA für die Synchronisation eine GS-Gesamtauslastung von etwa 5 % bzw. 7 % bei einfacher bzw. doppelter Führung der globalen Sperrtabelle.

Bei flexibler Tabellenorganisation fallen die relativen Verbesserungen wegen der größeren Primärinformationen pro Objekt (2 Einträge) etwas geringer aus. So verbleiben (für ein RX-Sperrverfahren) pro Sperre etwa 6 bis 10 GS-Zugriffe bei einfacher Speicherung der globalen Sperrtabelle, bei Doppelspeicherung kommen zusätzliche 4 GS-Zugriffe hinzu. Damit ergibt sich nun rund die doppelte GS-Zugriffshäufigkeit als bei starrer Tabellenorganisation.

3.2.2 Lokale Sperrgewährung durch die CM

Durch die Auslagerung von dynamischen Datenstrukturen wie oben diskutiert konnte die Zugriffshäufigkeit zum GS bereits deutlich reduziert werden. Dennoch ergibt sich v.a. bei expliziter Objektidentifizierung und doppelter Speicherung der globalen Sperrtabelle bereits eine relativ hohe GS-Auslastung, welche bei TA-Raten von mehreren tausend TPS (bzw. wenn der GS auch für weitere Aufgaben genutzt wird) zu Engpässen führen kann. Eine weitergehende Einsparung von GS-Zugriffen ist nun durch ein erweitertes Sperrprotokoll möglich, mit dem eine lokale Sperrgewährung durch die CM unterstützt werden soll und welches analog zu Vorschlägen zur Reduzierung globaler Sperranforderungen bei nachrichtenbasierten Sperrprotokollen für DB-Sharing arbeitet. Dort war die Begrenzung globaler Sperranforderungen v.a. zur Limitierung des Kommunikations-Overheads sowie zur Vermeidung signifikanter Antwortzeiterhöhungen erforderlich. Diese Aspekte spielen hier eine untergeordnete Rolle, da GS-Zugriffe um zwei bis drei Größenordnungen schneller abzuwickeln sind als globale Sperranforderungen bei loser Kopplung und mit einfachen Befehlen (ohne Prozeßwechsel) realisiert werden. Der hauptsächliche Grund für die Betrachtung der Optimierungen, welche eine erhebliche Komplexitätssteigerung verursachen, liegt hier also in der Reduzierung der GS-Auslastung.

Die angesprochenen Optimierungen sind Nutzung einer PCA-Verteilung, Einsatz eines Sole-Interest-Konzeptes sowie einer Leseoptimierung, welche alle auf der Nutzung von Lokalität im Referenzverhalten von TA basieren. Während die Nutzung einer PCA-Verteilung bei Verwendung zentraler Sperrinformationen im GS zur Reduzierung von GS-Zugriffen nicht einsetzbar ist, ist sowohl die Übertragung des Sole-Interest-Konzeptes als auch der Leseoptimierung möglich. In beiden Fällen wird einem Knoten (CM) eine Autorisierung zur lokalen Gewährung von Sperren übertragen, wodurch sich die Anzahl globaler Sperranforderungen (Anzahl der GS-Zugriffe) verringern läßt. So erlaubt eine Leseautorisierung, die gleichzeitig mehreren Rechnern gewährt sein kann, die lokale Behandlung von Anforderungen sowie Freigaben von Lesesperren. Bei Sole-Interest dagegen, das für ein Objekt jeweils nur einem CM zuerkannt werden kann, ist eine vollkommen lokale Sperrbehandlung (Lese- und Schreibsperren) möglich. Die Leseautorisierungen werden zurückgenommen, sobald eine X-Anforderung gestellt wird; Sole-Interest dagegen muß aufgegeben werden, sobald eine Sperranforderung von einem anderen Rechner kommt.

Es ist möglich entweder beide Formen der Autorisierungen zu unterstützen (wodurch das komplexeste Protokoll entsteht) oder nur eine der beiden Optimierungen, welche zudem auf verschiedenen Sperrgranulaten anwendbar sind (z.B. Block und Area). Empirische Untersuchungen für lose gekoppelte DB-Sharing-Systeme [Ra88a] haben gezeigt, daß aufgrund von Lokalität i.a. deutlich mehr globale Sperranforderungen eingespart werden als Verzögerungen durch den Entzug der Autorisierungen entstehen. Dies gilt v.a. für die Leseautorisierungen, die sich insbesondere bei kurzen Lesesperren als äußerst wirkungsvoll erwiesen. Bei Sole-Interest ist dagegen mit häufigerem Besitzrechtwechsel zu rechnen, insbesondere mit Zunahme der Rechneranzahl.

Bei Verwendung eines GS ist zu bedenken, daß die Optimierungen nur die vergleichsweise billigen GS-Zugriffe einzusparen helfen. Dagegen ist der Entzug von Sole-Interest bzw. einer Leseautorisierung im Vergleich zu einem GS-Zugriff erheblich aufwendiger, da hierzu Interprozessor-Kommunikationen zwischen den beteiligten CM erforderlich werden. Dies gilt auch für die 'schnelle Interprozessor-Kommunikation' unter Nutzung des GS. Denn setzt man wie in [Kö88a] für 2 SEND-/RECEIVE-Operationen, die zum Entzug einer Autorisierung mindestens anfallen (beim Entzug einer Leseautorisierung sind ggf. mehrere CM betroffen), 5000 Instruktionen an, dann verursachen diese bei 25 MIPS-Prozessoren bereits eine Unterbrechung von 200 Mikrosekunden, also den 100-fachen Aufwand wie zum Setzen einer Sperre in der globalen Sperrtabelle des GS (2 GS-Zugriffe). Dazu kommen natürlich noch die eigentlichen GS-Zugriffe zum Austausch der Nachrichten.

Da die expliziten Aufforderungen unmittelbar die Antwortzeit der TA, deren Sperranforderung den Entzug verursacht hat, erhöhen, ist zu befürchten, daß die Antwortzeiterhöhungen durch diese Entziehungen i.a. schwerer wiegen als die potentiellen Einsparungen durch die lokale Gewährung von Sperren; außerdem ist aufgrund der erhöhten Verfahrenskomplexität ein größerer Instruktionsbedarf für die Sperrbearbeitung in den CM zu erwarten. Allerdings kann (bei NOFORCE) der Entzug von Sole-Interest möglicherweise häufig mit der Anforderung einer geänderten Seite kombiniert werden (s. Kap. 4), so daß in diesem Falle (im Gegensatz zum Entzug von Leseautorisierungen) keine zusätzliche Verzögerung entsteht. Daneben sollte sich trotz dem aufwendigen Entzug der Autorisierungen durch die lokale Vergabe von Sperren die GS-Zugriffshäufigkeit i.a. deutlich reduzieren lassen, was letztlich das Hauptziel der Verfahrenserweiterung darstellt.

Zur Darstellung des globalen Synchronisationszustandes eines Objektes können im wesentlichen die in 3.2.1 vorgeschlagenen Angaben verwendet werden, so daß im GS dazu keine zusätzlichen Felder einzuführen sind. Z.B. kann vereinbart werden, daß Sole-Interest für Rechner P dadurch gekennzeichnet ist, daß gilt:

GRANTED-MODE (P) = X;   GRANTED-MODE (i) = 0 für i <> P;   #WE = 0;

Analog läßt sich die Vergabe mehrerer Leseautorisierungen durch GRANTED-MODE = R für zwei oder mehr Rechner sowie #WE = 0 kennzeichnen.

Bei der LOCK-Bearbeitung sind nun v.a. Wechsel zwischen den verschiedenen Synchronisationszuständen zu realisieren. So führt z.B. bei vorliegendem Sole-Interest für P eine Leseanforderung eines anderen Rechners zur Abstufung von Sole-Interest zu einer Leseautorisierung (sofern nicht in P Schreibanforderungen vorliegen) bzw. zur vollkommenen Rückgabe der Autorisierung. Eine X-Anforderung muß stets bis zur Rückgabe sämtlicher Leseautorisierungen verzögert werden, wobei dann dem Rechner, von dem die X-Anforderung stammt, ggf. Sole-Interest zugewiesen werden kann (falls unterdessen nicht weitere Anforderungen anderer Rechner eingetroffen sind). Man erkennt bereits, daß in dem erweiterten Verfahren eine Vielzahl neuer Sonderfälle zu behandeln sind, auf die hier im Detail nicht eingegangen werden kann. Weitere Einzelheiten bezüglich eines solchen Verfahrens bei loser Kopplung sind in [Ra88a, Sch88] zu finden.

3.3 Diskussion

Die Ablage globaler Sperrinformationen im GS verspricht eine weit effizientere Synchronisation für DB-Sharing als bei loser Rechnerkopplung, da dann durch globale Sperranforderungen die Antwortzeiten kaum erhöht werden (< 1 ms) und der Kommunikationsaufwand i.a. deutlich reduziert wird (Kommunikation zwischen den CM fällt u.a. noch für die Aktivierung wartender TA an). Allerdings eignet sich der GS nur wenig zur Realisierung variabler Datenstrukturen, da hierzu nur 8 B-Einträge zur Verfügung stehen (Verzeigerung von GS-Einträgen, umständliche Listenoperationen, Verwaltung eines Eintrag-Pools etc.). Aus diesem Grund sollte, wie in 3.2.1 skizziert, im GS pro Objekt nur eine reduzierte Sperrinformation fester Länge abgelegt werden, zumal dadurch die GS-Zugriffshäufigkeit deutlich abgesenkt werden kann.

Für die Realisierung der globalen Sperrtabelle kommen im wesentlichen eine starre Tabellenorganisation mit impliziter Objektidentifizierung sowie eine Hash-Tabelle mit expliziter Speicherung des Objektnamens in Frage. Die starre Organisation erlaubt zwar die einfachere Verwaltung sowie eine geringere Häufigkeit von GS-Zugriffen, sie verlangt jedoch für jede DB-Seite einen Kontrolleintrag im GS zu speichern, obwohl typischerweise nur ein Bruchteil davon tatsächlich benötigt wird. Dadurch wird bereits für die globale Sperrtabelle i.a. ein prohibitiv hoher Speicherplatzbedarf im GS eingeführt, insbesondere bei doppelter Speicherung der Tabelle. Sinnvoller erscheint daher die flexiblere Organisationsform über eine Hash-Tabelle, bei dem nur für tatsächlich zu sperrende Objekte Kontrollinformationen im GS verwaltet werden. Dazu sollten jedoch größere Einträge als 8 B angeboten werden, da der Name des Objektes nun explizit in der globalen Sperrtabelle zu speichern ist. Eine Verwendung von mehreren (zwei) 8 B-Einträgen pro Objekt ist, wie in 3.1.2 gezeigt, zwar auch möglich, führt aber zu einem mehrfachem Zugriffsaufwand. Eine Vereinfachung für die Sperrbearbeitung innerhalb der CM ergibt sich bereits, wenn mehrere benachbarte 8 B-Einträge im GS mit einem Befehl gelesen bzw. geschrieben werden könnten, auch wenn dabei die Zugriffszeiten entsprechend höher als beim Zugriff auf einen einzigen Eintrag wären.

Wenn sämtliche Sperrgewährungen und -freigaben über die globale Sperrtabelle abgewickelt werden, ergibt sich v.a. bei doppelter Speicherung der Tabelle und TA-Raten von über 1000 TPS eine zu hohe GS-Zugriffshäufigkeit für die Synchronisation (Engpaßgefahr). Zur Reduzierung der GS-Zugriffshäufigkeit kann entweder auf ein nachrichtenbasiertes Synchronisationsverfahren (ohne globale Sperrtabelle im GS) wie das Primary-Copy-Verfahren ausgewichen werden, oder aber das Sperrverfahren wird derart erweitert, daß die CM lokal Sperren vergeben können, z.B. mittels eines Sole-Interest-Konzeptes oder/und einer Leseoptimierung (Vergabe von Leseautorisierungen). In letzterem Fall ergibt sich jedoch ein sehr komplexes Protokoll (*), wobei neben der Verwendung der globalen Sperrtabelle im GS die Synchronisation wieder verstärkt über Nachrichtenaustausch zwischen den CM erfolgt, so z.B. zum Entzug sowie der Gewährung von Autorisierungen. Da v.a. Sole-Interest-Zuweisungen häufig sehr instabil sind, ist keineswegs sicher, ob in diesem Fall noch weniger Kommunikationsvorgänge als z.B. beim Primary-Copy-Sperr-verfahren auftreten.

Eine andere Möglichkeit zur Umgehung potentieller Engpaßsituationen ist die Speicherung der globalen Sperrtabelle in mehreren, unabhängigen GS (sofern möglich), etwa über eine Partitionierung des Namensraumes. Eine solche Aufteilung wäre auch bei der Realisierung eines globalen Systempuffers (Kap. 5) anwendbar.

4. Verwendung des GS zur Behandlung von Pufferinvalidierungen

Zur korrekten TA-Verarbeitung ist sicherzustellen, daß die TA trotz Pufferinvalidierungen in den lokalen Systempuffern (LSP) nur auf die aktuellste Version der Daten zugreifen. Dazu ist erforderlich, daß veraltete Seiten in den LSP entweder vermieden oder zumindest rechtzeitig vor einem Zugriff entdeckt werden [Ra88b]. Wird dies erreicht, dann ist im Falle einer FORCE-Strategie bezüglich des Ausschreibens auf Platte [HR83] das Veralterungsproblem bereits gelöst, da hier (bei einer Synchronisation auf Seitenebene) die aktuelle Version einer Seite stets von Platte eingelesen werden kann. Eine solche Strategie ist jedoch, wie erwähnt, für Hochleistungssysteme nicht tragbar.

