In: Proc. 11. ITG/GI-Fachtagung "Architektur von Rechensystemen", München, 1990, VDE-Verlag, pp. 123-137.

Nutzung neuer Speicherarchitekturen in Hochleistungs-Transaktionssystemen

Theo Härder, Erhard Rahm, Universität Kaiserslautern

ABSTRACT

Wir untersuchen die Verwendung neuer Speicherarchitekturen in lokal gekoppelten Mehrrechner-Datenbanksystemen zur Realisierung von Transaktionssystemen hoher Leistungsfähigkeit. Eine nahe Rechnerkopplung über einen *gemeinsamen erweiterten Halbleiterspeicher (GEH)* erlaubt vielfältige Einsatzformen und verspricht enorme Leistungsgewinne gegenüber lose gekoppelten Systemen mit konventioneller E/A-Architektur. Wesentliche Merkmale des GEH sind Nicht-Flüchtigkeit, sehr schneller Zugriff im unteren Mikrosekundenbereich sowie zwei verschiedene Zugriffsgranulate (Seite und Eintrag). Alternative Verwendungsmöglichkeiten des GEH zur systemweiten Synchronisation werden im Detail beschrieben und diskutiert.

1. EINFÜHRUNG

An DB-gestützte Transaktionssysteme werden zunehmend steigende Anforderungen bezüglich Leistung (Durchsatz), Verfügbarkeit und Wachstumsfähigkeit gestellt, die den Einsatz von Mehrrechner-DBS verlangen [5]. 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 [12] (einfachere Administration, 'gewachsene' Strukturen, langsame Übertragung nur für Ein-/Ausgabenachrichten und nicht für DB-Operationen innerhalb einer Transaktion [3]). Lokal gekoppelte Mehrrechner-DBS, welche aus einer homogenen Menge räumlich benachbarter Rechnerknoten bestehen, stellen daher eine geeignete Grundlage zur Realisierung von Hochleistungs-Transaktionssystemen dar. Die lokale Rechneranordnung erlaubt dabei eine effiziente Kopplung der einzelnen Verarbeitungsrechner (VAR), welche entweder nachrichtenbasiert ('lose Kopplung') oder unter Verwendung gemeinsamer Speicherbereiche ('enge Kopplung') erfolgen kann. Von einer sogenannten 'nahen Rechnerkopplung' (closely coupled systems) über gemeinsame (Halbleiter-) Speicherbereiche verspricht man sich eine wesentlich effizientere Kommunikation [5, 7] als bei loser Rechnerkopplung, wo aufwendige Protokolle und Prozeßwechsel typischerweise einen hohen Kommunikations-Overhead und möglicherweise signifikante Antwortzeitverschlechterungen verursachen; andererseits gewährleistet die nahe Kopplung ein höheres Maß an Fehlerisolation als die enge Kopplung.

DB-Sharing ('shared disk') bezeichnet eine Klasse lokal gekoppelter Mehrrechner-DBS, bei der jeder VAR direkten Zugriff auf die gesamte materialisierte Datenbank auf den Externspeichern (Platten) besitzt [5, 9]. 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 Rechner auftreten (häufig nur bis zu acht Knoten). Desweiteren müssen für eine Reihe von DB/DC-Komponenten neue und aufeinander abgestimmte Lösungen entwickelt werden (Synchronisation und Behandlung des Pufferinvalidierungsproblems, 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 [8] 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 spezielle Halbleiterspeicherbereiche. Allgemeinere Überlegungen bezüglich einer nahen Speicherkopplung bei DB-Sharing wurden bereits in [7] vorgestellt. Im Amoeba-Projekt [10] soll ein solcher (nicht-flüchtiger) Speicherbereich zum beschleunigten Ausschreiben geänderter Seiten sowie deren Austausch zwischen den einzelnen Rechnerknoten genutzt werden; ähnliche Verwendungsformen werden in [2] vorgeschlagen.

In Kap. 2 zeigen wir die mangelnde Eignung herkömmlicher Speicherarchitekturen und motivieren die Schlüsseleigenschaften eines neuen Speichertyps für Hochleistungs-Transaktionssysteme. Kap. 3 gibt dann einen Überblick über seine Verwendungsmöglichkeiten bei DB-Sharing, bevor in Kap. 4 seine Nutzung zur Synchronisation vertieft wird.

2. E/A- UND SPEICHER-ARCHITEKTUREN

Die Speicherkonzepte eines Systems zur Verwaltung großer Datenmengen und seine E/A-Architektur üben einen starken Einfluß auf einige kritische Systemfunktionen wie Pufferverwaltung, Logging und Recovery, Synchronisation sowie COMMIT-Verarbeitung aus; andererseits bestimmen sie indirekt einen beträchtlichen Anteil des Systemoverheads zur Laufzeit (Paging, Prozeßwechsel, Kommunikation). Wir möchten zunächst aufzeigen, wie durch eine zugeschnittene E/A- und Speicherarchitektur die kritischen Probleme bei zentralisierten DBS entschärft werden, und danach ihre spezielle Nutzung für Mehrrechner-DBS vorschlagen.

Das Transaktionskonzept [4] verlangt Serialisierbarkeit aller Transaktionen (TA), und das bedeutet in allen praktischen Systemimplementierungen, daß alle Sperren für gelesene und geänderte Daten bis zum Ende der TA (COMMIT) gehalten werden müssen. Es ist offensichtlich, daß die Synchronisation mit zunehmenden Durchsatzforderungen zum Schlüsselproblem in DBS wird; von heutigen Hochleistungssystemen wird eine Durchsatzleistung von > 10^3 TPS (kurze Transaktionen pro Sekunde vom Typ Kontenbuchung) verlangt. Sperren zwingen TA mit unverträglichen Zugriffsmodi auf die entsprechenden Daten zum Warten; die von wartenden TA gehaltenen Sperren blockieren wiederum andere TA usw. Die wirksamste Maßnahme ist demnach die Reduktion des kritischen Pfades, in dem eine TA Sperren hält. Hier sollen nicht spezielle Synchronisationsalgorithmen diskutiert werden; es sollen vielmehr Häufigkeit und Dauer von E/A-Vorgängen und ihr Einfluß auf die Synchronisation grob abgeschätzt werden.

Als Musterbeispiel einer kurzen Transaktion ziehen wir die "Kontenbuchung" (DEBIT_CREDIT) heran; sie wird bereits weltweit als "Einheitsmaß" für den Durchsatz von Transaktionssystemen genutzt. Im vorgeschlagenen Benchmark für eine Bankanwendung besitzt die DB folgende Kennwerte für die verwendeten Satztypen [1]:

- KONTO 10^7 Sätze, direkter Zugriff (Hash)
- ZWEIGSTELLE 10^3 Sätze, direkter Zugriff
- SCHALTER 10^4 Sätze, direkter Zugriff
- BUCHUNG 20×10^7 Sätze/Monat, sequentieller Zugriff

Eine TA führt

- jeweils 3 Lese- und 3 Aktualisierungsvorgänge (KONTO, ZWEIGSTELLE, SCHALTER)
- 1 Einfügevorgang (BUCHUNG)
- mind. 2 Modifikationsoperationen auf Systemtabellen (Adressierung, Freispeicherverwaltung)
- Logging

durch. Als typischen Bedarf für die TA-Verarbeitung unterstellen wir hier 200.000 Instruktionen, was einem CPU-Zeitanteil von 20 ms bei einem Rechner von 10 MIPS Leistung entspricht. Den Einfluß der E/A-Operationen auf die TA-Verarbeitung diskutieren wir anhand verschiedener E/A-Architekturen nach Bild 1. Ihre für unsere Diskussion wichtigen Merkmale sind in den einzelnen Schemadarstellungen veranschaulicht.

