Hauptbeiträge

Informatik Spektrum
© Springer-Verlag 1989

Informatik-Spektrum (1989) 12: 65-81

Der Database-Sharing-Ansatz zur Realisierung von Hochleistungs-Transaktionssystemen

E. Rahm

IBM, Yorktown Heights

Zusammenfassung. Database Sharing (DB-Sharing) bezeichnet eine allgemeine Klasse von Mehrrechner-Datenbanksystemen, bei der jeder Rechner direkten Zugriff auf die gemeinsamen Datenbestände hat. Hinsichtlich der Realisierung von Hochleistungs-Transaktionssystemen bieten DB-Sharing-Systeme signifikante Vorteile gegenüber Mehrrechner-Datenbanksystemen, bei denen eine statische Datenpartitionierung vorzunehmen ist. Dieser Aufsatz diskutiert diejenigen Systemkomponenten, für die bei DB-Sharing zur Nutzung des hohen Leistungspotentials neue und koordinierte Lösungskonzepte erforderlich sind. Insbesondere hinsichtlich der Synchronisation sowie der Behandlung von Pufferinvalidierungen werden die vielversprechendsten Realisierungsformen überblickartig vorgestellt und bewertet. Außer Aspekten der Lastkontrolle und Recovery werden auch Optimierungsmöglichkeiten durch Einsatz einer nahen Rechnerkopplung (‚closely coupled systems') betrachtet.

Schlüsselwörter: Transaktionssysteme, Mehrrechner-Datenbanksysteme, Database Sharing, Synchronisation, nahe Kopplung

Summary. Database Sharing (DB-Sharing) refers to a general class of multiprocessor database systems where each node has direct access to the shared databases. With respect to building a high-performance transaction system, the DB-Sharing approach offers significant advantages compared to distributed architectures with a static database partitioning. This paper discusses the functional components of a DB-Sharing system requiring new and coordinated solutions in order to exploit the high performance potential. In particular, we survey and evaluate the most-promising concepts for concurrency control and the treatment of buffer invalidations. Further considerations deal with load control, recovery and potential performance improvements by a close coupling of the nodes.

Key words: Transaction systems, Multiprocessor database systems, Database sharing, Concurrency control, Closely coupled systems

Computing Reviews Classification: H.2.4, C.2.4, D.4.1

1. Einführung

Der ständig wachsende Einsatz von Transaktionssystemen [17], etwa im Rahmen von Buchungs- und Auskunftsanwendungen bei Banken, Versicherungen oder Fluggesellschaften, führt zunehmend zu Anforderungen, die von zentralen Datenbanksystemen (DBS) nicht mehr erfüllt werden können. Die Hauptanforderungen an solche Hochleistungs-Transaktionssysteme sind hohe Transaktionsraten (mehrere 1000 Transaktionen vom Typ ‚Kontenbuchung' [3] pro Sekunde), sehr kurze Antwortzeiten, hohe Verfügbarkeit, modulare Wachstumsfähigkeit sowie einfache Handhabbarkeit und Verwaltung [14, 19]. Die Verwendung eines einzigen Rechners zur Transaktionsverarbeitung kann weder die Verfügbarkeits- und Erweiterbarkeitsforderungen erfüllen noch die für die genannten Transaktionsraten erforderliche Verarbeitungskapazität (mehrere 100 MIPS) erbringen.

Für den daher für die Realisierung von Hochleistungs-Transaktionssystemen zwingend erforderlichen Einsatz von *Mehrrechner-Datenbanksystemen* kommen verschiedene Architekturen in Betracht, die in [19] klassifiziert und gegenübergestellt wurden. Neben spezielleren Architekturen wie Datenbankmaschinen [8] eignen sich vor allem drei allgemeine Mehrrechner-Architekturen für die genannten Anforderungen, nämlich *eng gekoppelte Mehrrechner-DBS* (oft auch kurz als ‚shared memory' oder ‚shared everything' bezeichnet), Database Sharing [39] oder *DB-Sharing* (‚shared disk') sowie sogenannte *DB-Distribution*-Systeme (‚shared nothing') [19, 49].

In eng gekoppelten Mehrrechner-DBS findet die Transaktionsverarbeitung auf einem eng gekoppelten Multiprozessor statt, wobei sich alle Prozessoren einen gemeinsamen Hauptspeicher sowie die gesamte Peripherie (Terminals, Platten etc.) teilen. Software-Komponenten wie das Betriebssystem oder das Datenbankverwaltungssystem (DBVS) liegen nur in einer Kopie im gemeinsamen Speicher vor, die von allen Prozessoren benutzt wird. Die Verfügbarkeit und Erweiterbarkeit dieses Ansatzes wird durch die Verwendung eines gemeinsamen Hauptspeichers bzw. das Verbindungsnetzwerk zwischen den Prozessoren und dem Haupt-