Bei NOFORCE gilt es, geänderte Seiten zwischen den Rechnern auszutauschen (z.B. über den GS), um den Zugriff auf die jeweils aktuellsten Daten zu ermöglichen. Zur Realisierung dieses Änderunsaustausches kommen im wesentlichen zwei Alternativen in Betracht, wobei einer nur beim Primary-Copy-Sperrverfahren anwendbar ist [Ra88b]:

- Ein allgemeiner Ansatz ist ein *Propagate-on-Demand-Schema*, bei dem stets genau ein Rechner für eine (gegenüber der Kopie auf Platte) geänderte Seite zuständig ist und diese auch (aktuell) in seinem Systempuffer hält (i.a. ist dies der Rechner, der die letzte Änderung der Seite vorgenommen hat). Dieser Rechner ist für die Weitergabe der Seite an die anderen CM zuständig, ebenso für das Ausschreiben der Seite auf Platte (um sicherzustellen, daß keine veralteten Seiten ausgeschrieben werden). Der Name dieses Rechners kann hier im Rahmen der globalen Sperrtabelle im GS hinterlegt werden (*), so daß beim Erwerb einer Sperre über den GS bekanntgemacht wird, von welchem Knoten ggf. die aktuelle Version der Seite anzufordern ist. Kennzeichnend bei diesem Ansatz ist, daß

* Dabei ergibt sich wegen der Verwaltung der globalen Sperrtabelle im GS (Realisierung der Hash-Tabelle, Verwaltung von Warteeinträgen) auch eine höhere Komplexität gegenüber ähnlichen Verfahren bei loser Kopplung, wo die globale Sperrtabelle im Hauptspeicher eines der Knoten (zentraler Lock-Manager) verwaltet wird.

* Für eine geänderte Seite kann bei NMAX=4 z.B. durch 2 Bit der zuständige Rechner angezeigt werden sowie durch ein weiteres Bit, ob die Kopie auf Platte bereits aktualisiert wurde oder nicht. Diese Angaben können in der globalen Sperrtabelle noch ohne zusätzliche Einträge untergebracht werden.

geänderte Seiten in der Regel explizit angefordert werden müssen (Propagate-on-Demand), wozu Interprozessor-Kommunikation notwendig wird.

- Bei dem nachrichtenbasierten Primary-Copy-Sperrverfahren, bei dem keine globale Sperrtabellen im GS geführt werden, kann vorgesehen werden, daß der PCA-Rechner für sämtliche Seiten seiner Partition für das Ausschreiben und Weitergeben von geänderten Seiten verantwortlich ist [Ra88a,b]. Dazu wird bei Änderungen außerhalb des PCA-Rechners die geänderte Seite selbst mit Freigabe der für die Änderung benötigten X-Sperre an den PCA-Rechner geschickt (hier z.B. über den GS), so daß dieser stets die aktuellste Version von Seiten seiner Partition besitzt. Ein Vorteil dabei ist, daß nun geänderte Seiten nicht mehr durch eigene (zusätzliche) Nachrichten anzufordern sind, sondern dies mit Anforderung der Sperre beim PCA-Rechner verbunden werden kann. Ebenso schickt der PCA-Rechner quasi zusammen mit der Sperrgewährung die aktuelle Seite ggf. an den anfordernden Rechner bzw. teilt mit, daß die aktuelle Seitenversion von Platte eingelesen werden kann.

In 4.1 sollen zunächst die verschiedenen Methoden zur Erkennung bzw. Vermeidung von veralteten Seiten in den LSP genauer diskutiert werden. In 4.2 wird dann untersucht, wie der Austausch geänderter Seiten über den GS realisiert werden kann.

4.1 Erkennung oder Vermeidung von Pufferinvalidierungen

Die einfachste Methode, veraltete Seiten in den LSP zu erkennen bzw. zu vermeiden, ist die sogenannte *synchrone Broadcast-Lösung* [Ra88a,b]. Dabei sendet jede Update-TA bei EOT eine Broadcast-Nachricht an alle Knoten mit der Angabe, welche Seiten sie geändert hat. Die Schreibsperren dürfen erst freigegeben werden, wenn alle Knoten quittiert haben, daß sie die Broadcast-Meldung empfangen und ggf. veraltete Seitenkopien aus ihrem LSP entfernt haben. Diese Lösung, die nur bei einer Synchronisation auf Seitenebene funktioniert, hat den Nachteil eines sehr hohen Kommunikations-Overheads, der mit zunehmender Rechneranzahl ansteigt. Zudem ergeben sich durch das synchrone Warten auf die Quittierungen Antwortzeiterhöhungen für Update-TA sowie vermehrte Sperrkonflikte (längeres Halten der Sperren).

Diese Nachteile können mit zwei anderen Verfahren umgangen werden, die besser in das Synchronisationsprotokoll integriert sind und die Tatsache ausnutzen, daß eine TA vor dem Zugriff auf ein Objekt eine ausreichende Sperre anzufordern hat. Der erste Ansatz verwendet sogenannte Haltesperren zur gänzlichen Vermeidung von Pufferinvalidierungen, während im zweiten Verfahren veraltete Seiten in den LSP möglich sind und eine Seite erst beim Zugriff auf ihre Aktualität hin überprüft wird (check-on-access).

- *Vermeidung von Pufferinvalidierungen mit Haltesperren*
  Die Verwendung von Haltesperren wurde zuerst für das erweiterte Pass-the-Buck-Protokoll zur integrierten Behandlung des Veralterungsproblems vorgeschlagen ([HR85, Ra88a]), ist jedoch auch auf andere Sperrverfahren bei DB-Sharing übertragbar. Liegen dabei nach Freigabe einer regulären Sperre bei TA-Ende keine weiteren Sperranforderungen vor, erwirbt der Rechner für die betreffende Seite eine Haltesperre (in [Kö88a] Leihkennzeichen genannt) für die Zeit, in der die Seite im LSP residiert. Diese Haltesperren, bei denen man z.B. analog zu R- und X-Sperren zwischen HR- und HX-Sperren unterscheiden kann, sichern die Aktualität der lokalen Seitenkopie zu und müssen bei der Vergabe regulärer Sperren beachtet werden. So ist auch gewährleistet, daß kein unbemerktes Ändern einer Seite in einem anderen Rechner erfolgt. Vielmehr müssen vor einer Änderung alle Haltesperren in den anderen CM explizit aufgegeben und die zugehörigen Seiten aus den Puffern entfernt werden, was jedoch als Nachteil des Verfahrens gelten muß. Andererseits gestatten die Haltesperren

eine lokale Sperrgewährung durch die CM, da z.B. bei einer HR-Sperre in den anderen Knoten höchstens R-Sperren gewährt sein können; bei HX sind externe Sperrgewährungen sogar völlig ausgeschlossen. Es handelt sich bei HR-Sperren also de facto um Leseautorisierungen und bei HX-Sperren um Sole-Interest-Zuweisungen, mit denen Lokalität im Referenzverhalten zur Einsparung globaler Sperranforderungen (von GS-Zugriffen) nutzbar wird.
Deshalb kommt auch die Haltesperren-Technik v.a. bei Verwendung dieser Autorisierungen in Frage, wobei (wie erwähnt) ein komplexes Protokoll entsteht. Der Seitenaustausch zwischen den Rechnern entspricht einem Propagate-on-Demand-Schema, wobei i.a. die HX-Sperre den für eine geänderte Seite 'zuständigen' Rechner anzeigt. Zu beachten ist dabei, daß beim Ent-fernen einer geänderten Seite aus einem LSP aufgrund eines Haltesperrenentzuges die 'Verantwortung' für das Ausschreiben bzw. Weitergeben der Seite an den anfordernden Rechner zu übertragen ist [Ra88b], da für eine geänderte Seite jederzeit einer der Knoten zuständig sein muß.
Der Entzug einer HX-Sperre fällt hier also mit dem Anfordern der geänderten Seite zusammen, so daß in diesem Fall durch die Haltesperre keine zusätzliche Verzögerung entsteht. HR-Sperren (Leseautorisierungen) führen dagegen zu Verzögerungen von Schreibzugriffen, welche natürlich umso höher ausfallen, je geringer der Anteil von Lesezugriffen und je geringer die Lokalität im Referenzverhalten sind. Zur Vermeidung unnötiger Verzögerungen sollte nach Verdrängen einer Seite aus dem LSP die zugehörige HR-Sperre baldmöglichst aufgegeben werden; wird eine Seite mit HX-Sperre zur Verdrängung aus dem LSP bestimmt, so ist die Seite vor Aufgabe der Sperre auszuschreiben.

- *Entdeckung veralteter Seiten bei der Sperranforderung*
  Hierbei werden, wie erwähnt, veraltete Seiten in den LSP zugelassen und erst beim Zugriff auf die Seite - im Rahmen der Sperrgewährung - über die Aktualität einer im LSP befindlichen Kopie entschieden (*on-request invalidation* [HS88]). Dies kann durch Erweiterung der globalen Sperrinformationen, z.B. Änderungszähler (Zeitstempel) oder Invalidierungsvektoren (für NMAX=4 pro Seite 4 zusätzliche Bit), geschehen [Ra88a] und verursacht keinerlei Interprozessor-Kommunikation. Ein Nachteil, der i.a. jedoch weniger ins Gewicht fällt, ist, daß durch die spätere Entdeckung veralteter Seiten Pufferrahmen in den LSP unnötig lang besetzt bleiben und daher etwas geringere Trefferraten als bei sofortiger Eliminierung invalidierter Seiten verursachen [Hs88].

  Dieser Ansatz empfiehlt sich v.a. bei Sperrverfahren, die kein Sole-Interest-Konzept verwenden, z.B. dem Primary-Copy-Sperrverfahren. Eine Nutzung einer Leseoptimierung ist dabei jedoch problemlos möglich, da eine Leseautorisierung die Aktualität einer lokalen Seitenkopie garantiert (da eine Änderung der Seite in einem anderen Knoten durch die Leseautorisierung ausgeschlossen wird). Auch hier sollte dann die Leseautorisierung aufgegeben werden, nachdem die zugehörige Seite aus dem Puffer verdrängt wurde. Denn bei einem erneuten Zugriff muß ohnehin auf die globale Sperrinformation zugegriffen werden, um festzustellen, wo die aktuelle Seite vorliegt (Propagate-on-Demand) bzw. mit dem PCA-Rechner Kontakt aufgenommen werden, damit dieser ggf. die aktuelle Version der Seite zustellen kann.

Zur Realisierung eines Propagate-on-Demand-Schemas reichen im GS-Eintrag bei NMAX=4 3 Bit pro Seite aus (s.o.) Die Vermeidung von Pufferinvalidierungen über Haltesperren erfordert keine zusätzlichen Informationen in der globalen Sperrtabelle, wenn man eine HX- bzw. HR-Sperre analog wie eine Sole-Interest-Situation bzw. die Vergabe von Leseautorisierungen (s. 3.2.2) realisiert. Bei der Entdeckung von Pufferinvalidierungen (on-request invalidation) genügen bei Invalidierungsvektoren 4 zusätzliche Bit im Sperrkontrollblock des GS.

Insgesamt reicht also bei vier Rechnern etwa 1 B zur Speicherung der Informationen zur Behandlung des Veralterungsproblems. Zur Vermeidung zusätzlicher GS-Zugriffe sollten diese Angaben möglichst im Rahmen der globalen Sperrtabelle abgelegt werden, was auch ohne zusätzliche Einträge möglich ist. Dazu genügt bei impliziter Objektidentifizierung ein 8 B-Eintrag pro Objekt, bei Verwendung einer Hash-Tabelle werden bei einem RX-Verfahren zwei Einträge (sonst drei) pro Hash-Klasse bzw. Objekt erforderlich. Zu beachten ist, daß nun die Anzahl von Kontrollblöcken stark zunehmen dürfte, da für alle geänderte DB-Seiten die Veralterungsinformationen zu führen sind, auch wenn keine Sperranforderungen vorliegen. Bei großen Systempuffern von z.B. 100 MB pro CM sind so bei 2 KB-Seiten und einem Anteil geänderter Seiten von 50 % bereits 100.000 geänderte Seiten zu verwalten.

4.2 Austausch geänderter Seiten über den GS

Bei den bisher diskutierten Verwendungsformen des GS zur Synchronisation bzw. zur integrierten Erkennung/Vermeidung von Pufferinvalidierungen wurde lediglich das Zugriffsgranulat 'Eintrag' genutzt. Hier soll nun betrachtet werden, wie bei einer NOFORCE-Auschreibstrategie geänderte DB-Seiten zwischen zwei CM über den GS ausgetauscht werden können, wozu natürlich der GS-Seitenbereich (bestehend aus 4 KB-Seiten) genutzt wird. Die beim Seitenaustausch im einzelnen anfallenden Schritte können im Prinzip auch zur Realisierung eines allgemeineren Nachrichtenaustausches (Interprozessor-Kommunikation) über den GS Anwendung finden.

Soll eine Seite (Nachricht) X über den GS von CM1 nach CM2 gebracht werden, dann sind im wesentlichen drei Schritte vorzunehmen: 1.) CM1 schreibt X in den GS 2.) CM1 schickt eine Benachrichtigung an CM2 (Interprozessor-Kommunikation), daß X vom GS eingelesen werden kann. 3.) CM2 liest X vom GS in den Hauptspeicher (lokalen Systempuffer).

Die Realisierung eines allgemeinen Nachrichtenaustausches über den GS mit diesem Schema hat offenbar den Nachteil, daß neben den Zugriffsoperationen zum GS weiterhin eine Interprozessor-Kommunikation (Schritt 2) notwendig ist. Ein Vorteil ergibt sich also nur dann, wenn diese Benachrichtigung sehr viel effizienter (geringer Instruktionsbedarf, kein Prozeßwechsel) als eine 'allgemeine' Nachrichtenübertragung realisiert werden kann (z.B. per Interrupt).

Zur Realisierung eines Seitenaustausches über den GS, auf den sich die folgende Diskussion beschränkt, sind neben den drei genannten Schritten noch zusätzliche *Verwaltungsaufgaben bezüglich des GS-Seitenbereiches* zu lösen, insbesondere

- Suchen eines freien Seitenrahmens im GS, in den die DB-Seite hineingeschrieben werden kann
- Sichern des benutzten GS-Seitenrahmens vor Überschreiben durch einen anderen CM (FIX)
- Freigeben des GS-Seitenrahmens (UNFIX) nachdem die DB-Seite vom Empfänger eingelesen wurde.