Eine konventionelle E/A-Architektur ist in Bild 1a skizziert. Ohne ins Detail zu gehen, wollen wir mit den angegebenen Kennwerten den E/A-Bedarf der TA abschätzen. Als "worst case" wird zunächst an-genommen, daß der DB-Puffer so klein bemessen ist, daß keine Lokalität auf den Daten ausgenutzt werden kann. Weiterhin wird das für viele Systeme typische Seitenlogging unterstellt, wobei das WAL-Prinzip eingehalten werden muß [4]. Es ist also ggf. UNDO und REDO-Information (BEFORE- und AFTER-IMAGE) auf die Log-Datei zu schreiben. Unterstellen wir für eine sequentielle Log-Ausgabe 20 ms und für die restlichen E/As 50 ms, so ergeben

- 6 Lese- und 6 Schreiboperationen auf Daten- und Systemseiten 600 ms
- 12 Schreiboperationen für Log-Information 240 ms.

Damit stehen unserem CPU-Bedarf von 20 ms ein E/A-Bedarf von 840 ms gegenüber.

Zunächst soll untersucht werden, wie gut sich ein *Platten-Cache* [11] zur E/A-Beschleunigung für Magnetplatten (Bild 1b) bei der TA-Verarbeitung eignet. Heutige Platten-Caches sind i.d.R. flüchtig und führen deshalb zu einem "unsicheren Schreiben", was zumindest für die Log-Ausgabe nicht akzeptabel ist. Als zusätzliche Maßnahme ist deshalb die Möglichkeit des Durchschreibens auf Platte zu fordern. Ein hinreichend großer Cache unterstützt die Lokalität der Datenzugriffe; mit großer Wahrscheinlichkeit werden deshalb alle benötigten Sätze bis auf KONTO im Cache aufgefunden. Damit ergibt sich folgende Bilanz:

- 1 Leseoperation (KONTO) von Platte 50 ms
- 5 Lese- und 6 Schreiboperationen von/in Cache 22 ms
- 12 Schreiboperationen für Log-Information 240 ms.

Als charakteristischen Wert halten wir hier 312 ms fest.

Die durch Cache-Nutzung erzielte Lokalität läßt sich jedoch viel effektiver und ohne Qualitätsverlust durch große DB-Puffer im (flüchtigen) Hauptspeicher erreichen (NOFORCE-Strategie bei COMMIT [4]). Durch die hohe Referenzlokalität bleiben alle Datenseiten bis auf die vom Typ KONTO (z.B. bei LRU) im DB-Puffer. Die Schreiboperation des modifizierten Kontosatzes kann asynchron nach COMMIT erfolgen; lediglich für den aktuellen KONTO-Satz fällt ein synchroner Lesevorgang von Platte an. Bei dieser Betriebsweise sind außerdem nur REDO-Informationen zu schreiben:

- 1 Leseoperation (KONTO) von Platte 50 ms
- 6 Schreiboperationen (REDO) auf Log 120 ms

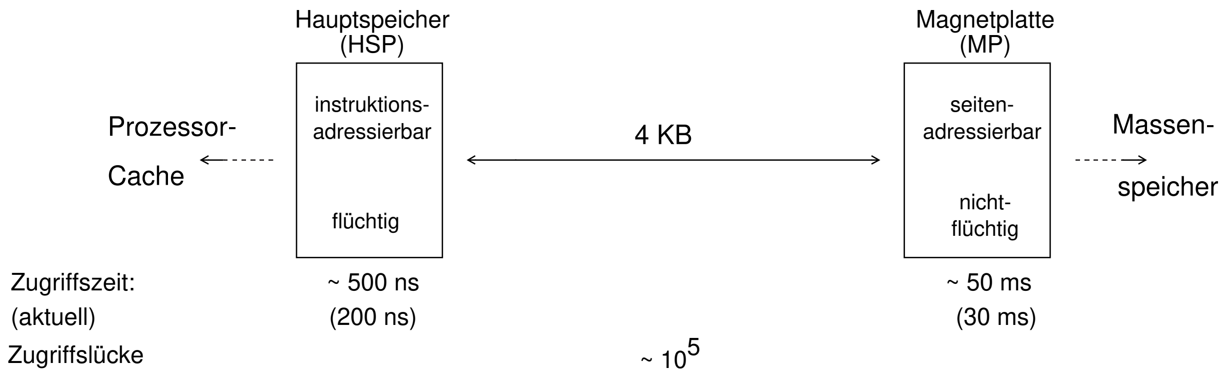
Der für die konventionelle E/A-Architektur mit großem DB-Puffer (Bild 1a) und NOFORCE-Strategie ermittelte Kennwert beträgt also 170 ms.

Zwei Beobachtungen sollen hier explizit festgehalten werden:

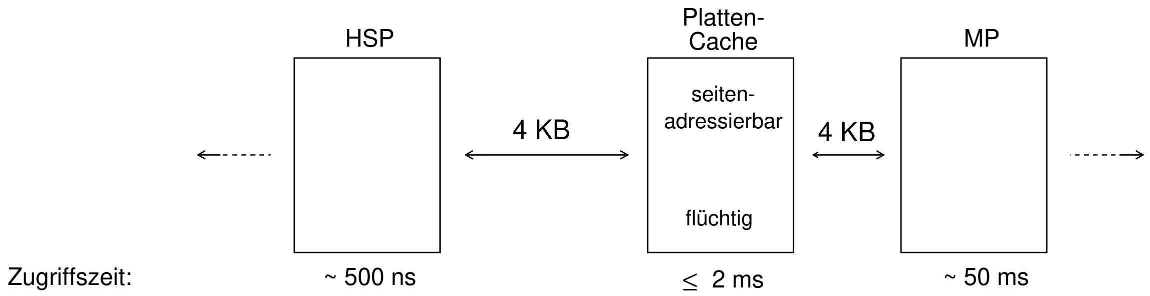
- Lokalität läßt sich im Hauptspeicher wesentlich besser nutzen als im Platten-Cache. Der gemeinsame Einsatz beider Maßnahmen bringt für die TA-Verarbeitung kaum Vorteile; es besteht vielmehr die Gefahr, daß die Ersetzungsstrategien im DB-Puffer und im Platten-Cache "gegeneinander" arbeiten.
- Durch die Verringerung der Daten-E/A wurde die Log-E/A zur dominierenden Größe. Durch Konzepte wie Chained-I/O, Eintrags-Logging und Group-Commit läßt sich der Log-Aufwand drastisch reduzieren.