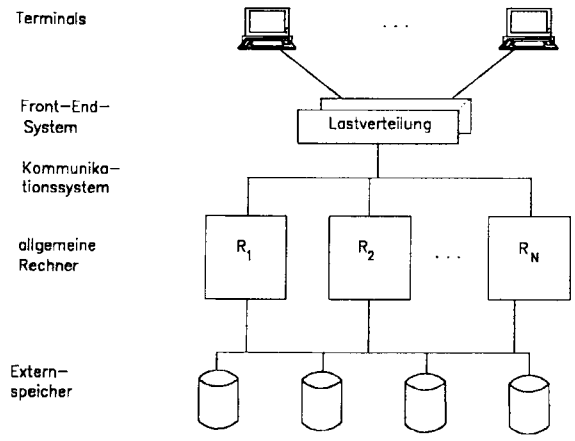

Abb. 1. Grobstruktur eines DB-Sharing-Systems

speicher (zumeist Busse) oft beeinträchtigt, obwohl neuere Systementwicklungen (z. B. Sequoia [6]) hier Fortschritte versprechen. Dennoch ist auch die Prozessorleistung von Multiprozessoren für höchste Transaktionsraten i. a. nicht ausreichend, zumal der Durchsatzbedarf oft stärker steigt als der technologische Zuwachs bei der CPU-Kapazität.

Diese Gründe führen dazu, daß die Mehrrechner-Architekturen DB-Sharing (Abb. 1) und DB-Distribution, bei denen jeder Knoten ein eng gekoppelter Multiprozessor sein kann, als am vielversprechendsten zur Realisierung von Hochleistungs-Transaktionssystemen anzusehen sind. Beide Architekturen bestehen aus einer Menge von autonomen Verarbeitungsrechnern mit eigenem Hauptspeicher sowie einer separaten Kopie von Betriebssystem sowie DBVS. Die Forderungen nach hoher Leistungsfähigkeit und einfacher Bedienbarkeit verlangen dabei i. a. eine räumlich benachbarte Aufstellung der Rechner, da dies ein sehr schnelles Kommunikationssystem (z. B. 10–100 MB/s) sowie eine zentrale Administration ermöglicht. Außerdem kann dann die Transaktionslast unter Berücksichtigung der aktuellen Rechnerauslastung auf die Knoten verteilt werden (z. B. über ein Front-End-System, Abb. 1). Die Kommunikation zwischen Rechnern findet in der Regel über Nachrichtenaustausch statt (lose Kopplung), kann aber auch über gemeinsame (Halbleiter-) Speicherbereiche oder Spezialprozessoren erfolgen (nahe Kopplung [19], s. Kap. 6).

Der wesentliche Unterschied zwischen DB-Sharing und DB-Distribution besteht in der Zuordnung der Externspeicher zu den Rechnern. Bei DB-Sharing hat jeder Rechner, wie in Abb. 1 angedeutet, direkten Zugriff auf alle Plattenlaufwerke und damit auf die gesamte Datenbank („shared disk"). Diese direkte Plattenanbindung ist nur bei lokaler Rechneranordnung möglich. Bei DB-Distribution liegt dagegen eine Partitionierung der Externspeicher und damit der Daten vor, so daß jeder Knoten nur auf seine Partition direkt zugreifen kann. Zur Klasse der DB-Distribution-Systeme gehören auch verteilte Datenbanksysteme [4], bei denen eine ortsverteilte Anordnung der Verarbeitungsrechner vorliegt und Teile der Datenbank repliziert an verschiedenen Knoten gespeichert sein können. Eine Replikation der Daten in geographisch verteilten Knoten ist auch Voraussetzung für eine schnelle Katastrophen-Recovery [15].

Dieser Aufsatz behandelt den DB-Sharing-Ansatz zur Transaktionsverarbeitung, da er trotz einiger existierender Systeme (s. u.) im Vergleich zu DB-Distribution und verteilten Datenbanksystemen noch relativ unbekannt ist. DB-Sharing verspricht darüber hinaus eine Reihe von Vorteilen gegenüber DB-Distribution, weil es nicht notwendig ist, eine physische und nur schwer zu ändernde Partitionierung und Allokierung der Datenbank vorzunehmen[1]. Die wichtigsten sich daraus ergebenden Unterschiede können wie folgt zusammengefaßt werden (für einen ausführlichen Vergleich siehe [19, 20, 38]):

- Der DB-Sharing-Ansatz bietet Vorteile hinsichtlich Verfügbarkeit nach einem Rechnerausfall sowie Erweiterbarkeit des Systems. Fällt nämlich ein Knoten aus, so können die überlebenden Rechner (nach bestimmten Recovery-Aktionen) weiterhin auf die gesamte DB zugreifen. Bei DB-Distribution dagegen ist die DB-Partition des ausgefallenen Rechners zunächst nicht mehr erreichbar. Übernimmt ein anderer Knoten die betroffene Partition, so wird der leicht überlastet. Ebenso ist bei DB-Distribution eine Umverteilung der Daten notwendig, wenn das System um einen Rechner erweitert werden soll.
- Eine ausreichende Flexibilität zur Partitionierung des Datenbestandes ist allenfalls bei relationalen DBS gegeben. Von nicht-relationalen Datenbanken dagegen werden i. a. nur grobe Verteileinheiten unterstützt (Segmente, Satztypen u. ä), mit der Einschränkung, daß jede DB-Operation nur Daten einer Partition berühren darf. Für solche Systeme kann eine Umverteilung der Daten (sofern möglich) sogar die Anpassung von Anwendungsprogrammen erfordern. Der DB-Sharing-Ansatz eignet sich dagegen für relationale und nicht-relationale DBS gleichermaßen. DB-Sharing kann als natürliche Weiterentwicklung zentralisierter DBS aufgefaßt werden, wobei der Übergang zum Mehrrechnersystem weder eine Änderung bestehender Datenbanken noch von Anwendungsprogrammen erfordert.
- Das Transaktionsverarbeitungsmodell unterscheidet sich erheblich für die beiden Architekturklassen. Bei DB-Distribution wird eine DB-Operation i. a. dort ausgeführt, wo die Daten residieren. Dies führt zu einer verteilten Transaktionsausführung, wenn externe Daten referenziert werden, verbunden mit dem Verschicken von Beauftragungen zur externen Verarbeitung einer (Teil-) Operation sowie einem verteilten Commit-Proto-