Wie in Kap. 2 schon erwähnt, kommen für diese Verwaltungsaufgaben unterschiedliche Strategien in Frage, welche entweder zentrale Datenstrukturen im GS (die wieder durch 8 B-Einträge zu realisieren sind) oder, bei einer Partitionierung des GS, Datenstrukturen in den CM benutzen.

Partitionierte GS-Verwaltung

Bei einem *partitionierten Ansatz* kann z.B. vorgesehen werden, daß jeder CM nur auf eine, von ihm verwaltete GS-Partition schreibend zugreifen kann, lesend jedoch auf alle Partitionen (*).

In diesem Falle kann ein CM die zur Verwaltung seiner Partition benötigten Datenstrukturen (z.B. Freispeicherangaben) im Hauptspeicher halten; ein Überschreiben von GS-Seiten durch andere CM ist auch nicht mehr zu befürchten. Wird von einem anderen CM eine Seite angefordert, dann kann mit den Hauptspeicher-Datenstrukturen ein freier Platz in der lokalen GS-Partition bestimmt werden und die gewünschte DB-Seite dorthin ausgeschrieben werden. Die GS-Adresse, an der die Seite abgelegt wurde, wird dann zusammen mit der Aufforderung zum Einlesen der Seite an den anfordernden CM geschickt. Ein Nachteil bei diesem Schema, neben der Notwendigkeit einer (starren) GS-Partitionierung, ist, daß das erfolgte Einlesen der Seite aus dem GS eigens quittiert werden muß (Interprozessor-Kommunikation), damit der GS-Rahmen wieder freigegeben werden kann. Allerdings können diese Quittierungen asynchron und möglicherweise gebündelt übertragen werden.

Eine weitergehende Nutzung des GS-Seitenbereiches bei partitionierter Verwaltung ergibt sich, wenn ein CM den von ihm beschreibbaren GS-Teil nicht nur zum Ablegen angeforderter Seiten verwendet, sondern quasi als *Erweiterung des lokalen Systempuffers*. In diesem Falle ergeben sich ähnliche Vorteile wie bei einem globalen Systempuffer (Kap. 5), insbesondere schnelleres Verdrängen geänderter Seiten aus dem LSP sowie, wegen der vergrößerten Pufferkapazität, weniger Lesevorgänge von Platte. Als weiterer Vorteil kommt hinzu, daß sich eine für einen anderen CM bereitzustellende Seite nun möglicherweise bereits im GS befindet und daher die Verzögerung zum Ausschreiben der Seite in den GS (obiger Schritt 1) entfällt. Ein solcher Ansatz ist natürlich auch für nachrichtenbasierte Synchronisationsverfahren wie dem Primary-Copy-Protokoll anwendbar.

Verwendung zentraler Datenstrukturen im GS

Eine flexiblere Nutzung des GS-Seitenbereiches ist offenbar erreichbar, wenn jeder CM jede GS-Seite überschreiben kann. In diesem Fall ist jedoch die Freispeicherverwaltung sowie die Zugriffssynchronisation durch Datenstrukturen im GS zu realisieren, wodurch für diese Aufgaben zusätzliche GS-Zugriffe anfallen. Die GS-Verwaltung zur Organisation des Seitenaustausches ist jedoch, etwa im Vergleich zur Realisierung eines globalen Systempuffers (Kap. 5), sehr einfach, da zu einem Zeitpunkt i.a. nur wenige geänderte DB-Seiten gleichzeitig zwischen zwei verschiedenen CM ausgetauscht werden dürften, so daß für den Seitenaustausch auch nur wenige GS-Seiten bereitgehalten werden müssen. Sieht man z.B. vor, daß nur 64 GS-Seiten für den Austausch geänderter DB-Seiten zur Verfügung gestellt werden, so reicht ein 8 B-Eintrag im GS zur Freispeicherverwaltung (und Zugriffssynchronisierung) dieses Seitenbereiches aus. Vor dem Ausschreiben einer angeforderten Seite wird dieser Freispeichereintrag gelesen, eine freie GS-Seite belegt und damit zugleich vor weiteren Schreibzugriffen geschützt (Setzen des zugehörigen Bit) sowie der geänderte Eintrag in den GS zurückgeschrieben. Analog kann der Empfänger-CM das Freispeicher-Bit nach dem Einlesen der DB-Seite wieder zurücksetzen, so daß für die Verwaltungsaufgaben insgesamt *4 zusätzliche GS-Zugriffe* genügen. Bei der zu erwartenden Austauschfrequenz geänderter DB-Seiten, dürfte die Freispeicherinformation i.a. keinen Engpaß darstellen. Zudem lassen sich durch Verwendung mehrerer Einträge zur Freispeicherverwaltung mögliche Engpaßsituationen leicht entschärfen.

---

* Nach [GS86] ist eine derartige GS-Partitionierung realisierbar, da der GS-Seitenbereich in verschiedene 'Regions' eingeteilt werden kann, für die unterschiedliche Zugriffsrechte für die einzelnen CM festlegbar sind (z.B. nur lesender Zugriff).

Der Vorteil gegenüber dem partitionierten Ansatz liegt v.a. darin, daß zum Freigeben einer GS-Seite keine Interprozessor-Kommunikation erforderlich wird. Dafür ist eine weitergehende Nutzung des GS z.B. als Erweiterung des lokalen Systempuffers nicht so einfach möglich (s. Kap. 5). So kann mit der vorgeschlagenen Methode eine in den GS gebrachte DB-Seite i.a. auch nur zur Befriedigung einer Seitenanforderung benutzt werden.

Zusammenspiel mit der Synchronisation

Der Austausch geänderter Seiten über den GS mit einer der vorgestellten Realisierungsformen kommt, wie erwähnt, nur bei NOFORCE in Betracht; bei einer FORCE-Strategie in einen globalen Systempuffer im GS (Kap. 5) oder auf Platte kann die aktuelle Seite stets direkt aus dem globalen Puffer bzw. von Platte in den LSP eingelesen werden. Die Bereitstellung einer aktuellen Seite bei NOFORCE erfolgt dabei, ebenso wie die Erkennung bzw. Vermeidung von Pufferinvalidierungen (4.1), im Zusammenspiel mit dem Synchronisationsprotokoll; z.B. wird bei einem Propagate-on-Demand-Schema in der globalen Sperrtabelle vermerkt, wo die aktuelle Version einer geänderten Seite erhältlich ist.

Bei einem *Primary-Copy-Verfahren*, bei dem geänderte Seiten durch den PCA-Rechner verwaltet werden (s.o.), sind zwei Arten des Seitenaustausches zu unterscheiden:

- Bei einer Seitenänderung außerhalb des PCA-Rechners wird die geänderte Seite beim EOT der Update-TA zunächst in den GS geschrieben; danach wird eine Nachricht an den PCA-Rechner geschickt, mit dem die X-Sperre freigegeben wird und mit der die GS-Adresse der geänderten DB-Seite übergeben wird. Der PCA-Rechner liest daraufhin die Seite in seinen Hauptspeicher ein und führt die Freigabe der X-Sperre durch.
- Wird bei einer Sperranforderung einer externen TA festgestellt, daß in deren CM die zu referenzierende Seite nicht oder nur veraltet vorliegt, dann kann der PCA-Rechner die aktuelle Seitenversion (sofern sie bei ihm vorliegt) im Rahmen der Sperrgewährung übergeben. Dazu muß die betreffende Seite natürlich nur dann vor Abschicken der Sperrgewährung (mit Angabe der GS-Adresse, von wo die Seite einzulesen ist) in den GS gebracht werden, sofern sie nicht schon dort vorliegt, z.B. bei Nutzung des GS als Erweiterung des lokalen Systempuffers.

In beiden Fällen wird also zur Mitteilung der GS-Adresse, von der die Seite gelesen werden kann, keine eigene Nachricht benötigt. Außerdem entfallen eigene Nachrichten zum Anfordern einer geänderten Seite.

Im folgenden wird nun die *Propagate-on-Demand-Alternative* noch etwas näher betrachtet, wobei angenommen wird, daß eine globale Sperrtabelle im GS zur Synchronisation sowie der integrierten Behandlung des Veralterungsproblemes verwendet wird. Dabei wird beim Anfordern einer Sperre durch CM2 über die globale Sperrtabelle im GS die Notwendigkeit einer Seitenanforderung bei einem anderen Knoten CM1 erkannt: dies ist der Fall, wenn für die zu referenzierende Seite im GS ein Sperreintrag vorliegt, der anzeigt, daß die Seite seit dem letzten Einlesen von Platte geändert wurde, jedoch im eigenen Puffer nicht aktuell (sondern im Puffer des verantwortlichen Rechners CM1) vorliegt, oder wenn die Seite gerade in CM1 geändert wird (Sperrkonflikt). Betrachten wir zunächst den letzteren Fall, bei dem der Sperrkonflikt mit einer Änderungs-TA auf CM1 erkannt und daher ein Warteeintrag für CM2 in der globalen Sperrtabelle im GS eingetragen wird. Der Vorteil dabei ist, daß bei Freigeben der X-Sperre durch CM1 nun die Sperranforderung seitens CM2 erkannt wird; CM1 kann daher die geänderte Seite zunächst in den GS schreiben und CM2 die GS-Adresse zusammen mit der Sperrgewährung mitteilen (*). In diesem Fall wird also der Aufwand einer expliziten Seitenanforderung vermieden. Außerdem wird zur Mitteilung der GS-Adresse keine eigene Nachricht benötigt, da dies mit der Gewährung der Sperre kombiniert wird. Die einzelnen Schritte sind im folgenden noch einmal zusammengefaßt.

CM1	CM2
	LOCK (Sperranforderung im GS hinterlegen)
UNLOCK freier GS-Rahmen bestimmen GS-WRITE Seite SEND Sperrgewährung + GS-Adresse	
	RECEIVE GS-READ Seite GS-Rahmen freigeben

Aufwendiger ist dagegen der Fall, in dem bei der Sperranforderung kein Konflikt vorliegt und die aktuelle Seite explizit bei CM1 anzufordern ist. Hier wird sowohl zum Anfordern der Seite als auch zur Mitteilung der GS-Adresse, an welcher die Seite vorliegt, eine separate Benachrichtigung (Interprozessor-Kommunikation) erforderlich:

CM1	CM2
	LOCK SEND Seitenanforderung
RECEIVE freier GS-Rahmen bestimmen GS-WRITE Seite SEND GS-Adresse	
	RECEIVE GS-READ Seite GS-Rahmen freigeben

Zu beachten ist, daß bei der Verwendung von Haltesperren zur Vermeidung von Pufferinvalidierungen auch im Falle eines Konfliktes aufgrund einer gesetzten Haltesperre (bzw. bei Sole-Interest oder vergebener Leseautorisierung) eine explizite Anforderung der Seite sowie der Sperre notwendig wird.

* Dieser optimierte Seitenaustausch kann prinzipiell auch bei Sperrkonflikten mit Lesesperren verwendet werden, z.B. X-Anforderung bei gesetzten R-Sperren. In diesem Fall müßte jedoch in der globalen Sperrinformation vermerkt werden, daß der Rechner, der die X-Anforderung gestellt hat, die aktuelle Seite noch nicht in seinem lokalen Puffer hält (bzw. es wird generell davon ausgegangen, daß dem so ist).

5. Einsatz eines globalen Systempuffers

Die Verwendung des GS-Seitenbereichs als globaler Systempuffer (GSP) sieht vor, daß aus den LSP verdrängte bzw. ausgeschriebene DB-Seiten aller CM im GS aufgenommen und bei Bedarf wieder in die LSP eingebracht werden können. Lokale Systempuffer werden neben dem GSP weiterhin benötigt, da auf dem GS keine Instruktionsadressierbarkeit vorliegt. Zudem kann auf Daten im Hauptspeicher (LSP) erheblich schneller als auf den GS zugegriffen werden und durch Nutzung von Lokalität in den CM können Platten- und GS-Zugriffe eingespart werden.

Der Einsatz eines GSP verspricht erhebliche Leistungsvorteile:

- Mit dem GSP können *Lesevorgänge von Platte eingespart* werden, da für im GS vorliegende DB-Seiten der langsame Plattenzugriff vermieden wird. So kann bei einer geeigneten Verwaltung des GSP nicht nur CM-spezifische Lokalität (innerhalb der LSP) zur Reduzierung von Plattenzugriffen genutzt werden, sondern auch systemweite Lokalität. Dies wurde in eingeschränktem Maße bereits durch den Austausch geänderter Seiten über den GS (4.2) erreicht, kann über den GSP jetzt aber auch für ungeänderte Seiten genutzt werden.
- Durch die Nichtflüchtigkeit und die schnellen Zugriffszeiten des GS bedingt können *Ausschreibvorgänge aus den LSP erheblich beschleunigt* werden (Faktor 1000). Dies ist v.a. für bezüglich der TA-Bearbeitung synchrone E/A-Vorgänge zur Reduzierung der Antwortzeiten und Sperrdauern von Bedeutung, z.B. für E/As während der EOT-Behandlung (FORCE, Logging (Kap. 6)). Für asynchrone Ausschreibvorgänge dagegen, etwa zum vorsorglichen Verdrängen geänderter Seiten im Rahmen der LSP-Verwaltung (NOFORCE), lohnt sich ein Schreiben in den GSP nur, falls eine erneute Referenzierung der Seite wahrscheinlich ist (dies ist z.B. für die Account-Seiten bei Debit-Credit nicht der Fall). Denn da das Ausschreiben vom GSP auf Platte über die Hauptspeicher der CM erfolgen muß (Kap. 1), würden ansonsten unnötige GS-Zugriffe eingeführt.
- Auch die *Anzahl der Schreibzugriffe auf Platte kann reduziert werden*, da eine Seite vor einem Ausschreiben auf Platte in mehrerern Rechnern geändert werden kann (systemweites Akkumulieren von Seitenänderungen). Ein schneller Austausch von Änderungen ist dabei v.a. gegeben, wenn eine geänderte Seite direkt vom GSP eingelesen werden kann und nicht erst bei einem anderen CM anzufordern ist (4.2).