Zur Beseitigung der Zugriffslücke wurden neuere E/A-Architekturen (Bild 1c und 1d) vorgeschlagen. Der *erweiterte Hauptspeicher (EH)* ist flüchtig und somit nur als Ergänzung der konventionellen Hauptspeicher-Architektur zu betrachten (Derzeit werden EH hauptsächlich als Paging-Speicher verwendet). Bei erweitertem DB-Puffer im EH zu im Vergleich zum Hauptspeicher geringeren Kosten ist eine höhere Trefferrate und damit eine im Mittel geringere Daten-E/A zu erwarten. *"Solid State Disks" (SSD)* bieten Nicht-Flüchtigkeit als wesentliches Merkmal und direkten Dateizugriff mit der Geschwindigkeit des verfügbaren Kanals (3 - 4.5 MBytes/sec). Wegen der hohen Kosten lassen sich SSDs nur sehr selektiv einsetzen. Sie lösen jedoch die Log-Problematik, da sich 6 Log-E/A in $\leq$ 12 ms bewältigen lassen.

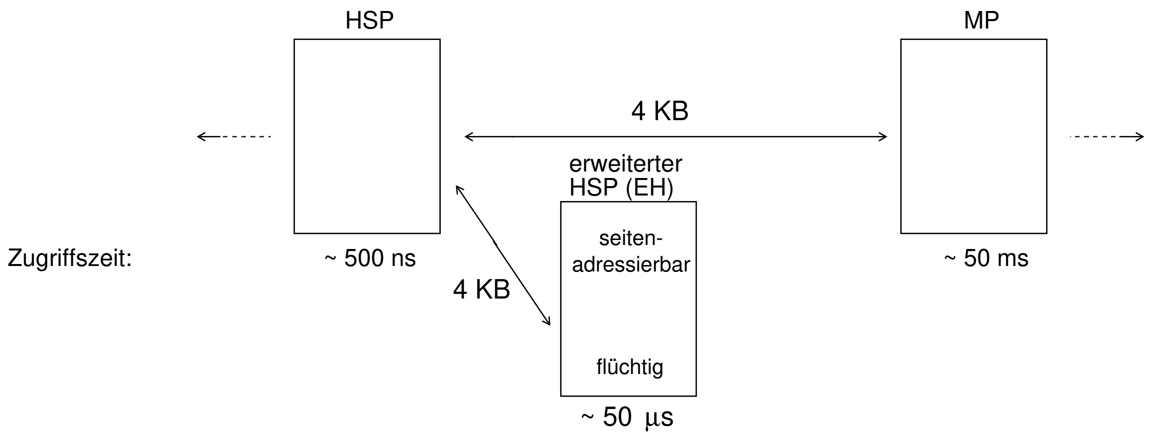
Durch die bisher betrachteten Vorschläge zur E/A-Architektur konnte der Löwenanteil des E/A-Bedarfs unserer Muster-Transaktion eliminiert werden, was direkt die "Behinderungsdauer" einer TA reduziert und unmittelbar ihrer Antwortzeit zugute kommt. Eine weitere Verringerung der E/A-Zeit läßt sich durch einen Halbleiterspeicher erzielen, der die Geschwindigkeit des EH mit der Nicht-Flüchtigkeit der SSD vereinigt. Bei einer seitenorientierten Schreibzeit in der Größenordnung von 50 μs fällt dann die Log-Ausgabe nicht mehr ins Gewicht. Werden bei entsprechender Größe des EH auch gesuchte Sätze vom Typ KONTO lokalisiert (im Mittel 0.5 pro TA), so reduziert sich der Aufwand zur Bearbeitung einer Kontenbuchungs-TA auf 20 ms CPU-Zeit und 25 ms E/A-Zeit.



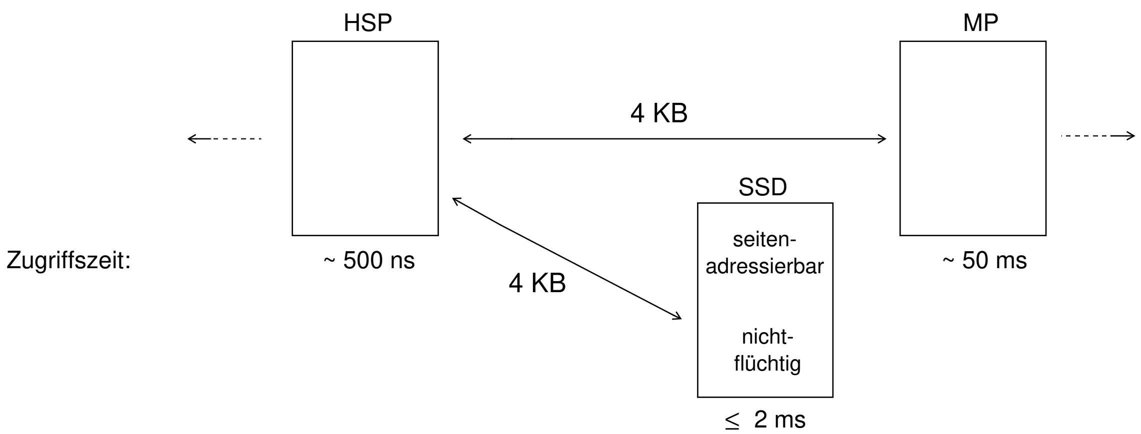
a) konventionelle E/A-Architektur



b) E/A-Beschleunigung durch Platten-Cache



c) Einsatz eines erweiterten Hauptspeichers (flüchtig)



d) Nutzung von nicht-flüchtigen Solid State Disks (SSD)

Bild 1: E/A-Architekturen für Transaktionssysteme

Bei Mehrrechner-DBS sind jedoch noch andere Gründe für den Einsatz nicht-flüchtiger EH maßgebend. Ein solcher nicht-flüchtiger EH ist ein idealer Speichertyp, um die Kooperation in einem Mehrrechner-DBS bei gleichzeitiger Isolation der Einzelsysteme (Fehlertoleranz) zu fördern. Wir nennen einen nicht-flüchtigen EH, der seitenadressierbar ist und von allen Rechnern des Verbundes erreicht werden kann, einen *GEH (gemeinsamer erweiterter Halbleiterspeicher)*. Wichtige Aufgaben in einem solchen Verbund liegen beispielsweise im schnellen Austausch von Datenseiten, in der Kommunikation, in der Speicherung gemeinsamer Datenstrukturen, wobei die Nicht-Flüchtigkeit des Speichers die Verfügbarkeit des Gesamtsystems erheblich steigert (Bild 2). Im Unterschied zu Speichertypen, die von einem Kontroller asynchron verwaltet werden, fordern wir für unsere TA-Anwendung die unmittelbare Verwaltung des GEH durch die VAR über eine synchrone Schnittstelle. Dadurch ist seitens eines VAR-Prozesses ein direkter seitenorientierter Zugriff auf den GEH möglich, ohne eine Prozeß-Deaktivierung durchführen zu müssen, was durch die Häufigkeit solcher Ereignisse enorme Einsparungen verspricht. Eine weitere Spezialisierung dieser Schnittstelle könnte wie folgt aussehen: Neben dem großen Seitengranulat (von 4 KB) wäre eine kleine Zugriffseinheit (z.B. 16 oder 32 B) in disjunkten Partitionen sehr hilfreich, da in Mehrrechner-DBS umfangreiche gemeinsame Datenstrukturen (Pufferverwaltung, Sperrtabellen, Warteschlangen usw.) realisiert werden müssen.