[1] Keine Datenverteilung ist bei DB-Distribution natürlich vorzunehmen, wenn die Datenbank an jedem Knoten vollkommen repliziert gespeichert wird. Dieser Ansatz scheidet in der Praxis jedoch i. a. wegen des extremen Änderungsaufwandes sowie der hohen Speicherkosten aus

koll. Bei DB-Sharing dagegen kann jede DB-Operation einer Transaktion lokal ausgeführt werden, da jeder Rechner direkten Zugriff auf die gesamte Datenbank hat[2]. Kommunikationsvorgänge zur Transaktionsbearbeitung können allerdings auch bei DB-Sharing notwendig werden, z. B. zur Synchronisation.

• DB-Sharing verspricht ein weit größeres Optimierungspotential hinsichtlich Lastverteilung und -balancierung. Bei DB-Distribution bestimmt die statische DB-Partitionierung bereits weitgehend die Auslastung der Rechner sowie die Kommunikationshäufigkeit, da jeder Rechner sämtliche Operationen lokaler und externer Transaktionen auf die ihm zugeordnete Datenpartition ausführen muß. Schwankungen im Lastprofil führen so unweigerlich zu ungleichmäßigen Rechnerauslastungen (oder gar Überlastung einzelner Knoten) und den damit verbundenen Leistungseinbußen. Bei DB-Sharing dagegen bestimmt keine statische Datenverteilung die Auslastung der Rechner; eine Transaktion kann in jedem Rechner weitgehend lokal bearbeitet werden. Hier ist es sogar möglich, bei Unterlast ganze Rechner für andere Aufgaben ‚abzustellen' [47].

Die älteste DB-Sharing-Realisierung ist das auf zwei Rechner beschränkte IMS Data Sharing [25], das jedoch keinen Anspruch auf Hochleistungseigenschaften erheben kann. Neuere System- und Prototyp-Entwicklungen, die verbesserte Leistungs- und Fehlertoleranzmerkmale anstreben, verkörpern u.a. das Power System 5/55 von Computer Console [52], das AIM/SRCF-System (Shared Resource Control Facility) von Fujitsu [1], das Data Sharing Control System (DCS) von NEC [45] sowie das Amoeba-Projekt [46, 51]. Die DEC VAX-Cluster [27] bieten zwar auch eine ‚shared disk'-Architektur, jedoch ohne Unterstützung durch ein entsprechendes Datenbanksystem (kein Transaktionskonzept). Das Spezialbetriebssystem TPF (Transaction Processing Facility), mit dem die derzeit wohl höchsten Transaktionsraten erzielt werden, erlaubt auch ein ‚disk sharing'; hierbei sind allerdings die Anwendungssysteme für die Lösung nahezu aller Probleme verantwortlich [14, 44]. In den genannten Systemen können zwischen zwei und 16 Rechnerknoten eingesetzt werden. Dies dürfte wegen der notwendigen Plattenanbindung auch die typische Größenordnung für DB-Sharing bleiben, wobei dies bei Verwendung leistungsfähiger Multiprozessor-Knoten (von z.B. 100-200 MIPS) für praktische Systeme keine Einschränkung für die absehbare Zukunft bedeutet.

Zur Nutzung des Leistungspotentials des DB-Sharing-Ansatzes sind für eine Reihe von DBVS-Funktionen sowie bezüglich Lastkontrolle und Systemadministration neue und koordinierte Lösungskonzepte zu entwickeln. Dieser Aufsatz diskutiert im folgenden die neuen Anforderungen sowie Lösungsmöglichkeiten im Bereich der Synchronisation (Kap. 2), der Systempufferverwaltung (Behandlung von Pufferinvalidierungen, Kap. 3), der Lastkontrolle (Kap. 4) sowie für Logging und Recovery (Kap. 5). Während in diesen Kapiteln eine lose Kopplung der Verarbeitungsrechner vorausgesetzt wird, diskutiert Kapitel 6 dann noch mögliche Optimierungen durch eine nahe Rechnerkopplung bei DB-Sharing. Die in Kap. 2 und 3 gegebene Übersicht über einsetzbare Protokolle basiert auf [38], wo die Verfahren ausführlich beschrieben sind und die Leistungsfähigkeit ausgewählter Protokolle im Rahmen einer empirischen Simulationsstudie quantitativ bewertet wird.