Auf der anderen Seite lassen sich diese Vorteile nur nutzen, wenn die Verwaltung des GSP durch die CM zufriedenstellend gelöst werden kann; darauf wird in 5.2 näher eingegangen. Zunächst soll jedoch eine andere zentrale Entwurfsentscheidung hinsichtlich der Realisierung eines GSP diskutiert werden, nämlich die Wahl der Ausschreibstrategie (FORCE oder NOFORCE).

5.1 FORCE oder NOFORCE

Ohne GSP fällt die Wahl der Ausschreibstrategie ziemlich leicht, da ein FORCE-Ansatz bezüglich der materialisierten DB auf Platte einen unvertretbar hohen E/A-Aufwand verursacht und die Antwortzeiten von Update-TA stark erhöht. Wegen der starken Beschleunigung von Ausschreibvorgängen durch den GS kommt der FORCE-Strategie (write through) nun jedoch wieder verstärkte Bedeutung zu, insbesondere da in anderen Punkten Vorteile gegenüber der NOFORCE-Alternative (write in) bestehen:

- So kann bei einer geeigneten Implementierung des GSP erreicht werden, daß bei FORCE nur aktuelle DB-Seiten im GS vorliegen und der GSP für Lesezugriffe als echte Erweiterung der Speicherhierarchie zwischen LSP und Platte aufgefaßt werden kann. Dabei kann für eine Seitenreferenz, die im LSP nicht befriedigt werden kann, stets direkt der GSP inspiziert werden. Liegt die

DB-Seite dort vor, ist ihre Aktualität gesichert und kann in den LSP eingelesen werden; anderenfalls wird die Seite von Platte eingelesen. Es entfallen also die bei NOFORCE notwendigen Seitenanforderungen bei anderen Rechnern und die damit verbundenen Verzögerungen.

- Als noch wesentlicher ist anzusehen, daß bei FORCE die Notwendigkeit einer REDO-Recovery nach einem Rechnerausfall entfällt, da wegen die Nichtflüchtigkeit des GS die in ihn geschriebenen Änderungen erfolgreicher TA weiter z.V. stehen. Bei NOFORCE dagegen müssen nach einem Rechnerfall alle Änderungen des gescheiterten CM, die noch nicht in den GSP bzw. in die materialisierte DB auf Platte gebracht wurden, nachgefahren werden, was sehr langwierig sein kann (Kap. 6).

NOFORCE führt also zu einer aufwendigeren Behandlung des Veralterungsproblems (Berücksichtigung von Seiten des GSP, explizite Seitenanforderungen bei Propagate-on-Demand) sowie von Rechnerausfällen. Andererseits stellt die hohe Ausschreibfrequenz geänderter Seiten bei FORCE immer noch einen signifikanten Nachteil dar:

- So wird z.B. bei 4 geänderten Seiten pro TA von je 1000 TPS eine GS-Auslastung von 6 % allein für das Ausschreiben der Seiten (zuzüglich der GS-Zugriffe zur Verwaltung) verursacht; bei 25-MIPS-Prozessoren verursachen die synchronen Schreibzugriffe ein Instruktionsäquivalent von 1500 Instruktionen pro TA (1.5 MIPS pro 1000 TPS).
- Aufgrund der hohen Schreibfrequenz in den GSP verkürzt sich die durchschnittliche Verweildauer einer Seite im Vergleich zu einem NOFORCE-Ansatz, so daß Lokalität im Referenz-verhalten weit weniger stark genutzt werden dürfte (die meisten der ausgeschriebenen Seiten werden i.a. von anderen CM nicht benötigt).
- Durch die verkürzte Verweilzeit sowie dem vermutlich hohen Anteil geänderter Seiten im GSP, ergibt sich ein hohes Ausschreibvolumen vom GSP auf Platte. Da dies über die Hauptspeicher der CM zu geschehen hat, ergibt sich daraus eine hohe Belastung der CM sowie eine ähnliche starke GS-Belastung für das Einlesen der auf Platte zu bringenden Seiten wie für deren vorheriges Ausschreiben in den GS. Eine Alternative wäre, jede Seite parallel auf Platte und in den GS auszuschreiben; in diesem Fall werden jedoch keinerlei Ausschreibvorgänge auf Platte durch Akkumulierung von Änderungen eingespart (hoher I/O-Overhead).

Die letztgenannten Punkte lassen eine NOFORCE-Strategie auch bei Verwendung eines GS noch vielversprechender erscheinen, auch wenn der (hoffentlich seltene) Fehlerfall dabei schwieriger und aufwendiger zu behandeln ist.

5.2 Verwaltung des globalen Systempuffers

Bei der Verwaltung eines GSP sind v.a. die folgenden Aufgaben zu bewältigen:

- Lokalisieren von DB-Seiten im GSP (feststellen ob und wo sich Seite X im GSP befindet).
- Synchronisierung von Zugriffen auf den GSP
  So ist z.B. zu vermeiden, daß während eines Ausschreibens in den GSP ein anderer CM diesselbe GS-Seite liest bzw. beschreibt.
- Freispeicherverwaltung des GSP
- Ersetzung von DB-Seiten im GSP
  Die Bestimmung von Ersetzungskandidaten sollte möglichst unter Berücksichtigung von Lokalitätsaspekten erfolgen, um hohe Trefferraten im GSP (wenige Plattenzugriffe) er-reichen zu können.

- Behandlung von Pufferinvalidierungen im GSP (Vermeidung oder Erkennung)
- Auswahl von Seiten der LSP, welche in den GSP verdrängt werden
  Am einfachsten ist natürlich, jede aus dem LSP zu verdrängende bzw. auszuschreibende DB-Seite in den GSP zu bringen; wird die Seite im GSP jedoch nicht mehr referenziert, war der Ausschreibvorgang umsonst und die GS-Platz wurde schlecht genutzt (geringere GSP-Trefferraten). Wenn möglich sollten daher zur Reduzierung des Ausschreibvolumens und Verbesserung der GSP-Trefferraten nur solche Seiten in den GSP geschrieben werden, für die eine Wiederverwendung wahrscheinlich ist (typbezogene Entscheidung). Für geänderte Seiten ist festzulegen, ob bei einem Ausschreiben in den GS gleichzeitig das Ausschreiben auf Platte angestoßen werden soll oder nicht.
- Ausschreibkoordinierung für geänderte Seiten im GSP
  Für geänderte DB-Seiten im GSP ist zu regeln, welcher CM das Ausschreiben auf Platte vorzunehmen hat. Um ein Akkumulieren von Änderungen einer Seite zu ermöglichen, ist zwar eine möglichst lange Verweilzeit geänderter Seiten im GS wünschenswert; andererseits ist durch ein vorausschauendes Ausschreiben zu gewährleisten, daß stets ungeänderte GSP-Seiten zur Ersetzung zur Verfügung stehen (um Verzögerungen beim Einbringen in den GSP zu vermeiden).

Für die Lösung dieser Verwaltungsaufgaben kommen wieder unterschiedliche Ansätze in Betracht, wobei GS-Datenstrukturen und/oder Hauptspeicher-Datenstrukturen der CM einsetzbar sind. Im folgenden werden dazu drei Realisierungsansätze betrachtet: zuerst wird auf die schon in Kap. 4 angesprochene Einsatzform mit ausschließlicher Verwendung von Hauptspeicher-Datenstrukturen eingegangen, danach wird eine GSP-Verwaltung mit zentralen Datenstrukturen im GS diskutiert und abschließend eine gemischte Strategie aus partitionierter Verwaltung und Einsatz von GS-Datenstrukturen. Der erste Ansatz ist dabei im wesentlichen auf eine NOFORCE-Strategie beschränkt, während ansonsten die FORCE-Alternative auch in Frage kommt.

5.2.1 Vollkommen partitionierte Verwaltung

Eine einfache Lösung der Verwaltungsaufgaben wird möglich bei einer vollkommen partitionierten Verwaltung des GS-Seitenbereiches, wobei dieser lediglich als *Erweiterung der lokalen Systempuffer* benutzt wird (4.2). Dabei wird jedem CM eine GS-Partition zugeordnet, in die nur er Schreibzugriffe vornehmen kann; Lesezugriffe durch andere CM sind jedoch zulässig, um einen Austausch von DB-Seiten zwischen den Rechnern zu ermöglichen. In diesem Fall kann jeder CM seine GS-Partition durch lokale Hauptspeicher-Datenstrukturen praktisch wie den lokalen Systempuffer verwalten, insbesondere die Freispeicherverwaltung, die Bestimmung von Ersetzungskandidaten oder das vorausschauende Ausschreiben geänderter Seiten. Die Organisation des Seiten-austausches mit anderen CM erfolgt gemäß 4.2, wobei geänderte Seiten entweder explizit (Propagate-on-Demand) oder zusammen mit dem Lock-Request (Primary Copy) angefordert werden und nach dem Einlesen der im GS bereitgestellten Seite diese durch eine asynchrone Quittierungsnachricht freigegeben wird.

Mit dieser eingeschränkten Form eines globalen Systempuffers wird ein schnelles Ausschreiben aus den LSP ebenso möglich wie eine Einsparung von Lese- und Schreibvorgängen von/auf Platte. Allerdings kann die GS-Kapazität durch die starre Partitionierung i.a. nicht optimal genutzt werden, zumal eine Seite in mehreren GS-Partitionen repliziert vorliegen kann. Auch wird eine systemweite Lokalität nur für geänderte Seiten zum Einsparen von Plattenzugriffen nutzbar, da i.a. nur geänderte Seiten zwischen den CM ausgetauscht werden.
Der Ansatz ist für FORCE weniger von Interesse, da dann trotzdem Pufferinvalidierungen möglich sind und Änderungen explizit angefordert werden müssen. Die Behandlungen von Pufferinvalidierungen bei NOFORCE ist unproblematisch, da hierzu die DB-Seiten der lokalen GS-Partition wie Seiten im LSP be-

handelt werden können. Bei einem Primary-Copy-Verfahren, bei dem Änderungen von nicht lokal synchronisierten Seiten stets zum PCA-Rechner gebracht werden, können veraltete Seiten bzw. Mehrfachspeicherungen im GS vermieden werden, indem jeder Knoten nur Seiten seiner Partition in den GS bringt. In diesem Fall ist der Seitenaustausch zwischen CM auch nicht notwendigerweise auf geänderte Seiten beschränkt, da der PCA-Rechner auch ungeänderte Seiten (die in seinem LSP bzw. GS-Anteil vorliegen) mit der Sperrgewährung zur Verfügung stellen kann (*).

5.2.2 Verwendung zentraler Datenstrukturen im GS

Die flexibelste Nutzung des GS-Seitenbereichs als globaler Systempuffer wird sicherlich möglich, wenn jeder Rechner in den gesamten GSP schreiben darf und DB-Seiten nicht repliziert im GS vorliegen. Dies erfordert jedoch eine Verwaltung des GSP über globale Datenstrukturen im GS. Im folgenden werden für die einzelnen Verwaltungsaufgaben (Lokalisieren von DB-Seiten, Zugriffssynchronisation, Seitenersetzung, Ausschreibkoordinierung und Behandlung von Pufferinvalidierungen im GSP) mögliche Realsierungsansätze unter Verwendung von GS-Datenstrukturen diskutiert.

Lokalisierung von DB-Seiten im GSP

Die Lokalisierung (Suche) von DB-Seiten im GSP läßt sich relativ einfach mit Hilfe einer *Seitenzuordnungstabelle (SZT)* [Kö88a] lösen. Die SZT realisiert dabei eine Zuordnung von DB-Seiten zu GS-Seitenrahmen, d.h. für jede im GSP vorliegende DB-Seite ist die zugehörige GS-Adresse (sowie ggf. weitere Statusinformationen bezüglich der DB-Seite, z.B. ein Änderungsvermerk) in der SZT gespeichert. Zur Organisation der Tabelle stehen die gleichen Alternativen wie bei der globalen Sperrtabelle (Kap. 3) zur Auswahl, also entweder eine starre Tabellenstruktur mit impliziter Objektidentifizierung oder eine Hash-Tabelle mit expliziter Speicherung der DB-Seiten-IDs (mindestens zwei 8 B-Einträge pro Hash-Klasse). Auch hier ist trotz der höheren GS-Zugriffshäufigkeit bei 8 B-Einträgen letzterer Ansatz vorzuziehen.

Zugriffssynchronisation und Ersetzung von GSP-Seiten

Da der Zugriff auf die SZT stets über die Nummer der zu referenzierenden DB-Seite erfolgt, eignet sich diese Tabelle nicht zur Verwaltung der GS-Seiten, insbesondere bezüglich der Zu-griffssynchronisation auf GS-Seiten sowie zur Speicherung von Informationen zur Freispeicher-verwaltung und Seitenersetzung. Diese Angaben sind in einer eigenen GS-Tabelle zu verwalten, in der zu jeder GS-Seite des GSP geeignete Angaben fester Länge zu verwalten sind. Wesentlich ist dabei, daß für eine GS-Seite die Angaben zur Zugriffssynchronisation (Sperrvermerk von 1 Bit je GS-Seite), mit denen ein paralleles Lesen und (Über-) Schreiben derselben GS-Seite zu verhindern ist, sowie zur Seitenersetzung möglichst zusammen erreicht werden, da z.B. keine Seiten ersetzt werden dürfen, die sich im Zugriff befinden. Ferner sehen wir vor, daß gegenüber der Platte geänderte Seiten im GSP nicht zur Ersetzung

* Bei einem Propagate-on-Demand-Ansatz ist der Austausch ungeänderter Seiten prinzipiell auch erreichbar. Nur müßte in diesem Fall für jeden Rechner die komplette Pufferbelegung auf Tabellen im GS (bzw. innerhalb der globalen Sperrtabelle) abgebildet werden, um auch feststellen zu können, wo Kopien ungeänderter Seiten vorliegen. Die Folge wäre ein stark erhöhter Wartungsaufwand für die globale Sperrtabelle (häufigere GS-Zugriffe), da jeder Einlesevorgang bzw. jede Verdrängung eine Anpassung der GS-Information erfordern würde. Zudem würde sich natürlich die Anzahl der Kontrollblöcke (Speicherplatzbedarf) beträchtlich erhöhen.

verwendet werden. Ersetzbar bzw. überschreibbar sind demnach freie (unbelegte) GS-Seiten sowie ungeänderte GSP-Seiten, für die von keinem CM ein Zugriff angemeldet ist.