Zusammenfassend sei noch einmal festgestellt: Die drastische Reduktion der E/A-Zeit, die durch die Kapazität und Zugriffsgeschwindigkeit von GEH erzielt wird, ist eine Voraussetzung zum Erreichen einer hohen TA-Leistung (vor allem durch Eliminierung langer Blockierungszeiten). Daneben sind für unseren Ansatz folgende Schlüsseleigenschaften des GEH maßgebend:

- Nicht-Flüchtigkeit
- gemeinsamer Zugriff über eine synchrone Schnittstelle
- verschiedene Zugriffsgranulate für Datenseiten und Verwaltungsdaten.

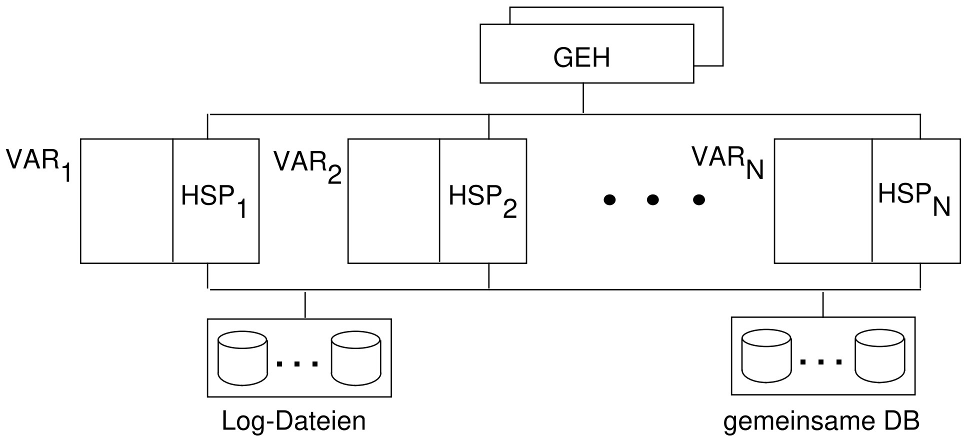


Bild 2: Stellung des GEH in einem DB-Sharing-System

3. GENERELLE VERWENDUNGSMÖGLICHKEICHTEN EINES GEH BEI DB-SHARING

Die beiden Zugriffsgranulate 'Eintrag' und 'Seite' bestimmen im wesentlichen mögliche Verwendungsformen des GEH:

- *GEH-Einträge* bieten sich zur Speicherung globaler Datenstrukturen (Tabellen) an, z.B. zur systemweiten Synchronisation, zur Behandlung des bei DB-Sharing zu lösenden Puferrinvalidierungsproblems oder aber zur Verwaltung des GEH selbst.
- Der *GEH-Seitenbereich* eignet sich vor allem zur Aufnahme von DB-Seiten - entweder zum Austausch von geänderten Seiten zwischen den Rechnern oder infolge einer Verdrängung aus einem lokalen Systempuffer - und Logging-Daten.
- Daneben kann der GEH zur beschleunigten *Interprozessor-Kommunikation* verwendet werden. Dies kann entweder mit GEH-Einträgen oder GEH-Seiten realisiert werden, abhängig von der Größe der

Nachricht. So kann z.B. der Austausch einer geänderten DB-Seite über den GEH als beschleunigtes Senden einer 'großen Nachricht' aufgefaßt werden.

Bei DB-Sharing kommt der GEH 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 [8, 9], einzusetzen und den GEH nur zur beschleunigten Interprozessor-Kommunikation zu verwenden. Von der Vielzahl der denkbaren Protokolle kommt dabei v.a. das *Primary-Copy-Sperrverfahren* in Frage, daß sich bei Trace-getriebenen Simulationen als am leistungsfähigsten erwiesen hat [8].

Eine weitergehende Verwendung des GEH ergibt sich, wenn *im GEH globale Datenstrukturen zur Synchronisation* abgelegt werden, welche von allen Knoten benutzt werden. Eine derartige Nutzung des GEH zur Synchronisation läuft - im Gegensatz zum Primary-Copy-Verfahren - auf ein zentrales Sperrverfahren (*) hinaus, mit dem es aufgrund der hohen GEH-Zugriffs- geschwindigkeit möglich sein soll, Sperren wesentlich schneller als über ein nachrichtenbasiertes Protokoll zu erwerben und freizugegeben.

BEHANDLUNG VON PUFFERINVALIDIERUNG

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) [8, 9]. 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) vorzusehen. Wird die globale Sperrtabelle im GEH abgelegt, so können nun natürlich auch für geänderte DB-Seiten die *Angaben zur Behandlung des Veralterungsproblems in Tabellen des GEH* abgelegt werden. Desweiteren kann der GEH 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 GEH.

GLOBALER SYSTEMPUFFER

Eine erweiterte Nutzung des GEH-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* [4] bezüglich des GEH verfolgt werden soll. Dabei werden bei FORCE am Ende einer Update-TA sämtliche von ihr geänderte Seiten vom LSP in den GEH geschrieben, während bei NOFORCE ein solches 'Hinauszwingen' der Änderungen aus den lokalen Puffern unterbleibt. NOFORCE erlaubt damit eine geringere Zugriffshäufigkeit auf den GEH und vermeidet die mit den Schreibvorgängen einhergehende Antwortzeitverlängerung (erhöhte Sperrdauer) für Änderungs-TA; (diese Verzögerungen sind mit dem nicht-flüchtigen GEH allerdings erheblich kürzer als bei einem Durchschreiben der Änderungen auf Platte). Andererseits erschwert NOFORCE die Recovery nach einem Rechnerausfall (REDO verlorengegangener Änderungen) sowie die Behandlung von Pufferinvalidierungen, die bei NOFORCE auch im globalen Systempuffer möglich sind.

---

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

LOGGING

Aufgrund der Nicht-Flüchtigkeit des GEH sowie der im Vergleich zu Platten wesentlich schnelleren Zugriffszeiten bietet es sich natürlich zur Reduzierung der Antwortzeiten an, Log-Daten in den GEH auszuschreiben. Dazu kann der GEH sowohl zur (temporären) Aufnahme lokaler Log-Daten (Terminalnachrichten, UNDO- bzw. REDO-Daten) der einzelnen VAR genutzt werden, als auch für globale (REDO-) Log-Daten aller Rechner.

Die Verwendung des GEH für lokales Logging ist sehr einfach möglich, wenn jedem VAR für diesen Zweck eine GEH-Partition fest zugeordnet wird (keine rechnerübergreifende Koordinierung erforderlich). Die Erstellung der globalen Log-Datei verlangt, daß alle Rechner am Ende einer Änderungs-TA die REDO-Informationen an das aktuelle Ende dieser Datei schreiben. Zur notwendigen Koordinierung kann ein spezieller GEH-Eintrag verwendet werden, der auf das aktuelle Ende der globalen Log-Datei zeigt und von jeder Änderungs-TA um die benötigte Anzahl von Seiten inkrementiert wird. Wegen der hohen GEH-Zugriffsgeschwindigkeit wird durch diesen Eintrag kein Engpaß hinsichtlich der erzielbaren Transaktionsraten eingeführt. Allerdings müssen wegen der begrenzten Größe der Log-Dateien die REDO-Informationen asynchron vom GEH auf Hintergrundspeicher ausgelagert werden.