2. Synchronisation in DB-Sharing-Systemen

Der Zugriff der Rechner auf die gemeinsamen Datenbanken erfordert offensichtlich eine Synchronisation, um die Korrektheit der Transaktionsausführung (globale Serialisierbarkeit [7]) zu gewährleisten[3]; hierzu sind bei loser Kopplung Kommunikationsvorgänge zwischen den Rechnern erforderlich. Die Hauptanforderungen an ein geeignetes Synchronisationsprotokoll sind außer Korrektheit Robustheit gegenüber Rechnerausfällen (nach Ausfall eines Knotens muß eine korrekte Fortführung der Synchronisation gewährleistet sein) sowie Effizienz. Ein effizientes Synchronisationsverfahren muß vor allem folgende Eigenschaften aufweisen:

• Geringer Kommunikationsaufwand.
Die Minimierung der Nachrichtenhäufigkeit zur Synchronisation ist für DB-Sharing zur Erlangung eines hohen Durchsatzes, der durch den Kommunikations-Overhead geschmälert wird, sowie kurzer Antwortzeiten gleichermaßen von Bedeutung. Für die Antwortzeiten besonders kritisch sind *synchrone* Nachrichten, für die eine Transaktion (TA) bis zum Eintreffen einer Antwortnachricht unterbrochen wird (z.B. externe Sperranforderung). Um eine möglichst geringe Gesamtbelastung durch Kommunikationsvorgänge zu erreichen, ist auch eine *integrierte Lösung für Synchronisation und Behandlung von Pufferinvalidierungen* anzustreben (Kap. 3).

• Geringe Häufigkeit von Synchronisationskonflikten (TA-Blockierungen bzw. -Rücksetzungen).
Diese Forderung ist für Mehrrechner-DBS aufgrund der systemweit höheren Parallelität (höhere Anzahl paralleler TA-Aktivierungen) i.a. schwieriger zu errei-

[2] Obwohl in diesem Aufsatz nicht näher untersucht, könnte jedoch auch bei DB-Sharing eine verteilte Transaktionsausführung angewendet werden, z.B. um eine Parallelbearbeitung einer komplexen Transaktion zu ermöglichen. Dies ist allerdings bei Verwendung von Multiprozessoren auch innerhalb eines Knotens möglich, ohne die zusätzlichen Kommunikationsvorgänge zum Starten und Beenden der parallel zu bearbeitenden Verarbeitungsfolgen in Kauf zu nehmen

[3] Im wesentlichen ist dazu einer Transaktion die parallele Verarbeitung konkurrierender Transaktionen (Mehrbenutzerbetrieb) zu verbergen (logischer Einbenutzerbetrieb). Das Korrektheitskriterium der Serialisierbarkeit verlangt hierzu, daß das Ergebnis der parallelen Transaktionsausführung dem Ergebnis einer der möglichen seriellen Bearbeitungsreihenfolgen der beteiligten Transaktionen entspricht

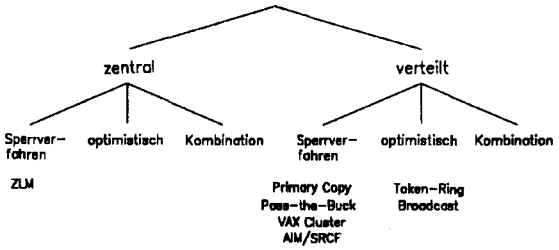

Abb. 2. Synchronisationsverfahren für DB-Sharing (nach [38])

chen als in zentralisierten DBS. Längere Antwortzeiten aufgrund von Kommunikationsvorgängen führen zu einer weiteren Steigerung der Konfliktrate. Die Häufigkeit von Synchronisationskonflikten kann u. a. durch Wahl eines feinen Synchronisationsgranulates, geringere Konsistenzzusicherungen als Serialisierbarkeit (z. B. Konsistenzebene 2 [16], wobei eine TA während ihrer Ausführung möglicherweise verschiedene Werte desselben Objektes sieht) oder Einsatz von Spezialprotokollen für bestimmte Objekt- bzw. TA-Klassen (Hot-Spot-Behandlung, Sonderbehandlung von Lesetransaktionen u. ä) reduziert werden.

Von den für verteilte DBS vorgeschlagenen Synchronisationsverfahren [7, 10] können viele auf die DB-Sharing-Umgebung übertragen werden; jedoch ergeben sich dann wegen des unterschiedlichen Verarbeitungsmodells sowie unterschiedlicher Anforderungen (integrierte Behandlung von Pufferinvalidierungen) meist uninteressante Lösungen mit hohem Kommunikationsbedarf. So können z. B. reine Zeitmarkenverfahren [7] von den Überlegungen hier ausgeschlossen werden, da sie im Vergleich zu Sperrverfahren keine Kommunikationseinsparungen erlauben, jedoch i. a. weit mehr Rücksetzungen verursachen.

Abbildung 2 zeigt eine Übersicht über die wichtigsten Synchronisationsverfahren, die bisher für DB-Sharing vorgeschlagen bzw. implementiert wurden. Wie gezeigt, handelt es sich dabei um Sperrverfahren, optimistische Protokolle sowie kombinierte (optimistische und pessimistische) Ansätze, die jeweils unter zentraler oder verteilter Kontrolle ablaufen können. In 2.1 wird zunächst näher auf die Sperrverfahren eingegangen, während die anderen Protokolle in 2.2 diskutiert werden. Verfahrenserweiterungen wie Spezialbehandlung von sogenannten Hot-Spot- bzw. High-Traffic-Objekten, auf die sehr häufig (ändernd) zugegriffen wird, oder Einsatz eines Mehrversionenkonzeptes können hier aus Platzgründen nicht betrachtet werden (s. [38]).