Ein einfacher Ansatz, mit dem jedoch keine Lokalitätsaspekte für die Ersetzung von Seiten berücksichtigt werden, sieht für jede GS-Seite neben dem Sperrvermerk ein weiteres Bit vor, das anzeigt, ob die Seite mit einer geänderten DB-Seite belegt ist. Eine GS-Seite kann dann nur überschrieben werden, wenn beide Bits ungesetzt sind, d.h. die GS-Seite nicht blockiert ist sowie eine veraltete oder ungeänderte DB-Seite enthält bzw. noch unbelegt ist (*). So reichen z.B. für einen GSP von 512 MB (128.000 GS-Seiten) zur Speicherung dieser Bits 4000 Einträge (von je 8 B) aus (32 KB). Zur Bestimmung eines Ersetzungskandidaten greift dann ein CM auf einen dieser Einträge zu (derart, daß alle Einträge etwa gleich häufig berücksichtigt werden), wobei dann i.d.R. mindestens eine der 32 einem Eintrag zugeordneten GS-Seiten ersetzbar sein sollte.

Mit diesen Verwaltungsinformationen fallen für das *Lesen einer DB-Seite aus dem GSP* folgende Schritte an:

- Feststellen der GS-Adresse über die SZT (2 Mikrosekunden bei expliziter Objektidentifizierung)
- Zugehörigen GS-Rahmen blockieren (2 Mikrosek.) - Lesen der Seite (15 Mikrosek.)
- Freigeben des GS-Rahmen (2 Mikrosek.)

Beim *Schreiben einer Seite in den GSP* sind folgende Schritte zu durchlaufen:

- Feststellen mit SZT, ob Seite bereits in GSP vorliegt (2 Mikrosek.)
- Wenn ja (und zu schreibende Seite geändert), blockiere zugehörigen GS-Rahmen (2 Mikrosek.); sonst bestimme Ersetzungskandidat (2 Mikrosek.)
- Seite in GSP schreiben (15 Mikrosek.)
- SZT-Information anpassen (3-4 Mikrosek.)
- GS-Rahmen freigeben und ggf. Änderungsvermerk setzen (2 Mikrosek.)

Insgesamt werden also zum Lesen bzw. Schreiben einer Seite bei Verwendung zentraler GS-Datenstrukturen bereits mindestens 21 bis 25 Mikrosekunden für GS-Zugriffe erforderlich.

Hauptmanko der skizzierten Ersetzungsstrategie ist natürlich, daß das tatsächliche Referenzverhalten bei der Ersetzung von GSP-Seiten vollkommen unberücksichtigt bleibt, so daß niedrige Trefferraten im GSP bzw. nur geringe Einsparungen von Plattenzugriffen zu befürchten sind. Verfeinerte Techniken, bei denen mehr als 2 Bit pro GS-Seite zur Auswahl eines Ersetzungskandidaten benötigt werden, sind zwar auch denkbar, bewirken jedoch auch einen erhöhten Zusatzaufwand.
Eine Möglichkeit wäre z.B. ein *LFU-Verfahren* (least frequently used [EH84]), bei dem pro GS-Seite ein Zähler für die Zugriffshäufigkeit geführt wird und jeweils diejenige der nicht blockierten GS-Seiten mit den wenigsten Zugriffen zur Verdrängung aus dem GSP ausgewählt wird (Zählerstand 0 kann dabei für unbelegte GS-Seiten und der höchstmögliche Zählerwert für geänderte Seiten verwendet werden). Diese erweiterten Informationen (z.B. 1 B pro GS-Seite für Sperrvermerk und Referenzierungszähler) führen nicht nur zu einem erhöhten Speicherplatzbedarf, sondern v.a. zu einem hohen Aufwand zur Bestimmung eines Ersetzungskandidaten. Denn da hierzu die Zähler sämtlicher Seiten auszuwerten

---

* Ein einziges Bit pro GS-Seite für Zugriffssynchronisation und Auswahl einer zu ersetzenden Seite reicht nicht aus, da z.B. geänderte Seiten zwar nicht ersetzbar sein sollen, dennoch aber (Lese-) Zugriffe auf sie durchführbar sein müssen. Bei nur einem Bit wäre eine nicht ersetzbare GS-Seite jedoch auch für Lesezugriffe blockiert.

wären, käme selbst bei einer Ablage der Ersetzungsinformationen in GS-Seiten ein enormer Leseaufwand zusammen. Für einen GSP von 512 MB würden bei 1 B pro GS-Seite z.B. 32 GS-Seiten für die Ersetzungsinformation benötigt, deren komplettes Einlesen (480 Mikrosekunden) bei jedem Ausschreibvorgang natürlich nicht tolerierbar ist. Wird nur jeweils eine der 32 GS-Seiten ausgewertet, wird dieser Aufwand zwar deutlich reduziert, jedoch wird das globale Referenzierungsverhalten auch nur noch eingeschränkt berücksichtigt. Ein großer Nachteil bei der Verwendung von GS-Seiten für die Ablage der Ersetzungsinformationen entsteht weiterhin für Lesevorgänge, da zum Setzen/Rücksetzen der Sperrvermerke sowie zum Anpassen der Zähler jeweils ganze Seiten zu lesen und ändern sind. So würden neben dem eigentlichen Lesen und Schreiben der DB-Seite jeweils 4 GS-Seitenzugriffe für Verwaltungszwecke anfallen, wobei dann insgesamt etwa 80 Mikrosekunden pro Lese- und Schreibvorgang bezüglich des GSP zusammenkämen. Ähnlich aufwendig wäre eine analoge Realisierung eines globalen LRU-Verfahrens, bei dem zu jeder GS-Seite der Zeitpunkt des letzten Zugriffs verwaltet wird. Ein vielversprechenderer Ansatz wird im folgenden vorgestellt.

Globale LRU-Seitenersetzung über GS-Datenstrukturen

Eine sinnvollere Realisierungsform eines globalen LRU-Mechanismus sieht vor, alle ersetzbaren GS-Seiten explizit in einer *LRU-Kette im GS* zu verwalten. Hierzu kann eine Tabelle benutzt werden, in der für jede GS-Seite des GSP ein 8 B-Eintrag geführt wird und die ersetzbaren GS-Seiten durch Verweise innerhalb der Tabellenstruktur (doppelt) verkettet sind. Für einen GSP von 512 MB umfaßt diese **LRU-Tabelle** z.B. 128.000 Einträge (1 MB) und 3 B reichen für einen Verweis auf einen Vorgänger bzw. Nachfolger innerhalb der LRU-Kette aus (3 B sind sogar aus-reichend für bis zu 16 Mill. GS-Seiten). Pro Eintrag der LRU-Tabelle wird neben den zwei Verweisen bezüglich der LRU-Kette (Nachfolger, Vorgänger) auch der Sperrvermerk zur Zugriffssynchronisation geführt:

VG, NF: Verweis innerhalb der LRU-Tabelle; (* je 3 B *)
Blockiert: Boolean;

Neben der LRU-Tabelle werden noch zwei spezielle Einträge LRU-HEAD und LRU-TAIL benötigt, die auf den aktuellen Anfang bzw. Ende der LRU-Kette verweisen und die ebenfalls einen Sperrvermerk führen, um die Listenoperationen zu synchronisieren. LRU-HEAD soll dabei immer auf den Eintrag derjenigen GSP-Seite verweisen, die ersetzbar ist (d.h. ungeändert und nicht im Zugriff befindlich) und auf die die längste Zeit nicht mehr zugegriffen wurde.

Mit einer solchen LRU-Tabelle gestaltet sich das *Lesen einer GSP-Seite* wie folgt:

- Feststellen der GS-Adresse der DB-Seite über SZT (2 Mikrosek.)
- Zugehörige GS-Seite blockieren (Konflikt möglich) und ggf. aus LRU-Kette ausketten:
  Eintrag X für GS-Seite lesen und blockieren; (2 Mikrosek.)
  Vorgänger-Eintrag Y lesen und anpassen: Y.NF := X.NF (2 Mikrosek.)
  Nachfolger-Eintrag Z lesen und anpassen: Z.VG := X.VG (2 Mikrosek.)
- - Lesen der Seite (15 Mikrosek.)
- - Anpassen der LRU-Kette und GS-Seite freigeben:
  Lesen und Blockieren von LRU-TAIL; (2 Mikrosek.)
  Schlußeintrag T lesen und anpassen: T.NF := X; (2 Mikrosek.)
  Eintrag X freigeben/anpassen: X.VG := T; X.NF := nil; (2 Mikrosek.)
  LRU-TAIL anpassen (zeigt nun auf X) und freigeben; (2 Mikrosek.)

Beim *Schreiben einer DB-Seite in den GSP* fallen folgende Aktionen an:

- Feststellen mit SZT, ob Seite bereits im GSP vorliegt (2 Mikrosek.)

- Wenn ja und Überschreiben der Seite erforderlich, wie oben zugehörige GS-Seite blockieren und aus LRU-Kette ausketten (6 Mikrosek.);

  ansonsten Ersetzungskandidat bestimmen und blockieren:

  Lesen und Blockieren von LRU-HEAD; (2 Mikrosek.)

  Lesen und Blockieren Starteintrag (= Ersetzungskandidat); (2 Mikrosek.)

  Nachfolger-Eintrag Z lesen und anpassen: Z.VG := nil; (2 Mikrosek.)

  LRU-HEAD anpassen (zeigt nun auf Z) und freigeben; (2 Mikrosek.)
- Seite in GSP schreiben (15 Mikrosek.);
- SZT-Information anpassen (3-4 Mikrosek.)
- Freigeben der GS-Seite und ggf. wie oben Anpassen der LRU-Kette (falls DB-Seite gegenüber Platte ungeändert und damit ersetzbar) (8 Mikrosek.)

Insgesamt fallen somit neben den GS-Zugriffszeiten von 15 Mikrosekunden zum Lesen bzw. Seiten einer GSP-Seite noch 16 bis 22 Mikrosekunden für Verwaltungsaufgaben an, insgesamt also 31 bis 37 Mikrosekunden pro Seitentransfer zwischen LSP und GSP. Der Vorteil für diesen erheblichen Verwaltungsaufwand ist, daß das globale Referenzverhalten zur Erhöhung der Trefferraten im GSP in adäquater Weise berücksichtigt werden kann.

Ausschreiben geänderter Seiten aus dem GSP

Ein weiteres Problem bei einer GSP-Verwaltung über GS-Datenstrukturen betrifft das Aus-schreiben von geänderten DB-Seiten im GSP durch die CM.

Die wohl einfachste Strategie ist, daß ein CM beim Einbringen einer geänderten Seite in den GSP zugleich das Ausschreiben auf Platte veranlaßt. Die GSP-Seite wird dann nur bis zum erfolg-reichen Ende der Platten-I/O im GSP als 'geändert' geführt und der zugehörige GS-Seiten-rahmen kann danach bei der Ersetzung berücksichtigt werden (z.B. durch Anhängen an die LRU-Kette). Die Vorteile dieser Strategie sind:

- Es liegen i.d.R. stets ersetzbare GS-Seitenrahmen vor.
- Der Zeitpunkt des Ausschreibens sowie der durchführende CM sind festgelegt.
- Zum Ausschreiben einer GSP-Seite muß diese nicht erst in den Hauptspeicher eingelesen werden.

Andererseits können damit, wie erwähnt, keine Ausschreibvorgänge auf Platte durch system-weites Akkumulieren von Seitenänderungen eingespart werden. Bei einer FORCE-Strategie können nicht einmal Änderungen einer Seite innerhalb eines Knotens akkumuliert werden, so daß ein sehr hoher Ausschreib-Overhead entsteht. Das sofortige Ausschreiben auf Platte parallel mit dem Einbringen in den GSP ist daher allenfalls bei NOFORCE zu empfehlen. Allerdings erscheint dies auch bei NOFORCE nicht generell sinnvoll zu sein. Wenn z.B. das Einbringen der geänderten Seite in den GSP aufgrund einer expliziten Anfrage (mit Änderungsabsicht) eines anderen CM erfolgt, dann wäre die auf Platte geschriebene Seitenversion ohnehin nur noch kurze Zeit gültig.

Werden geänderte DB-Seiten in den GSP, jedoch nicht auf Platte geschrieben, so ist dafür Sorge zu tragen, daß stets ungeänderte GS-Seiten zur Ersetzung vorliegen. Wünschenswert ist ferner, auch für geänderte Seiten das Referenzierungsverhalten zu berücksichtigen, um möglichst solche Seiten vorrangig auszuschreiben, die schon längere Zeit nicht mehr geändert bzw. referenziert wurden. Dazu kann analog zu der Verkettung ungeänderter GSP-Seiten eine LRU-artige Verkettung geänderter GSP-Seiten innerhalb der LRU-Tabelle realisiert werden. Dabei zeigt ein spezieller Eintrag stets auf diejenige geänderte GSP-Seite, welche die längste Zeit nicht mehr referenziert wurde, und daher als nächstes auszuschreiben ist. Nach Abschluß des Ausschreibvorganges wird die GSP-Seite ersetzbar und im Rahmen der LRU-Kette ungeänderter Seiten verwaltet. Das Ausschreiben geänderter GSP-Seiten erfolgt entweder durch einen ausgezeichneten Rechner oder, synchronisiert, durch alle.

Behandlung von Pufferinvalidierungen im GSP

Bei einer *FORCE-Auschreibstrategie* können veraltete Seiten im GSP vermieden werden, da Änderungen stets sofort durchgeschrieben werden; dabei ist sicherzustellen, daß ältere Versionen einer geänderten Seite im GSP (mit der SZT) lokalisiert und überschrieben werden. Ein Verdrängen einer ungeänderten DB-Seite in den GSP erfolgt nur, wenn die betreffende Seite nicht schon dort vorliegt (feststellbar über SZT) und die Aktualität der in den GSP zu bringenden Seitenkopie gesichert ist (aufgrund einer Haltesperre oder anderer Informationen zur Behandlung von Pufferinvalidierungen in den LSP). So kann bei FORCE stets gewährleistet werden, daß nur aktuelle DB-Seiten im GSP vorliegen. Der GSP kann daher auch als echte Erweiterung der Speicherhierarchie aufgefaßt werden, wobei für jede im LSP nicht (aktuell) vorliegende DB-Seite stets auf den GSP zugegriffen wird. Nur wenn die DB-Seite auch im GSP nicht geführt wird, erfolgt ein Zugriff auf Platte.