LASTKONTROLLE

Wird die globale Lastkontrolle auf mehreren Front-End-Rechnern realisiert, dann können die einzelnen Prozessoren bei einer nahen Kopplung über einen (eigenen) GEH diesen zur schnelleren Kommunikation untereinander nutzen, zur schnellen Sicherung von Eingabenachrichten oder zur Ablage gemeinsam benutzter Datenstrukturen (z.B. Routing-Tabelle, Angaben zur aktuellen Last- und Verteilsituation, u.ä.). Solche globalen Informationen im GEH können auch von den VAR genutzt werden, sofern diese selbst die Lastverteilung vornehmen (anstatt einer TA-Verteilung durch Front-End-Rechner). Dieser Ansatz wird nun wesentlich attraktiver als bei loser Rechnerkopplung, da Umverteilungen von TA-Aufträgen über den GEH effizient möglich sind.

Weitere Alternativen beim Einsatz eines GEH, die bisher noch nicht angeprochen wurden, ergeben sich bezüglich der *Verwaltung des GEH*, insbesondere bezüglich der Verwaltung des GEH-Seitenbereiches (Austausch geänderter Seiten, globaler Systempuffer, Logging). Neben einer Verwaltung über zentrale Datenstrukturen im GEH selbst kommt dabei auch ein sogenannter *partitionierter Ansatz* in Frage, bei dem jeder VAR eine Partition des GEH kontrolliert und die dazu erforderlichen Datenstrukturen lokal verwaltet (z.B. zur Realisierung lokaler Log-Dateien im GEH). In [2] wird ein solcher Ansatz vorgestellt, wobei ein VAR zwar Lesezugriffe bezüglich des gesamten gemeinsamen Speicherbereiches (hier: GEH) 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 GEH denkbar, wobei dies im einzelnen natürlich stark von der vorgesehenen Verwendung der zu verwaltenden GEH-Daten abhängt.

Im folgenden Kapitel wird näher ausgeführt, wie der GEH zur Synchronisation in DB-Sharing-Systemen genutzt werden kann. Detailliertere Lösungsstrategien für die anderen Anwendungs- möglichkeiten können in diesem Aufsatz aus Platzgründen nicht diskutiert werden.

4. SYNCHRONISATION BEI DB-SHARING UNTER NUTZUNG EINES GEH

Die systemweite Zugriffssynchronisation auf die gemeinsamen Datenbestände ist in DB-Sharing-Systemen eine wesentliche und leistungsbestimmende Funktion. Bei loser Kopplung ist die Synchronisation nur über Nachrichtenaustausch zu lösen, wobei zur Erzielung hoher TA-Raten und kurzer Antwortzeiten die Anzahl der Kommunikationsvorgänge möglichst gering zu halten ist [9]. In diesem Kapitel werden drei Einsatzformen eines GEH zur Synchronisation untersucht, mit dem die Leistungsfähigkeit gegenüber lose gekoppelten DB-Sharing-Systemen verbessert werden soll. Zunächst wird in 4.1 ein nachrichtenbasiertes Synchronisationsverfahren unterstellt, wobei der Nachrichtenaustausch über den GEH vorgenommen wird. Danach werden zwei Verfahren untersucht, bei denen eine globale Sperrtabelle im GEH geführt wird. In 4.2 diskutieren wir dazu den Fall, bei dem die gesamten globalen Sperrangaben im GEH geführt werden, während in 4.3 dort nur eine reduzierte Sperrinformation gehalten wird. In beiden

Fällen müssen die Sperrangaben auf GEH-Einträge abgebildet werden, die nicht direkt, sondern nur im Hauptspeicher der zugreifenden VAR gelesen und modifiziert werden können. Im Hauptspeicher geänderte Einträge müssen wieder in den GEH zurückgeschrieben werden, um für alle VAR sichtbar zu sein.

4.1 NACHRICHTENAUSTAUSCH ÜBER DEN GEH

Werden Nachrichten zwischen VAR über den GEH ausgetauscht, so können sich auch bei Verwendung eines nachrichtenbasierten Synchronisationsverfahrens, z.B. dem Primary-Copy-Sperrprotokoll [8], verbesserte Leistungsmerkmale im Vergleich zu loser Rechnerkopplung ergeben. Soll eine Nachricht X von VAR1 nach VAR2 gebracht werden, so sind im wesentlichen drei Schritte vorzunehmen:

1.) VAR1 schreibt X in den GEH.
2.) VAR1 schickt eine Benachrichtigung (z.B. Interrupt) an VAR2, daß X vom GEH eingelesen werden kann. Dabei wird auch die GEH-Adresse mitgeteilt, an der X vorliegt.
3.) VAR2 liest X vom GEH in den Hauptspeicher.

Diese Vorgehensweise kann offenbar nur dann einen signifikanten Vorteil gegenüber einem direkten Nachrichtenaustausch bringen, wenn die Benachrichtigung in Schritt 2 wesentlich effizienter als eine 'allgemeine' Nachrichtenübertragung realisiert werden kann. Dies sollte i.a. möglich sein, da es sich bei solchen Benachrichtigungen um sehr kurze Nachrichten fester Länge handelt.
Der Austausch einer DB-Seite zwischen zwei VAR kann ebenfalls mit dem obigen Schema abgewickelt werden (Nachricht X = DB-Seite).

Die Realisierung der angegebenen drei Schritte zum Nachrichtenaustausch über den GEH erfordert allerdings noch die Behandlung folgender Verwaltungsaufgaben:

1.) Bevor eine Nachricht (Seite) in den GEH geschrieben werden kann, muß zunächst ein Eintrag (Seite) im GEH lokalisiert werden, der frei ist bzw. überschrieben werden kann.
2.) Die in den GEH geschriebene Nachricht muß vor einem Überschreiben durch andere VAR geschützt werden (zumindest bis sie vom Adressaten eingelesen wurde).
3.) Nach Einlesen der Nachricht ist der betreffende Eintrag (Seite) im GEH freizugeben.

Wie schon in Kap. 3 erwähnt, kommen für diese Verwaltungsaufgaben unterschiedliche Strategien in Betracht, z.B. eine partitionierte GEH-Verwaltung oder die Verwendung von zentralen Verwaltungsdaten im GEH. Beim partitionierten Ansatz kann jeder VAR nur auf eine ihm zugeordnete GEH-Partition schreibend zugreifen (lesend dagegen auf den gesamten GEH), so daß die Punkte 1 und 2 mit lokalen Datenstrukturen im Hauptspeicher zu lösen sind. Punkt 3 dagegen erfordert jedoch, daß der Empfang einer Nachricht quittiert wird, damit die betreffende GEH-Information als überschreibbar gekennzeichnet werden kann. Die Verwaltung des Nachrichtenaustausches über GEH-Datenstrukturen kann z.B. über eine Bitliste realisiert werden. Dabei wird pro 'Nachrichtenbehälter' (GEH-Eintrag oder Seite) ein Bit vorgesehen, das anzeigt, ob der zugehörige Behälter frei ist oder nicht. Damit werden pro Nachricht vier weitere GEH-Zugriffe notwendig: zwei zum Lokalisieren und Blockieren eines 'Behälters' (Lesen und Zurückschreiben der Bitliste) und zwei zum Freigeben. Damit ergeben sich sechs GEH-Zugriffe pro Nachricht. Die Gesamtzugriffshäufigkeit zum GEH ist natürlich davon abhängig wieviele Nachrichten das zugrundeliegende Synchronisationsverfahren benötigt.