2.1. *Sperrverfahren für DB-Sharing*

In sämtlichen kommerziellen DBS sowie in den existierenden DB-Sharing-Systemen werden Sperrverfahren zur Synchronisation eingesetzt. Wir wollen hier vor allem zwei Sperrverfahren für DB-Sharing näher diskutieren, nämlich den Einsatz eines zentralen Sperrverwalters sowie das Primary-Copy-Protokoll. Danach wird noch eine generelle Optimierung zur Behandlung von Lesesperren angegeben sowie auf die in existierenden DB-Sharing-Systemen verwendeten Protokolle eingegangen. Bei den Verfahren wird unterstellt, daß eine TA Lese- bzw. Schreibsperren für zu referenzierende Objekte anfordert und die Sperren zur Sicherstellung der Serialisierbarkeit bis zum TA-Ende gehalten werden. Zur Behandlung von Deadlocks können dieselben Techniken wie in verteilten DBS angewendet werden (für eine Übersicht s. [7, 26]).

Zentraler Sperrverwalter

Das wohl einfachste Synchronisationsverfahren für DB-Sharing besteht darin, jede Sperranforderung und -freigabe an einen zentralen Sperrverwalter oder Lock-Manager (ZLM) zu schicken, der auf einem der Knoten residiert und eine globale Sperrtabelle zur Synchronisation verwaltet. Damit fallen zwei Nachrichten pro Sperranforderung an sowie eine weitere Nachricht am Transaktionsende, um alle Sperren freizugeben. Dieser offenbar zu hohe Kommunikationsaufwand kann am einfachsten durch eine gebündelte Übertragung von Sperranforderungen reduziert werden. Allerdings ist dies nur bedingt sinnvoll, da sich durch die Bündelungsverzögerungen die Antwortzeiten und damit die Sperrhaltedauern und die Konfliktgefahr erhöhen.

Zur Reduzierung des Kommunikationsaufwandes sowie der Antwortzeiten wurde in [41] die Verwendung eines sogenannten *Sole-Interest-Konzeptes* vorgeschlagen, bei dem neben dem zentralen Sperrverwalter in jedem Knoten ein lokaler Sperrverwalter vorgesehen wird. Wenn der ZLM nun eine Sperre gewährt, teilt er sogleich mit, ob auf dem betreffenden Objekt ‚sole interest' vorliegt; dies ist der Fall, wenn beim ZLM für das Objekt keinerlei Sperranforderung seitens eines anderen Rechners vorliegt. Eine Sole-Interest-Situation autorisiert nun den lokalen Sperrverwalter, alle weiteren Sperranforderungen und -freigaben lokaler TA für das betreffende Objekt (z. B. DB-Seite) lokal (d. h. ohne Kommunikation mit dem ZLM) zu behandeln. Diese Technik läßt sich auf ein hierarchisches Sperrverfahren erweitern, bei dem Sole-Interest-Zuweisungen für unterschiedliche Sperrgranularitäten unterstützt werden. Besitzt z. B. ein Knoten ‚sole interest' für einen gesamten Satztyp, dann wird eine lokale Synchronisierung für sämtliche Sätze bzw. DB-Seiten dieses Satztypes möglich.

Im Gegensatz zu Lese- und Schreibsperren, die von einzelnen TA angefordert und freigegeben werden, handelt es sich bei den Sole-Interest-Zuweisungen um Autorisierungen zur lokalen Sperrvergabe auf Rechnerebene. Diese Autorisierungen werden daher von den lokalen Sperrverwaltern auch dann noch gehalten, wenn zeitweilig keine Sperranforderungen für die betreffenden Objekte in dem Rechner vorliegen, um im Falle einer späteren Rereferenzierung eine lokale Synchronisierung

zu erlauben. Andererseits verursachen Sole-Interest-Zuweisungen eine verzögerte Gewährung von Sperranforderungen anderer Rechner, da in diesem Fall der ZLM zunächst die Sole-Interest-Zuweisung zurücknehmen muß (zusätzliche Kommunikationsverzögerungen). Das Sole-Interest-Konzept kann daher nur dann eine nennenswerte Reduzierung der Kommunikationshäufigkeit bewirken, wenn die Anzahl der wegen ‚sole interest' lokal gewährten Sperranforderungen deutlich höher ist als die Anzahl solcher Sole-Interest-Entziehungen.

Die Wirksamkeit des Sole-Interest-Ansatzes hängt somit entscheidend davon ab, inwieweit nur TA eines Rechners dieselben Objekte bzw. DB-Bereiche referenzieren, d.h. vom Ausmaß *rechnerspezifischer Lokalität* im Referenzverhalten. Inwieweit dies erreicht werden kann, wird außer von den Charakteristika der TA-Last vor allem von der Strategie zur Lastverteilung bestimmt, deren Ziel es sein muß, die TA so auf die Rechner zu verteilen, daß in unterschiedlichen Rechnern möglichst verschiedene DB-Bereiche referenziert werden. Dies ist mit zunehmender Rechneranzahl jedoch immer schwieriger zu erreichen, insbesondere wenn, wie in realen Anwendungen oft der Fall, häufig referenzierte DB-Bereiche oder dominierende TA-Typen vorkommen, die zur Vermeidung von Überlastsituationen von mehreren Rechnern bearbeitet werden müssen. In diesen Fällen ist nur für DB-Bereiche mit geringerer Zugriffsfrequenz mit Sole-Interest-Zuweisungen längerer Zeitdauer zu rechnen (geringe Kommunikationseinsparungen), während für die wichtigeren DB-Bereiche in der Regel die langsame Sperrbehandlung über den ZLM vorzunehmen ist. Zudem ist hier die Gefahr ständig wechselnder Sole-Interest-Zuweisungen gegeben, wobei das jeweils notwendige Entziehen der Autorisierungen zu einem hohen Zusatzaufwand führen kann. Das dahinter stehende Problem ist die Instabilität der Sole-Interest-Zuweisung, da z.B. eine einzige externe Referenz den Sole-Interest-Entzug für einen gesamten DB-Bereich verursachen kann.