Bei *NOFORCE* können Pufferinvalidierungen im GSP prinzipiell vorkommen, da geänderte Seiten i.a. nicht bzw. verzögert (nach Ende der Änderungs-TA) in den GS gelangen. Veraltete GSP-Seiten können jedoch auch bei NOFORCE vermieden werden, wenn vor jedem X-UNLOCK auf die SZT zugegriffen wird, um festzustellen, ob eine (veraltete) Kopie der geänderten Seite im GSP vorliegt; ist dies der Fall, wird die invalidierte Seite aus dem GSP eliminiert. Diese Vermeidungsstrategie entspricht in etwa einer Vermeidung von Invalidierungen durch Haltesperren, nur müßte bei Haltesperren (für GSP-Seiten) eine zu ändernde Seite bereits vor Zuteilung der Schreibsperre aus dem GSP entfernt und die zugehörige Haltesperre aufgegeben werden (das Sperrprotokoll würde sich also verkomplizieren und die Gewährung von Schreibsperren verzögern). Der generelle Nachteil der Vermeidungsstrategie liegt in dem Aufwand zur Eliminierung invalidierter GSP-Seiten. Denn dazu genügt nicht eine einfache Änderung der SZT, sondern es muß auch die Ersetzungsinformation für den zugehörigen GS-Seitenrahmen angepaßt werden. So fallen bei jeder Seitenänderung neben der SZT-Zugriffszeit von 2 Mikrosekunden im Falle einer Invalidierung zusätzliche GS-Zugriffszeiten von etwa 18 Mikrosekunden (bei Verwendung einer LRU-Tabelle) für die Manipulation der GS-Datenstrukturen an. Andererseits kann damit der GSP potentiell am besten genutzt werden, da invalidierte Seiten sofort ersetzt werden können.

Wie schon bei der Behandlung von Pufferinvalidierungen in den LSP besteht auch für den GSP die Alternative, Pufferinvalidierungen zuzulassen und durch Erweiterung des Zugriffsprotokolls Zugriffe auf veraltete Seiten zu umgehen (Erkennung von Invalidierungen beim Zugriff). Eine Möglichkeit dazu besteht darin, in der SZT pro GSP-Seite ein Invalidierungsbit zu führen und bei einer Seitenänderung nur dieses (nicht jedoch die Ersetzungsinformation) anzupassen (3-4 Mikrosek. GS-Zugriffszeit); die veralteten DB-Seiten werden dann im Rahmen der normalen Seitenersetzung aus dem GSP entfernt. Eine GSP-Seite wird dann natürlich nur bei nicht gesetztem Invalidierungsbit in den LSP eingelesen.
Eine Alternative zu diesem Vorgehen ist, auch für den GSP die Behandlung von Puffer-invalidierungen mit dem Sperrprotokoll zu koppeln. Werden zur Erkennung veralteter LSP-Seiten z.B. Invalidierungsvektoren im Rahmen der globalen Sperrtabelle geführt (4.1), so können diese Vektoren um ein zusätzliches Invalidierungsbit für den GSP erweitert werden, welches dann nach einer Seitenänderung in einem der CM gesetzt wird (unabhängig davon, ob die DB-Seite im GSP vorliegt oder nicht). Beim Ausschreiben einer DB-Seite (deren Aktualität gesichert ist) in den GSP, wird für die Seite das GSP-Invalidierungsbit im Invalidierungsvektor der globalen Sperr-tabelle zurückgesetzt. Ein Zugriff auf veraltete GSP-Seiten wird nun dadurch verhindert, daß eine DB-Seite nur dann noch im GSP (über die SZT) gesucht wird, wenn der Invalidierungsvektor in der globalen Sperrtabelle anzeigt, daß die

aktuelle DB-Seite in den GSP geschrieben wurde. Der Vorteil dieses Ansatzes ist vor allem, daß er mit den wenigsten GSP-Zugriffen auskommt. Wird die globale Sperrtabelle im GS geführt, kann der Invalidierungsvektor bekanntlich (außer beim Ausschreiben einer Seite in den GSP) ohne eigene GS-Zugriffe im Rahmen der normalen Sperrbehandlung gelesen und angepaßt werden.

5.2.3 Mischform aus partitionierter und zentralisierter GSP-Verwaltung

Obwohl die Verwaltung des GSP prinzipiell über zentrale Datenstrukturen im GS realisierbar ist, so zeigte der Vorschlag mit Hilfe der LRU-Tabelle, daß z.B. die Wartung der Informationen zur Seitenersetzung bzw. bezüglich der Ausschreibreihenfolge sehr aufwendig und umständlich ist. Mit einer Mischform aus partitionierter GSP-Verwaltung (5.2.1) und Einsatz von GS-Datenstrukturen (5.2.2) kann daher versucht werden, v.a. das Ersetzen und Ausschreiben von GSP-Seiten möglichst über Hauptspeicher-Datenstrukturen (ohne GS-Zugriffe) zu regeln, allerdings unter Inkaufnahme anderer Nachteile:

Wie in 5.2.2 soll dabei im GS eine SZT zur Lokalisierung der DB-Seiten benutzt werden; ferner erfolgt die Zugriffssynchronisation der CM über GS-Datenstrukturen (1 Bit pro GS-Seite). Wie in 5.2.1 wird vorgesehen, daß jedem CM eine Partition des GSP fest zugeordnet ist, in die i.a. nur er Schreibzugriffe vornehmen darf. Daher kann auch jeder CM die aktuelle Belegung seiner Partition in lokalen Datenstrukturen verwalten sowie das Ausschreiben geänderter GSP-Seiten seiner Partition vornehmen. Da die Ersetzung von GSP-Seiten ebenfalls über lokale Datenstrukturen realisiert werden soll, kann dazu im wesentlichen jedoch nur das GSP-Referenzverhalten lokaler TA berücksichtigt werden. Denn wenn die Zugriffe auf eine GSP-Partition durch andere CM bei der Ersetzung auch Berücksichtigung finden sollen, müßten entweder wieder aufwendige GS-Datenstrukturen geführt werden oder diese Zugriffe müßten, möglicherweise gebündelt, über eigene Nachrichten an den für die Ersetzung verantwortlichen CM gemeldet werden.
Das Lesen und Schreiben bezüglich des GSP erfolgt nun im wesentlichen wie bei dem ersten Vorschlag in 5.2.2 (21-25 Mikrosekunden); insbesondere muß also während des GSP-Zugriffs der zugehörige GS-Rahmen blockiert werden. Eine Besonderheit besteht dabei hinsichtlich des Ausschreibens einer geänderten Seite, zu der in der GSP-Partition eines anderen CM eine ältere Version vorliegt. Diese Version wird entweder überschrieben und dem zuständigen CM eine entsprechende Mitteilung gesendet (zur Anpassung seiner lokalen Datenstrukturen bezüglich geänderter Seiten), oder das Ausschreiben erfolgt in die eigene Partition und die veraltete Kopie wird invalidiert (Vermerk in der SZT und Benachrichtigung des zuständigen CM). In beiden Fällen wird also eine Interprozessor-Kommunikation erforderlich, um sicherzustellen, daß zu jeder DB-Seite höchstens eine Kopie im GSP erreichbar ist.

Der Preis für die reduzierte GS-Zugriffshäufigkeit über eine derartige Mischlösung ist also (neben der starren Partitionierung), daß bei der Seitenersetzung im wesentlichen nur das Referenzverhaltens von jeweils einem CM berücksichtigt wird und daß beim Schreiben geänderter Seiten in den GSP möglicherweise Kommunikationsvorgänge erforderlich werden. Letzteres ist natürlich v.a. bei einem FORCE-Ansatz wieder von Nachteil.

5.3 Diskussion

In diesem Kapitel wurden verschiedene Alternativen zur Verwendung eines GS als globaler Systempuffer (GSP) bei DB-Sharing diskutiert, welche v.a. durch die Wahl der Ausschreibs-trategie in den GSP (FORCE oder NOFORCE) sowie die Realisierung der GSP-Pufferverwaltung (mit Hauptspeicher- oder/und GS-Datenstrukturen) gekennzeichnet sind.

Beim *FORCE-Ansatz* ist von Vorteil, daß nach einem Rechnerausfall keine REDO-Recovery erforderlich ist, daß Pufferinvalidierungen im GSP vermieden werden können und daß geänderte Seiten nicht explizit (über Nachrichten) in anderen Knoten anzufordern sind. Andererseits ergibt sich ein sehr hoher Ausschreibaufwand in den GSP sowie vom GSP auf die Platten. Außerdem läßt sich Lokalität im GSP-Zugriffsverhalten wegen der i.a. kürzeren Verweilzeit von DB-Seiten im GSP weniger zu Einsparungen an Plattenzugriffen nutzen als bei NOFORCE.
Zur GSP-Verwaltung bei FORCE scheidet der vollkommen partitionierte Ansatz (5.2.1) aus, da hierbei Pufferinvalidierungen sowie explizite Seitenanforderungen in Kauf zu nehmen wären. Bei ausschließlicher Verwendung globaler GS-Datenstrukturen ergibt sich v.a. mit einer LRU-Tabelle ein enormer Verwaltungsaufwand sowie eine entsprechend hohe GS-Zugriffshäufigkeit, die zu Engpaßsituationen führen kann. Setzt man nämlich für das Schreiben in den GS sowie das Ausschreiben aus dem GS jeweils eine GS-Zugriffszeit von 35-40 Mikrosekunden an, ergibt sich für je 1000 TPS (und 4 geänderten Seiten pro TA) bereits eine GS-Auslastung von etwa 30 % für das Einbringen und spätere Ausschreiben geänderter DB-Seiten. Daneben verursachen sämtliche Lesezugriffe, die im LSP nicht befriedigt werden können, einen Zugriff auf den GSP. Weniger GS-Zugriffe würden bei einer Seitenersetzung ohne bzw. mit eingeschränkter Berücksichtigung des globalen Referenzverhaltens enstehen, etwa wie in 5.2.2 diskutiert bzw. bei einer Mischform aus partitionierter und zentralisierter GSP-Verwaltung (5.2.3), wobei in letzterem Fall jedoch zusätzliche Nachrichten anfallen können. Ein anderer Ausweg wäre eine Verwendung mehrerer unabhängiger GS, wobei dann pro GS ein eigener GSP existiert, in dem nur DB-Seiten einer festzulegenden DB-Partition zugelassen werden, so daß sich die GS-Zugriffe entsprechend aufteilen.

Zur Realisierung eines GSP besser geeignet ist der *NOFORCE-Ansatz*, mit dem mindestens zwei vielversprechende Verwendungsformen möglich sind. Der einfachste Ansatz ist dabei eine vollkommen partitionierte GSP-Verwaltung, die vor allem bei einem nachrichtenbasierten Primary-Copy-Sperrprotokoll zu empfehlen ist, bei dem die PCA-Rechner nur Seiten ihrer Partition in ihre LSP-Erweiterung im GS bringen (5.2.1). Die flexibelste Nutzung eines GSP wird möglich bei ausschließlicher Verwendung globaler GS-Datenstrukturen (5.2.2), wobei der relativ hohe Verwaltungsaufwand mit einer LRU-Tabelle hier wegen der im Vergleich zu FORCE erheblich niedrigeren GSP-Zugriffshäufigkeit tolerierbar sein dürfte. So kann z.B. durch eine erweiterte Verwendung von Invalidierungsvektoren innerhalb der globalen Sperrtabelle erreicht werden, daß Lesezugriffe auf den GSP nur erfolgen, wenn die aktuelle Version der betreffenden DB-Seite zuvor ausgeschrieben wurde. Das Ausschreiben geänderter DB-Seiten in den GSP erfolgt entweder aufgrund einer Anforderung durch einen anderen CM oder im Rahmen der Verdrängung aus dem LSP. Von Nachteil ist die höhere Komplexität der zentralisierten GSP-Verwaltung im Vergleich zum vollkommen partitionierten Ansatz, v.a. durch die umständliche Realisierung von Datenstrukturen im GS verursacht. Weiterhin sind die GS-Datenstrukturen nach einem CM-Ausfall i.a. inkonsistent (keine atomaren Änderungen, gesetzte Zugriffssperren im GS durch den ausgefallenen CM), wobei zur Fehlerbehebung möglicherweise ein doppeltes Führen der Datenstrukturen erforderlich wird.

6. Logging und Recovery

Aufgrund der Nichtflüchtigkeit sowie der rund 1000-fach schnelleren Zugriffszeit im Vergleich zu Platten ist der GS zur Aufnahme von Log-Daten besonders zu empfehlen, zumal die Verzögerungen für das Logging i.a. unmittelbar die Antwortzeiten der Änderungs-TA (und damit die Dauer gehaltener Sperren) erhöhen. Wesentliche Einflußgrößen bezüglich Logging und Recovery bei DB-Sharing sind u.a. die Ausschreibstrategie für geänderte Seiten (FORCE oder NOFORCE), die Existenz eines globalen Systempuffers, das Log-Granulat (Seite oder Eintrag), die Realisierung einer globalen Log-Datei sowie das verwendete Protokoll für Synchronisation und Veralterungsproblem. Erste Überlegungen hierzu finden sich in [Kö88b]; auch hier kann nur auf einige Aspekte eingegangen werden.