4.2 GLOBALE SPERRTABELLE VOLLKOMMEN IM GEH

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 [6]. Semaphore (z.B. pro Hash-Klasse) verhindern den gleichzeitigen Zugriff auf die Datenstrukturen durch verschiedene DBVS-

Prozesse eines Rechners. 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 GEH verwaltet werden, wobei im Prinzip lediglich die GEH-Einträge zur Verfügung stehen. Um die Diskussion überschaubar zu halten, konzentrieren wir uns hier auf die Speicherung der *objektspezifischen 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. 4-8 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 [6]. Pro Sperranforderung ist dabei eine TA-Kennung (z.B. aus VAR-ID und relativer TA-Nummer) sowie der Sperrmodus zu führen.

Zur Realisierung der Listenstruktur(en) im GEH ist offensichtlich eine explizite *Verzeigerung von GEH-Einträgen*, welche Sperrinformationen desselben Objektes führen, erforderlich, wobei die Verweise zu den Listen im festen Teil der Sperrangaben abzulegen sind.

Bild 3 zeigt eine mögliche Realisierung der globalen Sperrtabelle. Die Sperrangaben fester Länge sind im Rahmen einer Hash-Tabelle mit Überlaufbereich gespeichert, worin nur für aktuell gesperrte DB-Objekte Einträge verwaltet werden. Wie Bild 3 zeigt, besteht die Hash-Tabelle aus einem Primärbereich mit H1 Hash-Klassen sowie einem Überlaufbereich von H2 Hash-Klassen, wobei pro Hash-Klasse jeweils ein GEH-Eintrag mit Angaben zu höchstens einem Objekt vorgesehen ist. Ein GEH-Eintrag enthält neben der Objekt-ID weitere Sperrangaben fester Länge (höchster gewährter Sperrmodus, Anzahl wartender Sperranforderungen, etc.) sowie zwei Pointer, welche auf andere GEH-Einträge verweisen. Einer der Pointer dient zur Kollisionsbehandlung und verkettet GEH-Einträge zu Objekten, welche (zunächst) auf dieselbe Hash-Klasse abgebildet werden. Der zweite Pointer zeigt in den variablen Teil der globalen Sperrtabelle, und zwar auf das erste Element der gemeinsamen Liste gewährter und wartender Sperranforderungen (*GWS-Liste*). Im Gegensatz zu den *Primäreinträgen* in der Hash-Tabelle bezeichnen wir die Elemente der GWS-Liste als *Sekundäreinträge*. In den Sekundäreinträgen wird neben einem Folgeverweis im wesentlichen noch die Rechner- und TA-Kennung sowie der Zugriffswunsch (Sperrmodus) geführt. Das Beispiel in Bild 3 zeigt, daß für das betrachtete Objekt den TA T1 und T2 eine

*TA-spezifische Angaben wie Menge gewährter Sperren einer TA werden sinnvollerweise in den VAR 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.

Lesesperre (R-Sperre) gewährt wurde, während T3 auf eine X-Sperre wartet.

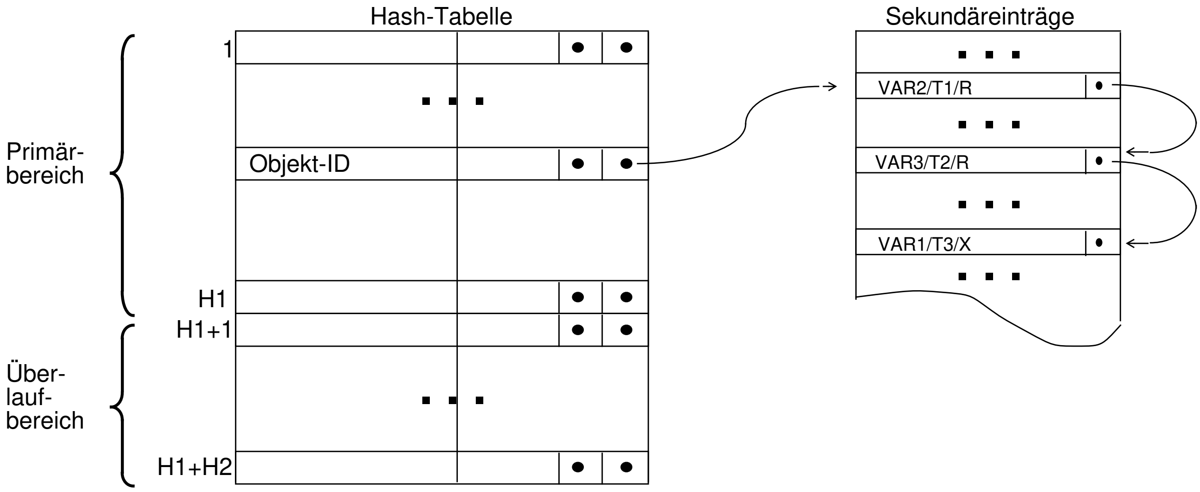


Bild 3: Realisierung einer globalen Sperrtabelle im GEH (Beispiel)

Die Sekundäreinträge können aus einem Pool von GEH-Einträgen allokiert werden, wobei z.B. wieder durch ein Bit pro Eintrag angezeigt werden kann, ob er 'frei' oder 'belegt' ist, oder eine rechnerspezifische Partitionierung des Pools vorgesehen wird. Die Größe des Pools und damit der variable Teil der globalen Sperrtabelle dürfte relativ klein gehalten werden können, da typischerweise kurze TA mit wenigen Sperren dominieren und die Gesamtzahl parallel gestarteter TA auch begrenzt ist.

Im folgenden soll nun kurz diskutiert werden, welche Aktionen bezüglich des GEH 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 in der Hash-Tabelle in den Hauptspeicher des zugreifenden VAR einzulesen. Dieser Eintrag 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 GEH-Zugriffe pro Sperranforderung* an (Lesen und Zurückschreiben des geänderten Primär- sowie des Sekundäreintrages). Noch mehr Zugriffe entstehen bei Namenskollisionen sowie bei abgelehnter Sperranforderung. In letzterem Fall ist ein Sekundäreintrag an das Ende der GWS-Liste anzufügen. Dies erfordert jedoch ein sequentielles Lesen der gesamten Liste, wenn nicht im Primäreintrag das aktuelle Listenende abgelegt wird.

UNLOCK:

Es wird davon ausgegangen, daß für jede TA im HSP des ausführenden VAR sämtliche Sperren (mit GEH-Adresse des zugehörigen Primär- und 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 GEH-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.