Ein weiteres Problem des ZLM-Ansatzes ist die mit wachsenden TA-Raten steigende Engpaßgefahr im zentralen Knoten, der zudem einen ‚single point of failure' darstellt und daher spezielle Vorkehrungen für den Fehlerfall verlangt (z.B. redundantes Speichern der globalen Sperrtabelle in unabhängigen und möglicherweise nichtflüchtigen Speicherbereichen).

Primary-Copy-Sperrverfahren

Die Grundidee des Primary-Copy-Sperrverfahrens für DB-Sharing [35] ist denkbar einfach. Statt auf einen ZLM wird hier die globale Synchronisationsverantwortung auf alle Rechner verteilt. Hierzu wird eine *logische* Partitionierung der gemeinsamen Datenbank(en) vorgenommen und jedem Knoten die Zuständigkeit oder *primary copy authority* (PCA) für eine Partition zugeordnet. Jeder Knoten kann daher sämtliche DB-Zugriffe bezüglich seiner Partition lokal synchronisieren, während für Objekte anderer DB-Partitionen Sperranforderungen und -freigaben an den jeweils zuständigen PCA-Rechner zu schicken sind. Die PCA-Zuordnung ist jedem Rechner (DBVS) bekannt.

Um die Häufigkeit externer Sperranforderungen einzugrenzen, gilt es, die PCA-Verteilung und die Lastverteilung derart aufeinander abzustimmen, daß eine TA möglichst an dem Knoten bearbeitet wird, an dem die meisten DB-Zugriffe lokal synchronisiert werden können. Ist dies (ohne Überlastung einzelner Knoten) möglich, so ergibt sich automatisch ein hohes Maß an rechnerspezifischer Lokalität, da dann jede der Partitionen vorwiegend von TA des PCA-Rechners referenziert wird. Der Vorteil gegenüber dem ZLM-Ansatz ist, daß die PCA-Zuordnungen im Gegensatz zu Sole-Interest-Zuweisungen stabil sind und nicht durch externe TA zerstört werden können. So ist bei dominierenden DB-Bereichen zwar auch beim Primary-Copy-Verfahren mit einem erhöhten Anteil an nicht-lokalen Sperrbehandlungen zu rechnen, jedoch kann wenigstens im PCA-Rechner noch eine lokale Synchronisierung erfolgen, während bei dem Sole-Interest-Ansatz in solchen Fällen i.a. sämtliche Zugriffe global zu synchronisieren sind. Dies konnte im Rahmen einer umfassenden empirischen Simulationsstudie [38] bestätigt werden, wo mit einem Sole-Interest-Ansatz (und ZLM) meist mehr als doppelt so viele globale Sperranforderungen als bei Einsatz einer ‚geeigneten' PCA-Verteilung verursacht wurden (abhängig v.a. von der TA-Last und der Rechneranzahl). Eine weitere wichtige Beobachtung war, daß sich rechnerspezifische Lokalität im Referenzverhalten (z.B. infolge einer Abstimmung zwischen TA- und PCA-Verteilung) nicht nur zum Einsparen von globalen Sperranforderungen nutzen läßt, sondern auch zu besseren Trefferraten im Systempuffer (E/A-Einsparungen) und weniger Pufferinvalidierungen (s. Kap. 3) führt.

Ein wesentlicher Unterschied zu DB-Distribution-Systemen besteht darin, daß beim Primary-Copy-Sperrverfahren die Partitionierung nur für Synchronisationszwecke vorgenommen wird, während die Daten weiterhin für jeden Rechner direkt erreichbar sind (lokale Ausführung der DB-Operationen). Die Flexibilität der DB-Sharing-Architektur bleibt somit erhalten; die Partitionierung ist lediglich durch interne Kontrollstrukturen repräsentiert, die relativ leicht wechselnden Anforderungen angepaßt werden kann (z.B. bei geänderter Rechneranzahl). Auch können die Verteileinheiten oder Fragmente, die zunächst festzulegen sind und dann bei der Bestimmung der PCA-Verteilung zu Partitionen zusammengefaßt werden, nahezu beliebig fein gewählt werden (im Gegensatz zu den physischen Verteileinheiten bei DB-Distribution). Die Bestimmung bzw. Anpassung der PCA-Verteilung kann z.B. zusammen mit der Festlegung oder Änderung der globalen Strategie zur Lastverteilung erfolgen (s. Kap. 4).