Log-Granulat

Ein seitenorientiertes Logging, bei dem die (Before- bzw.) After-Images ganzer DB-Seiten protokolliert werden, führt zu einer hohen Log-Umfang und häufigen Schreibvorgängen. Damit würde z.B. bei NO-FORCE für das Schreiben von After-Images in den GS ein ähnlicher Schreibaufwand verursacht wie bei einer FORCE-Strategie zum Ausschreiben geänderter Seiten in den globalen Systempuffer. Aus diesen Gründen sollte das Logging stets auf Eintragsebene erfolgen, wenngleich dadurch die REDO-Recovery aufwendiger als bei Seiten-Logging wird [Re81, Re83].
Die komplexere *REDO-Recovery* ist *bei NOFORCE* nicht nur bei Plattenfehlern, sondern auch nach einem Rechnerausfall durchzuführen, wobei geänderte DB-Seiten, die zuletzt nur im Systempuffer des ausgefallenen CM vorlagen, rekonstruiert werden müssen. Für das Nachfahren der betroffenen Seiten, welche bei Verwendung einer globalen Sperrtabelle im GS durch die dort geführten Angaben zum Veralyterungsproblem bestimmt werden können, genügt es wegen dem Eintrags-Logging jedoch nicht mehr nur die lokale Log-Datei des ausgefallenen Knotens zu berücksichtigen. Denn da eine DB-Seite seit dem letzten Einlesen von Platte in mehreren Knoten geändert worden sein kann, sind auch die zugehörigen After-Images von Teilen der Seite i.a. über mehrere lokale Log-Dateien verstreut. Daher erfordert die Crash-Recovery für NOFORCE (bei Eintrags-Logging) die Berücksichtigung aller lokalen Log-Dateien bzw. es wird eine zu erstellende globale Log-Datei benutzt.

Lokales Logging in den GS

Bei NOFORCE sind bei einer NOSTEAL-Verwaltung der LSP [HR83] im wesentlichen nur After-Images in die lokale Log-Datei zu protokollieren, welche von dort aus möglicherweise auf eine globale Log-Datei gebracht werden (s.u.). Bei FORCE kann das Schreiben mehrerer geänderter DB-Seiten in den globalen Systempuffer nicht atomar vorgenommen werden; ein Zurücksetzen von gescheiterten TA wird möglich durch eine Protokollierung von Before-Images in die lokale Log-Datei. Für die REDO-Recovery nach Plattenfehlern oder nach Ausfall des globalen Systempuffers (!) wird auch für FORCE eine Protokollierung von After-Images in eine globale Log-Datei erforderlich; die globale Log-Datei wird dabei entweder direkt oder 'offline' aus den lokalen Log-Dateien erstellt (s.u.).

Die Nutzung des GS zum *lokalen Logging* der CM ist relativ einfach möglich. Dazu wird eine strikte GS-Partitionierung vorgenommen, so daß jedem CM eine GS-Partition ausschließlich zur Aufnahme seiner lokalen Log-Daten zur Verfügung steht; die Verwaltung dieser Partition kann daher auch weitgehend durch Hauptspeicher-Datenstrukturen erfolgen (im GS sind höchstens noch Angaben wie Beginn und Ende der relevanten Log-Daten zu führen, um sie für eine Crash-Recovery nutzen zu können). Sinnvollerweise werden auch bei Eintrags-Logging Seiten zur Aufnahme der (Before- und) After-Images verwendet, wobei bei einem Gruppen-Commit und kurzen TA i.a. die Log-Daten mehrerer TA in eine Seite passen. Aufgrund der begrenzten Kapazität einer GS-Partition, dient diese im wesentlichen als nicht-

flüchtiger Log-Puffer, von wo aus die Log-Daten laufend auf die Log-Dateien auf Platte weiterzuleiten sind. Um für dieses Ausschreiben der Log-Daten vom GS auf Platte die Seiten nicht erst wieder in den Hauptspeicher einlesen zu müssen, empfiehlt sich hier generell, parallel zum Logging in den GS das Ausschreiben der Log-Seiten auf die Platten-Log-Datei anzustoßen.
Setzt man pro Schreibvorgang in die lokale Log-Datei rund 20 Mikrosekunden an, so bewirkt das lokale Logging bei 1000 Änderungs-TA pro Sekunde und einer Log-Seite pro Update-TA eine GS-Auslastung von 2 %; bei Verwendung von Gruppen-Commit i.a. deutlich unter 1 %. Bei doppelter Führung der Log-Dateien, wie sinnvoll, verdoppelt sich dieser Schreibaufwand natürlich.

Erstellung einer globalen Log-Datei

Für eine möglichst schnelle REDO-Recovery nach Ausfall einer Platte oder des globalen Systempuffers sowie bei NOFORCE (mit Eintrags-Logging) nach einem Rechnerausfall empfiehlt sich die Erstellung einer globalen Log-Datei, in der die DB-Änderungen (After-Images) aller Rechner in chronologischer Reihenfolge vorliegen. Eine Möglichkeit besteht darin, die lokalen Log-Dateien auf Platte offline durch einen speziellen Prozeß oder Rechner zu mischen (und komprimieren) und auf eine globale Log-Datei auf Platte zu schreiben. Um ein einfaches Mischen zu ermöglichen, kann in einem Eintrag des GS ein Commit-Zähler geführt werden, der beim Commit einer Update-TA gelesen und inkrementiert wird (2 Mikrosek.) und der bei den After-Images abgelegt wird. Die After-Images der lokalen Log-Dateien brauchen dann nur entsprechend der Commit-Reihenfolge in die globale Log-Datei gebracht zu werden.

Eine Alternative zu dem aufwendigen und E/A-intensiven Mischen der lokalen Log-Dateien ist die *direkte Erstellung einer globalen Log-Datei*, wobei der GS als nichtflüchtiger Log-Puffer benutzt wird, in den alle CM ihre After-Images schreiben. Die Änderungen in diesem globalen Log-Puffer (GLP) im GS werden dann durch einen der CM fortlaufend auf eine globale Log-Datei auf Platte durchgeschrieben, um im GS stets Pufferrahmen zur Aufnahme von After-Images vorliegen zu haben.
Die Organisation des globalen Log-Puffers im GS ist relativ einfach und erlaubt das Einbringen sämtlicher After-Images in chronologischer Reihenfolge ohne Engpaßprobleme. Der globale Log-Puffer besteht dabei aus GLP-MAX GS-Seitenrahmen, die sequentiell (gemäß einem Ringpuffer-Prinzip) beschrieben werden. Ein spezieller 8 B-Eintrag zeigt dabei auf den zuletzt beschriebenen GS-Rahmen GLP-LAST; dieser Eintrag wird bei jedem Log-Vorgang um die benötigte Seitenanzahl erhöht und legt die GS-Seiten fest, in welche die Log-Seiten zu schreiben sind. Da die Anpassung diese EOF-Zeigers nur 2 Mikrosekunden dauert und er pro Update-TA höchstens einmal geändert wird (bei Gruppen-Commit weniger), stellt er z.B. bei 1000 Update-TA pro Sekunde keinerlei Engpaß dar. Eine Zugriffssynchronisation bezüglich des globalen Log-Puffers ist nur erforderlich zwischen dem Schreiben der After-Images in den GS sowie dem Auslesen der Log-Seiten durch denjenigen Knoten, der das Durchschreiben auf die globale Log-Datei auf Platte vornimmt. Dazu kann wieder pro GS-Seite ein Blockierungsvermerk (je 1 Bit) im Rahmen von 8 B-Einträgen verwaltet werden.
Beim Schreiben von N Log-Seiten (mit After-Images) in den globalen Log-Puffer im GS sind somit im wesentlichen folgende Schritte durchzuführen:

- Lesen GLP-LAST-Eintrag; (1 Mikrosek.)
  GLP-LAST := (GLP-LAST + N) mod GLP-MAX;
- Schreiben des geänderten GLP-LAST-Eintrages; (1 Mikrosek.)
- Blockieren der N zu überschreibenden GS-Rahmen; (2 Mikrosek.)
- Schreiben der N Log-Seiten; (N*15 Mikrosek.)
- Freigeben der GS-Rahmen; (2 Mikrosek.)

Insgesamt fallen also für das Schreiben einer Log-Seite in den globalen Log-Puffer etwa 21 Mikrosekunden an. Bei 1000 Update-TA pro Sekunde wird für diese Schreibvorgänge ohne Gruppen-Commit eine GS-Auslastung von ca. 2.1 % verursacht, mit Gruppen-Commit entsprechend weniger. Da die Log-Seiten zum Ausschreiben auf Platte wieder vom GS in den Hauptspeicher eines der CM einzulesen sind, verdoppeln sich die GS-Zugriffe für das globale Logging; eine weitere Erhöhung der GS-Belastung (auf bis zu ca. 6 %) ergibt sich beim doppelten Führen des globalen Log-Puffers. Dafür kann dann bei NOFORCE (und NOSTEAL) prinzipiell auf lokale Log-Dateien verzichtet werden (abgesehen vom Nachrichten-Logging).
Das Einlesen sämtlicher Log-Seiten sowie die CPU-Belastungen zum (sequentiellen) Ausschreiben auf Platte können von einem CM leicht bewältigt werden. Allerdings ist es selbst bei sequentiellem Schreiben (bzw. chained I/O) nur bis zu einer bestimmten TA-Rate möglich, sämtliche globale Log-Daten ausreichend schnell auf eine einzige Platte auszuschreiben, da in der Regel pro Plattenumdrehung nur ein Schreibvorgang möglich ist (werden z.B. pro Plattenumdrehung im Mittel 10 Seiten geschrieben, so können bei einer Umdrehungszeit von 15 ms nur etwa 670 Log-Seiten pro Sekunde ausgeschrieben werden). So mag für 1000 TPS eine Platte bei Eintrags-Logging und Gruppen-Commit noch ausreichen (zumal i.a. nicht nur Update-TA abzuarbeiten sind), jedoch sind hier Engpäße beim globalen Logging am ehesten zu erwarten. Eine Abhilfe wird möglich durch die Aufteilung der globalen Log-Datei auf mehrere Plattenlaufwerke, wobei pro Teildatei nur die After-Images bezüglich jeweils eines DB-Bereiches gespeichert werden. Dies erfordert dann natürlich eine Separierung der einzelnen Log-Daten durch den ausschreibenden CM; von Vorteil ist, daß im Fehlerfall nur ein Teil der globalen Log-Datei zur Rekonstruktion einer bestimmten DB-Seite zu durchlaufen ist.

Crash-Recovery

Die Behandlung von CM-Ausfällen ist neben der Ausschreibstrategie v.a. von der Verwendung von GS-Datenstrukturen zur Synchronisation und Behandlung des Veralterungsproblems abhängig. Wird die globale Sperrtabelle ausfallsicher im GS gespeichert, entfällt die aufwendige Rekonstruktion mit lokalen Datenstrukturen der überlebenden CM. Eine solche Rekonstruktion wäre zudem nicht für Angaben zum Veralterungsproblem anwendbar, deren Verlust zu einer wesentlich aufwendigeren REDO-Recovery bei NOFORCE führt. Wird jedoch im GS für geänderte Seiten hinterlegt, welcher Knoten sie in seinem LSP führt, brauchen im Fehlerfall nur die Blöcke mit der globalen Log-Datei rekonstruiert zu werden, für die der ausgefallene Rechner zuletzt 'zuständig' war.

Beim nachrichtenbasierten Primary-Copy-Sperrverfahren geht dagegen mit Ausfall eines CM die globale Sperrtabelle mit den Veralterungsinformationen verloren; hier ist daher eine REDO-Recovery für die gesamte Partition des gescheiterten Rechners vorzunehmen [Ra88a]. Bei Verwendung eines globalen Systempuffers gemäß 5.2.1, wo jeder PCA-Rechner Seiten seiner Partition in die (nichtflüchtige) LSP-Erweiterung im GS bringt, braucht nach einem CM-Ausfall für die im GS vorliegenden Seiten keine REDO-Recovery vorgenommen zu werden, da ihre Aktualität gesichert ist. Ein Durchschreiben aller Änderungen in den GS durch den PCA-Rechner würde die REDO-Recovery vermeiden (und einem FORCE-Ansatz entsprechen), erzeugt aber einen zu hohen Schreibaufwand. Zur Reduzierung des REDO-Aufwandes wäre jedoch sinnvoll, wenn jeder PCA-Rechner eine einfache GS-Datenstruktur verwaltet, in der die geänderten DB-Seiten seiner Partition vermerkt sind, die sich in seinem LSP befinden. Im Fehlerfall könnte dann die REDO-Recovery auf diese Seiten beschränkt werden.

Nach einem Rechnerausfall sind globale GS-Datenstrukturen i.a. inkonsistent (gesetzte Sperrvermerke, Änderungen wurden nur teilweise ausgeführt, etc.). Die Beseitigung dieser Inkonsistenzen wird am einfachsten bei doppelter Auslegung der GS-Datenstrukturen; anderenfalls dürfte es schwerfallen, alle betroffenen GS-Einträge zu ermitteln. Im Rahmen der Crash-Recovery müssen natürlich auch Sperren und Autorisierungen von TA des gescheiterten CM identifiziert und zurückgegeben werden, um wartenden TA den Zugriff zu ermöglichen.

7. Zusammenfassung und Ausblick

Gegenstand des Aufsatzes war eine erste Analyse möglicher Einsatzformen eines Global Storage (GS) in DB-Sharing-Systemen. Nach heutiger Einschätzung erlaubt der GS eine nahe Speicherkopplung von bis zu vier Rechnerknoten (CM) und bietet zwei Zugriffsgranulate (Seite und Eintrag). Hauptsächliche Vorteile des GS sind die Nichtflüchtigkeit sowie sehr schnelle Zugriffszeiten im unteren Mikrosekundenbereich.

Es wurde deutlich, daß der GS nahezu für alle leistungsbestimmenden Systemkomponenten eines DB-Sharing-System genutzt werden kann (Synchronisation, Behandlung von Pufferinvalidierungen, globaler Systempuffer, Lastkontrolle, Logging/Recovery). Leistungssteigerungen gegenüber lose gekoppelten DB-Sharing-Systemen sind dabei v.a. aufgrund einer effizienteren Kommunikation bzw. Kooperation der Rechner (Synchronisation, integrierte Lösung des Veralterungsproblems, Austausch geänderter Seiten) im Vergleich zu nachrichtenbasierten Protokollen sowie einem verbesserten E/A-Verhalten (globaler Systempuffer, Logging) zu erwarten. Die Behandlung von Rechnerausfällen kann ferner davon profitieren, daß im GS Datenstrukturen ausfallsicher gespeichert werden können.