Nimmt man statt der mindestens 8 GEH-Zugriffe pro Sperre (LOCK und UNLOCK) 10 an, so ergibt das für eine TA mit 10 Sperren eine Antwortzeiterhöhung von 100 GEH-Zugriffen. Dies sind bei einer GEH-Zugriffseit von etwa 50 µs lediglich 5 ms, bei 10 µs sogar nur 1 ms (50 µs war der für einen Seitenzugriff unterstellte Wert, so daß für die wesentlich kleineren Einträge kürzere Zeiten erreichbar sein sollten). Wenn jeweils nur ein GEH-Zugriff pro Zeitpunkt stattfinden könnte, würde sich jedoch selbst bei 10 µs bereits für 1000 TPS ein überlasteter GEH ergeben. Daher ist dieser Ansatz für Hochleistungs-Transaktionssysteme nur anwendbar, wenn mehrere GEH-Zugriffe auf verschiedene Einträge/Seiten gleichzeitig möglich sind und/oder noch schnellere Zugriffszeiten erreichbar sind.

Problematisch ist 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 GEH-Einträge zu ändern sind, was i.a. nicht atomar (z.B. mit einem einzigen GEH-Befehl) möglich sein dürfte. Die betroffenen GEH-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 GEH-Zugriffe pro Sperre) und dementsprechend eine noch höhere GEH-Auslastung. Zur Durchführung der Crash-Recovery müssen auch alle Sperranforderungen des betroffenen VAR, 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.

4.3 SPERRINFORMATIONEN IM GEH SOWIE IN DEN VAR

Die obige Realisierungsform zur globalen Synchronisation verspricht bereits eine wesentlich effizientere Synchronisation als mit einem nachrichtenbasierten Protokoll und loser Kopplung. Denn bei loser Kopplung sind bei nur einer globalen Sperranforderung pro TA die damit verbundenen Antwortzeitauswirkungen sowie Kommunikationsbelastungen i.a. bereits größer als bei einer solchen Synchronisierung über den GEH. Andererseits wurde deutlich, daß die Verwaltung variabler Datenstrukturen im GEH nur sehr umständlich möglich ist, daß die Behandlung von Rechnerausfällen problematisch sein kann (Verfügbarkeit) sowie daß bei hohen TA-Raten möglicherweise Engpässe im GEH drohen (zumal der GEH 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 VAR objektspezifische Sperrinformationen (innerhalb von lokalen Sperrtabellen) geführt werden. Hierzu untersuchen wir im folgenden vor allem, wie die Speicherung von Listenstrukturen in der globalen Sperrtabelle umgangen und dennoch eine korrekte Synchronisierung über den GEH ermöglicht werden kann (jede Sperranforderung soll weiterhin über den GEH entschieden werden).

Ein erster und einfach zu lösender Schritt ist, im GEH die gewährten Sperren nicht mehr auf TA-Ebene, sondern auf Rechnerebene zu vermerken. Die TA-spezifischen Angaben bezüglich gewährter Sperren sowie wartender Sperranforderungen werden nun in der lokalen Sperrtabelle des VAR geführt, in dem die Sperre angefordert wurde. Da die maximale Rechneranzahl NMAX als begrenzt vorausgesetzt werden kann, genügt es hinsichtlich gewährter Sperren in der globalen Sperrtabelle im GEH 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=8 insgesamt 2 B pro Sperreintrag anfallen.

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 GEH entfällt. Damit werden zum Setzen einer Sperre statt mindestens 4 nur noch 2 GEH-Zugriffe erforderlich. Zum Freigeben einer Sperre können sogar weniger als 2 (statt mindestens 4) GEH-Zugriffe ausreichen: denn das Freigeben einer Sperre erfordert nur dann einen GEH-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 GEH-Zugriffe bereits auf rund die Hälfte reduziert.

Wünschenswert wäre nun natürlich auch, im GEH 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 GEH darzustellen, sieht folgendermaßen aus:

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

Ein Warteeintrag besteht also aus der Identifikation des VAR (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 8 B (4 Bit) vorzusehen, bei drei Sperrmodi (z.B. RAX-Verfahren) 15 B (5 Bit). Da daneben noch die Objekt-ID, GRANTED-MODE sowie ein Pointer zur Verkettung der Hash-Klassen in einem Sperreintrag abzulegen sind, sind bei 3 Sperrmodi GEH-Einträge von z.B. 32 B vorzusehen.

Die Abarbeitung der Wartelisten (d.h. Aktivierung wartender TA) erfolgt durch den Rechner, dessen Sperrfreigabe dazu führt, daß wartende Sperranforderungen gewährt werden können.

Die Verwendung der eingeführten Datenstrukturen verdeutlicht das Beispiel in Bild 4 für drei Rechner (NMAX=4) und das RX-Sperrverfahren. Dabei ist für das betrachtete Objekt O einer TA (T1) in Rechner VAR3 eine X-Sperre gewährt, während TA in allen Rechnern auf die Zuteilung einer Lesesperre warten. Darüber hinaus liegen in Rechner VAR1 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 VAR1 nur zum Teil im GEH 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 GEH-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 GEH weitergereicht werden.

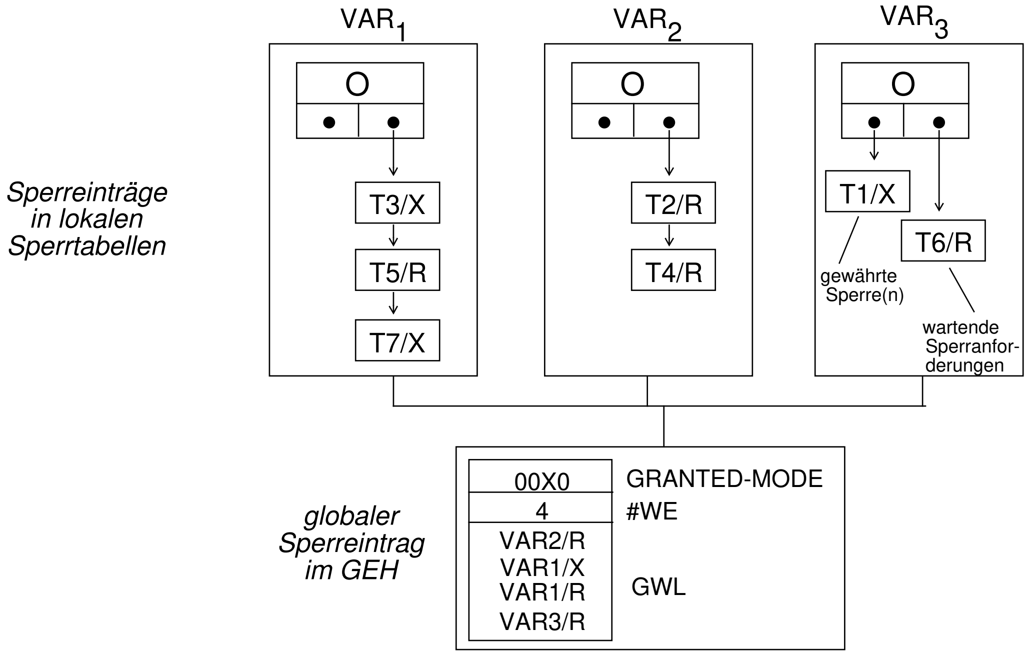


Bild 4: Beispielszenarium zur Verwaltung wartender Sperranforderungen (NMAX=4)