Optimierte Behandlung von Lesesperren

Das Sole-Interest-Konzept sowie das Primary-Copy-Sperrverfahren zielen beide darauf ab, rechnerspezifi-

sche Lokalität im Referenzverhalten für eine möglichst lokale Synchronisierung zu nutzen. Für die Wirksamkeit beider Ansätze (vor allem beim Sole-Interest-Konzept) ergeben sich dadurch starke Abhängigkeiten bezüglich des Lastprofils sowie der Strategie zur Lastverteilung, so daß bei DB-Bereichen, auf die verstärkt von mehreren Knoten aus zugegriffen wird, ein hohes Nachrichtenaufkommen zu befürchten ist. Insbesondere ist mit wachsender Rechneranzahl mit einem steigenden Anteil an globalen Sperranforderungen zu rechnen, was der Forderung nach modularer Wachstumsfähigkeit entgegensteht. Eine Reduzierung dieser Schwierigkeiten wird möglich durch eine verbesserte Synchronisierung für Lesezugriffe, die vom Ausmaß rechnerspezifischer Lokalität unabhängig ist. Die im folgenden skizzierte Vorgehensweise zur optimierten Behandlung von Lesesperren ist dabei für praktisch alle Sperrverfahren bei DB-Sharing anwendbar, z. B. in Kombination mit einem ZLM-Ansatz, beim Primary-Copy-Sperrverfahren [35] oder für ein Pass-the-Buck-Protokoll (s. u.) [18].

Mit der Leseoptimierung soll es mehreren Rechnern zur gleichen Zeit ermöglicht werden, Lesesperren für ein Objekt lokal zu gewähren und freizugeben. Ein Rechner muß dazu den ersten Lesezugriff zu einem Objekt stets global synchronisieren, z. B. über einen ZLM oder den zuständigen PCA-Rechner. Wenn nun die beantragte Lesesperre gewährt werden kann und zu dem Zeitpunkt keine Schreibanforderung vorliegt, dann wird dem anfordernden Rechner mit Gewährung der Sperre eine sogenannte *Leseautorisierung* zugewiesen, die i. a. über das Ende der betreffenden TA hinaus gehalten wird. Eine Leseautorisierung berechtigt den Rechner dazu, alle weiteren Lesesperranforderungen und -freigaben für das Objekt lokal zu bearbeiten (ohne Kommunikation). Wie bei den Sole-Interest-Zuweisungen handelt es sich bei den Leseautorisierungen um eine Bevollmächtigung der Rechner, zur Einsparung von Synchronisationsnachrichten Sperranforderungen und -freigaben lokal zu behandeln. Während Sole-Interest-Zuweisungen exklusiv vergeben werden (und daher eine lokale Sperrvergabe von Lese- und Schreibsperren ermöglichen), können Leseautorisierungen für ein Objekt (bzw. einen DB-Bereich) von mehreren Knoten gleichzeitig gehalten werden. Eine Rückgabe der Leseautorisierungen wird veranlaßt, sobald eine Schreibanforderung gestellt wird, wodurch sich zusätzliche Verzögerungen für Änderungs-TA ergeben können.

Allerdings haben Simulationen mit empirischen Lasten gezeigt [18, 38], daß außer bei Anwendungen mit hohem Anteil an Schreibzugriffen ($\geq 50\%$) die Kommunikationseinsparungen aufgrund der Leseoptimierung (die um so höher ausfallen, je höher die Lokalität von Lesezugriffen ist) etwaige Verzögerungen für Schreibzugriffe bei weitem übertreffen. Insbesondere wenn zur Reduzierung der Sperrkonfliktwahrscheinlichkeit Lesesperren nur kurz gehalten werden (Konsistenzebene 2), ist die Leseoptimierung als unabdingbar anzusehen, da sie i. a. verhindert (wie sonst möglich), daß eine TA für dasselbe Objekt mehrere Lesesperren global anfordert. Die Leseoptimierung erleichtert so eine modulare Wachstumsfähigkeit und erlaubt auch bei dominanten TA-Typen und häufig referenzierten DB-Bereichen eine weitgehend lokale Synchronisierung, solange auf die Bereiche von rechnerübergreifendem Interesse vorwiegend lesend zugegriffen wird.

Sperrverfahren in existierenden DB-Sharing-Systemen

Ein Primary-Copy-Sperrverfahren sowie die Leseoptimierung werden in existierenden DB-Sharing-Systemen noch nicht verwendet. Ein zentrales Sperrprotokoll wird dagegen trotz der genannten Schwierigkeiten in mehreren DB-Sharing-Systemen eingesetzt, so von Computer Console [52], dem DCS [45] und im Amoeba-Prototyp [46]. In den beiden erstgenannten Systemen wird ein Sole-Interest-Ansatz auf Dateiebene zur Einsparung globaler Sperranforderungen unterstützt. Auf die Sperrbehandlung beim DCS sowie bei TPF wird in Kap. 6 noch eingegangen.