Bei der Festlegung sinnvoller GS-Einsatzformen stellt sich zunächst die Frage, für welche Funktionen der GS genutzt werden soll. Wird eine möglichst umfassende Verwendung des GS angestrebt, so verspricht dies zwar den höchsten Leistungsgewinn, andererseits wird aber auch der höchste Speicherplatzbedarf für den GS (Kosten) benötigt, und es kommt am ehesten zu Engpässen bei der Gesamtzugriffshäufigkeit zum GS. Weiterhin besteht die größte Gefahr einer Fehlerausbreitung über gemeinsam benutzte GS-Datenstrukturen, z.B. infolge von Software-Fehlern in einzelnen Systemkomponenten. Daneben sind zu jeder Funktion eine Reihe alternativer Einsatzformen des GS möglich, unter denen eine Auswahl zu treffen ist. Erste Erkenntnisse und Empfehlungen hierzu werden im folgenden zusammengefaßt.

Aufgrund der schnellen Zugriffszeiten und Nichtflüchtigkeit ist ein Einsatz des GS für Logging-Zwecke generell zu empfehlen. Wie Kap. 6 zeigte, ist insbesondere die Erstellung einer globalen Log-Datei zur REDO-Recovery über den GS relativ einfach und ohne Engpaßgefahr möglich (bei Eintrags-Logging und Gruppen-Commit). Problemlos ist auch eine Beschleunigung des lokalen Logging zu erreichen.

Eine wesentliche Entwurfsentscheidung stellt die Wahl der Ausschreibsstrategie für geänderte DB-Seiten dar (FORCE oder NOFORCE), wobei der FORCE-Ansatz nur bei Verwendung des GS als globaler Systempuffer in Frage kommt. Die Überlegungen in Kap. 5 zeigten jedoch, daß mit dem FORCE-Ansatz ein globaler Systempuffer weit weniger gut zur Einsparung von Plattenzugriffen nutzbar wird als bei NOFORCE und aufgrund der hohen Ausschreibhäufigkeit ein enormer Verwaltungs-Overhead entsteht. Diese Probleme können u.E. auch durch Vorteile hinsichtlich der Recovery oder Behandlung von Pufferinvalidierungen nicht wettgemacht werden.
Für NOFORCE stellt sich die Frage, ob der GS-Seitenbereich außer für Logging nur zum Austausch geänderter DB-Seiten zwischen den CM (4.2) oder auch als globaler Systempuffer genutzt werden soll. In letzterem Fall ergeben sich darüber hinaus verschiedene Realisierungsformen (5.2), z.B. mit vollkommen partitionierter Verwaltung oder Verwendung zentraler GS-Datenstrukturen. Dabei verspricht der

Einsatz globaler GS-Datenstrukturen zur Pufferverwaltung die größten Einsparmöglichkeiten an Plattenzugriffen, jedoch ergibt sich eine komplexe und umständliche Realisierung (z.B. mit einer LRU-Tabelle).

Bezüglich Synchronisation sowie der integrierten Behandlung des Veralterungsproblems ist zu wählen zwischen einem nachrichtenbasierten Protokoll (mit beschleunigter Kommunikation über den GS) sowie Verfahren, welche eine globale Sperrtabelle im GS nutzen. In ersterem Fall kommt vor allem ein Primary-Copy-Sperrverfahren in Betracht, mit dem Pufferinvalidierungen bei der Sperranforderung (z.B. über Invalidierungsvektoren) entdeckt und Änderungen stets zum PCA-Rechner geschickt werden. Die Verwendung einer globalen Sperrtabelle im GS verspricht eine sehr effiziente globale Synchronisation, welche die Antwortzeit einer TA i.a. um deutlich weniger als 1 Millisekunde erhöht und die Abhängigkeiten zum Lastprofil bzw. der Lastverteilung verringert. Dabei sollte im GS nur eine reduzierte Sperrinformation auf Rechnerebene geführt werden, um die aufwendige und umständliche Verwaltung variabler Angaben (weitgehend) zu vermeiden. Im einfachsten Fall werden sämtliche Sperranforderungen und -freigaben über die globale Sperrtabelle im GS abgewickelt (3.2.1); es ergibt sich dann allerdings eine relativ hohe Zugriffs-häufigkeit zum GS. Dieser Nachteil wird durch Verwendung eines erweiterten Verfahrens umgangen, allerdings unter Inkaufnahme vermehrter Interprozessor-Kommunikationen und einer stark erhöhten Komplexität, bei dem die CM zur lokalen Sperrvergabe autorisiert werden (Zuweisung von Sole-Interest/Leseautorisierungen bzw. Haltesperren).

Aus den vorgenannten Überlegungen heraus erscheinen vor allem vier Einsatzformen des GS bei DB-Sharing von Interesse zu sein. Dabei wird jeweils eine NOFORCE-Ausschreibstrategie unterstellt sowie die Nutzung des GS zur Erstellung einer globalen Log-Datei. Die vier Alternativen unterscheiden sich vor allem hinsichtlich der Synchronisation und der integrierten Behandlung von Pufferinvalidierungen sowie der Verwendung eines globalen Systempuffers.

1. Zur Synchronisation wird das oben erwähnte Primary-Copy-Sperrverfahren (mit Leseoptimierung) verwendet, bei dem der Nachrichtenaustausch über den GS beschleunigt abgewickelt wird. Ferner wird der GS-Seitenbereich im Rahmen einer partitionierten Verwaltung als Erweiterung der lokalen Systempuffer sowie zum optimierten Austausch von Seiten zwischen den CM genutzt. Zur Verkürzung der REDO-Recovery vermerkt jeder CM in einer von ihm verwaltenden GS-Datenstruktur, welche geänderten Seiten der ihm zugeordneten Partition in seinem lokalen Systempuffer residieren (Kap. 6).
   Der Vorteil des Verfahrens ist, daß es primär den GS-Seitenbereich nutzt und die GS-Verwaltung weitgehend durch lokale Datenstrukturen realisiert wird. Mit dem Primary-Copy-Sperr-verfahren wird zudem noch am ehesten eine Synchronisation auf Eintragsebene möglich [Ra88a].

2. Zur Synchronisation wird der in 3.2.1 diskutierte Vorschlag mit expliziter Objektidentifizierung (Hash-Tabelle) und ohne lokale Sperrgewährung durch die CM verwendet; Pufferinvalidierungen werden durch Invalidierungsvektoren in der globalen Sperrtabelle entdeckt. Der Austausch geänderter Seiten erfolgt über ein Propagate-on-Demand-Schema über den GS, es wird aber kein globaler Systempuffer verwendet.
   Der Vorteil liegt hier v.a. in den geringeren Abhängigkeiten zum Lastprofil bzw. Lastverteilung im Vergleich zum Primary-Copy-Ansatz. Außerdem handelt es sich um das einfachste Verfahren mit einer globalen Sperrtabelle im GS.

3. Wie 2) nur mit globalem Systempuffer, der über zentrale GS-Datenstrukturen verwaltet wird. Pufferinvalidierungen im GSP werden zugelassen, und der Zugriff auf sie wird durch Erweiterung der Invalidierungsvektoren in der globalen Sperrtabelle verhindert (5.2.2).
   Damit kann gegenüber 2) ein besseres E/A-Verhalten erreicht werden, allerdings auf Kosten einer erhöhten GS-Belastung und gesteigerter Komplexität.

4. Nutzung einer globalen Sperrtabelle im GS sowie lokaler Sperrvergabe durch die CM; Pufferinvalidierungen in den lokalen Systempuffern werden durch Haltesperren vermieden. Wie in 3) soll ein globaler Systempuffer (Verwaltung über GS-Datenstrukturen) zur Einsparung von Plattenzugriffen genutzt werden.
   Bei diesem Ansatz handelt es sich um das komplexeste Verfahren.

Ein großer Nachteil beim Einsatz des GS stellt die Beschränkung auf nur vier CM dar (begrenzte Erweiterbarkeit). Auch das Ausschreiben von Seiten aus dem GS auf Platte (über die Haupt-speicher der CM) ist sehr umständlich und führt einen unerwünschten Zusatzaufwand ein, der sich v.a. beim Logging niederschlägt. Weiterhin können durch eine ausschließliche Verwendung von 8 B-Einträgen globale Datenstrukturen im GS nur bedingt und umständlich realisiert werden, vor allem wenn Kontrollangaben variabler Länge (Listen) zu verwalten sind. Eine Erleichterung wäre hier bereits durch größere Einträge als 8 B möglich, wenn so die Angaben zu einer Seite (inklusive der Seitennummer) in einem Eintrag abgelegt werden können.

Wünschenswert wäre ferner die Möglichkeit, in einem DB-Sharing-Komplex mehrere unabhängige GS einsetzen zu können. In diesem Fall ließe sich nämlich durch Aufteilung der globalen Sperr-tabelle sowie des globalen Systempuffers (über eine Partitionierung des Namensraumes) die Auslastung der GS reduzieren, so daß z.B. bei der Synchronisation mit einfacheren Verfahren gearbeitet werden kann. Weitere Unklarheiten betreffen die Realisierung der Interprozessor-Kommunikation über den GS (z.B. Geschwindigkeit und Overhead der Benachrichtigung, daß im GS eine Nachricht hinterlegt ist) sowie sinnvolle Auslastungsgrenzen für den GS. Daneben ist noch ungewiß, ob die angenommenen Zugriffszeiten auf den GS überhaupt eingehalten werden können; bei signifikant schlechteren Werten käme natürlich den Ansätzen mit relativ geringer Zugriffshäufigkeit zum GS (nachrichtenbasierte Synchronisation, lokale Sperrvergabe durch die CM) noch stärkere Bedeutung zu.

Literatur

[DIRY87] D.M. Dias, B.R. Iyer, J.T. Robinson, P.S. Yu: *Design and Analysis of Integrated Concurrency-Coherency Controls*. Proc. 13th VLDB, 1987, 463-471

[DYB87] D.M. Dias, P.S. Yu, B.T. Bennett: *On Centralized versus Geographically Distributed Database Systems.* Proc. 7th Int. Conf. on Distributed Computing Systems, 1987, 64-71

[EH84] W. Effelsberg, T. Härder: *Principles of Database Buffer Management*. ACM TODS 9 (4), 1984, 560-595

[GS86] *Global Storage GS*. Draft, Siemens AG, Okt. 1986

[HR83] T. Härder, A. Reuter: *Principles of Transaction-Oriented Database Recovery.* ACM Computing Surveys 15 (4), 1983, 287-317

[HR85] T. Härder, E. Rahm: *Quantitative Analyse eines Synchronisationsalgorithmus für DB-Sharing*. Proc. 3. GI/NTG-Fachtagung über Messung, Modellierung und Bewertung von Rechensystemen, Springer-Verlag, Informatik-Fachberichte 110, 1985, 186-201

[HR86] T. Härder, E. Rahm: Mehrrechner-Datenbanksysteme für Transaktionssysteme hoher Leistungsfähigkeit. Informationstechnik 28 (4), 1986, 214 - 225

[HR87] T. Härder, E. Rahm: *Hochleistungsdatenbanksysteme - Vergleich und Bewertung aktueller Architekturen und ihrer Implementierung*, Informationstechnik 29 (3), 1987, 127 - 140

[Hs88] Y.-P. Hsu: *Performance Evaluation of Data Sharing Transaction Processing Systems*. Master's Thesis, Dept. of Electrical and Computer Engineering, Univ. of Massachussetts at Amherst, 1988

[IYD85] B.R. Iyer, P.S. Yu, L. Donatiello: *Analysis of Fault-Tolerant Multiprocessor Architectures for Lock Engine Design.* IBM Research Report RC 11314, Yorktown Heights, 1985

[Kö88a] M. Köstler: *Nutzung des Global Storage in einer DB-Sharing-Konfiguration*. Fallstudie, Siemens AG, D ST DB1, April 1988

[Kö88b] M. Köstler: *Recovery in einer DB-Sharing-Konfiguration*. Internes Papier, Siemens AG, D ST DB1, April 1988

[Pe86] P. Peinl: *Synchronisation in zentralisierten Datenbanksystemen - Algorithmen, Realisierungsmöglichkeiten und quantitative Bewertung* -. Dissertation, Univ. Kaiserslautern, FB Informatik, 1986

[Ra86] E. Rahm: *Nah gekoppelte Rechnerarchitekturen für ein DB-Sharing-System.* Proc. 9. NTG/GI-Fachtagung über Architektur und Betrieb von Rechensystemen, VDE-Verlag, NTG-Fachberichte 92, 1986, 166-180

[Ra88a] E. Rahm: *Synchronisation in Mehrrechner-Datenbanksystemen - Konzepte, Realisierungsformen und quantitative Bewertung* -. Dissertation, Univ. Kaiserslautern, FB Informatik, 1988

[Ra88b] E. Rahm: *Design and Evaluation of Concurrency and Coherency Control Techniques for Database Sharing Systems*, Interner Bericht 182/88, Fachbereich Informatik, Univ. Kaiserslautern, 1988

[Re81] A. Reuter: *Fehlerbehandlung in Datenbanksystemen*. Carl Hanser 1981

[Re83] A. Reuter: *Schnelle Recovery-Algorithmen für Datenbanksysteme*. Interner Bericht 69/83, Fachbereich Informatik, Univ. Kaiserslautern, 1983

[Sch88] P. Scheug: *Entwicklung und simulative Bewertung von Synchronisationsverfahren für DB-Sharing unter zentraler Kontrolle*. Diplomarbeit, FB Informatik, Univ. Kaiserslautern, 1988

[Sh85] K. Shoens et al.: *The AMOEBA Project.* Proc. IEEE Spring CompCon, 1985, 102-105

[Tr83] I. Traiger: *Trends in Systems Aspects of Database Management.* Proc. 2nd Int. Conf. on Databases (ICOD-2), 1983, 1-20

Diese Arbeit wurde von der Siemens AG finanziell unterstützt.