So wird z.B. bei FIFO-Abarbeitung der globalen Warteliste GWL nach Freigabe der gesetzten X-Sperre zunächst eine R-Sperrgewährung von VAR3 nach VAR2 geschickt und der GEH-Eintrag entsprechend angepaßt (GRANTED-MODE := 0R00; #WE := 3; erstes GWL-Element entfernen). In VAR2 werden daraufhin natürlich beide R-Anforderungen befriedigt, obwohl die zweite R-Anforderung nicht mehr im GEH gespeichert wurde. Nach Freigabe der letzten R-Sperre greift VAR2 auf die globale Sperrtabelle zu und veranlaßt die weitere Abarbeitung der globalen Warteliste, wobei dann die X-Anforderung aus VAR1 (T3) befriedigt wird. Erst nach Entfernen des zugehörigen X-Warteeintrages für VAR1 aus der GWL kann nun die bereits länger gestellte X-Anforderung von TA T7 an den GEH weitergereicht werden, wobei ein erneuter X-Warteeintrag für VAR1 an das aktuelle Ende der GWL angefügt wird.

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 GEH eine faire Bedienung wartender Sperranforderungen möglich ist. Damit konnte nun erreicht werden, daß in der globalen Sperrtabelle im GEH für ein Objekt keine variablen Datenstrukturen (Listen) mehr zu verwalten sind, sondern nur noch ein einfaches Feld fester Länge (keine Verkettung).

Die Führung lokaler Sperrtabellen gestattet jedem VAR eine Erkennung lokaler Deadlocks, so daß nur noch globale Deadlocks einer gesonderten Behandlung bedürfen. Am einfachsten wäre für diese eine Timeout-Strategie oder eine Deadlock-Vermeidung (z.B. durch BOT-Zeitstempel, Zeitintervalle o.ä.) [8]. Eine Erkennung globaler Deadlocks kann entweder durch ein nachrichtenbasiertes Protokoll oder unter Nutzung geeigneter Datenstrukturen im GEH erfolgen. Allerdings wäre ein globaler Wartegraph mit GEH-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 GEH durch die Angaben in den lokalen Sperrtabellen rekonstruierbar ist (Fehlertoleranz). So können z.B. auch nach einem Rechnerausfall die GEH-Sperreinträge, die der ausgefallene Rechner in Bearbeitung hatte, durch die Sperrangaben in den überlebenden VAR 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 GEH die Gesamtzugriffshäufigkeit zum GEH signifikant reduziert werden. So sind statt etwa 10 lediglich 4 bis 5 GEH-Zugriffe pro Sperre erforderlich; bei doppelter Führung der Sperrtabelle fallen pro Sperre weitere 2 (statt mindestens weiteren 4) GEH-Zugriffe an. Dies entspricht bei einer Zugriffszeit von 10 µs pro GEH-Eintrag einer Antwortzeiterhöhung von unter 1 ms.

Die GEH-Zugriffshäufigkeit kann noch weiter reduziert werden, wenn nicht jede Sperranforderung über den GEH entschieden wird, sondern die VAR zur lokalen Vergabe von Sperren autorisiert werden. Ein solch erweitertes Sperrverfahren kann analog zu Vorschlägen zur Reduzierung globaler Sperranforderungen bei nachrichtenbasierten Protokollen realisiert werden, z.B. durch Verwendung sogenannter Schreib- und Leseautorisierungen [8, 9]. So erlaubt eine Leseautorisierung, die gleichzeitig mehreren Rechnern gewährt sein kann, die lokale Vergabe und Freigabe von Lesesperren (d.h. ohne auf die globale Sperrtabelle im GEH zugreifen zu müssen). Bei einer Schreibautorisierung, die für ein Objekt jeweils nur einem VAR zuerkannt werden kann, ist eine vollkommen lokale Sperrbehandlung (Lese- und Schreibsperren) möglich. Mit einer solchen Verfahrenserweiterung läßt sich die Anzahl der GEH-Zugriffe stark reduzieren, allerdings auf Kosten einer erheblich gesteigerten Komplexität des Sperrprotokolls. Außerdem führt der Entzug der Autorisierungen (nach einer unverträglichen Sperranforderung durch einen anderen Rechner) zu zusätzlichen Nachrichten, die u.U. stärker ins Gewicht fallen als die eingesparten GEH-Zugriffe.

5. ZUSAMMENFASSUNG

In diesem Aufsatz wurde ein spezieller Speichertyp GEH vorgeschlagen, der eine effiziente Kooperation von mehreren unabhängigen Datenbanksystemen im Rahmen von Hochleistungs-Transaktionsanwendungen unterstützt. Eine grobe Auswertung bereits existierender Technologien (EH und SSD) zeigte, daß mit diesen wesentlichen Systemfunktionen im Hochleistungsfall nicht ausreichend schnell abgewickelt werden können. Prägende Merkmale dieses neuen Speichertyps sind vielmehr Nicht-Flüchtigkeit, gemeinsamer und sehr schneller Zugriff über eine synchrone Schnittstelle sowie die Verfügbarkeit verschiedener Zugriffsgranulate für Datenseiten und Verwaltungsdaten.

Die Diskussion der Einsatzmöglichkeiten des GEH zeigte, daß mit ihm ein breites Spektrum an wichtigen Systemfunktionen bei einer nahen Rechnerkopplung realisiert werden kann. Dabei wurden leistungsfähige Verfahren zum Nachrichtenaustausch und zur systemweiten Synchronisation genauer diskutiert. Einerseits scheint der bloße Einsatz des GEH zum Nachrichtenaustausch nicht alle seine Eigenschaften angemessen auszunutzen, andererseits aber scheiden Realisierungsformen aus Leistungsgründen aus, die alle globalen Systemdaten ausschließlich im GEH halten. Am Beispiel eines Sperrprotokolls wurde deshalb ein vielversprechendes Schema entwickelt, bei dem komprimierte globale Sperrinformation im GEH und lokale Information in den Speichern der beteiligten Rechnern geführt wird (Kap. 4.3). Quantitative Untersuchungen von verschiedenen Variationen dieses Schemas müssen einen näheren Aufschluß über seine Tauglichkeit liefern.

LITERATUR

[1] Anon et al.: A Measure of Transaction Processing Power, Datamation, April 1985, 112-118

[2] D.M. Dias, B.R. Iyer, J.T. Robinson, P.S. Yu: *Integrated Concurrency-Coherency Controls for Multisystem Data Sharing.* IEEE Trans. on Software Engineering 15(4), 1989, 437-448

[3] 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

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

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

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

[7] 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

[8] E. Rahm: *Synchronisation in Mehrrechner-Datenbanksystemen - Konzepte, Realisierungsformen und quantitative Bewertung* -. Informatik-Fachberichte 186, Springer-Verlag, 1988

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

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

[11] A.J. Smith: *Disk Cache - Miss Ratio Analysis and Design Considerations.* ACM Trans. on Computing Systems 3(3), 1985, 161-203

[12] 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.