In Abb. 2 sind noch drei weitere (verteilte) Sperrverfahren für DB-Sharing genannt, die in existierenden Systemen implementiert sind, jedoch als weniger erfolgversprechend einzustufen sind:

- Bei *IMS Data Sharing*, das auf zwei Knoten beschränkt ist, wird ein Token-Ring-ähnliches Verfahren eingesetzt (Pass-the-Buck), bei dem jeder Knoten für eine festgelegte Dauer das Token hält und nach Ablauf der Zeitspanne alle lokal nicht entscheidbaren Sperranforderungen mit dem Token (in einem sogenannten Buck) gebündelt an den anderen Rechner weiterleitet [25, 56]. Um die Anzahl der globalen Sperranforderungen einzugrenzen, wird im Rahmen einer Hash-Tabelle für jede Hash-Klasse ein Interest-Bit pro Rechner geführt, das anzeigt, ob in dem Rechner eine Sperranforderung für ein Objekt der Hash-Klasse gestellt wurde. Damit kann eine Sperranforderung lokal gewährt werden, wenn die Hash-Tabelle anzeigt, daß im anderen Rechner für kein Objekt der betreffenden Hash-Klasse eine Sperranforderung vorliegt (‚sole interest' für die Hash-Klasse).

  Der Hauptnachteil des Verfahrens ist die Beschränkung der Rechneranzahl; eine Verallgemeinerung auf mehr als zwei Knoten ist zwar möglich, jedoch ergibt sich dann ein höherer Anteil an globalen Sperranforderungen und eine verlängerte Wartezeit, bis eine globale Sperre gewährt wird (verlängerte Token-Umlaufzeit). Eine empirische Simulationsstudie über ein verbessertes Pass-the-Buck-Protokoll [18] zeigte zudem, daß die Wahl einer festen Token-Verweilzeit zu inflexibel ist: Ein kleiner Wert führt zu einem hohen Kommunikations-Overhead, während sich bei längeren Verweilzeiten die Wartezeiten bis zur Gewährung globaler Sperren erhöhen.
- Im *AIM/SRCF-System* wird ein spezielles Majority-Voting-Protokoll [50] zur Synchronisation eingesetzt, bei dem jede Sperranforderung von einer Mehrheit der Rechner gewährt werden muß. Die Folge davon ist, daß

keine Sperre lokal vergeben werden kann, sondern jede Sperranforderung Kommunikationsverzögerungen unterworfen ist.

• Ähnlich wie bei PCL, existiert im Sperrprotokoll der *DEC VAX Cluster* [27] für ein Objekt ein sogenannter Master-Knoten, der die Synchronisationsverantwortung für das Objekt hat. Allerdings wird hier die Master-Zuordnung nicht vorab bestimmt, sondern dynamisch festgelegt. Dabei wird der Rechner, der zuerst eine Sperre für ein Objekt anfordert, automatisch Master und synchronisiert alle weiteren Objektzugriffe. Die Master-Zuordnung wird nicht in einer replizierten Tabelle gespeichert (wie bei PCL), sondern in einem Directory, das gemäß einer Hash-Abbildung über alle Rechner verteilt ist. Diese Indirektion führt zu einem vermehrten Kommunikationsaufwand, da vor einer Sperranforderung für ein Objekt O zunächst der Rechner, der den für O zuständigen Directory-Teil führt, befragt werden muß, ob bereits ein Master-Knoten für O vorliegt und, wenn ja, welcher Knoten diese Funktion ausübt. Dies führt nach [27] zu 2 bis 4 Nachrichten pro Sperranforderung, was für Hochleistungssysteme als zu hoch anzusehen ist. Ein einfaches Primary-Copy-Protokoll, bei dem die PCA-Zuordnung vorab über eine Hash-Abbildung festgelegt wird, verursacht bereits weniger Nachrichten (für N Rechner im Mittel $2-2/N$ Nachrichten pro Sperranforderung, falls jeder Knoten dieselbe Anzahl von Hash-Klassen verwaltet und jede Hash-Klasse mit der gleichen Wahrscheinlichkeit referenziert wird, s. auch [23]).

2.2. Optimistische Synchronisationsverfahren für DB-Sharing

Optimistische Synchronisationsverfahren [28] wurden vor einigen Jahren als Alternative zu Sperrverfahren vorgeschlagen, insbesondere für Anwendungen mit geringer Konflikthäufigkeit (z. B. bei Dominanz von Lesezugriffen). Der Hauptunterschied zu Sperrverfahren liegt darin, daß die Ausführung der TA weitgehend unsynchronisiert erfolgt und erst am TA-Ende im Rahmen einer sogenannten Validierung etwaige Synchronisationskonflikte aufgedeckt werden, die hier generell durch Rücksetzen von TA aufgelöst werden. Bei erfolgreicher Validierung werden die Änderungen einer TA, die zunächst auf privaten Objektkopien vorgenommen werden, für alle TA sichtbar gemacht. Insbesondere hinsichtlich der Validierungstechniken wurden eine Fülle von Alternativen und Erweiterungen (auch auf verteilte Umgebungen) vorgeschlagen [37].

Vorteile des optimistischen Ansatzes gegenüber Sperrverfahren sind eine potentiell höhere effektive Parallelität (keine Sperrkonflikte) sowie Deadlock-Freiheit. Bisherige Leistungsuntersuchungen (z. B. [32]) zeigten jedoch, daß zumindest die einfachen optimistischen Verfahren oft zu inakzeptabel vielen Rücksetzungen führen und v. a. lange TA durch ständiges Zurücksetzen (starvation) vom erfolgreichen Ende abgehalten werden können. Eine Einsetzbarkeit optimistischer Verfahren über konfliktarme Anwendungen hinaus erfordert daher eine Kombination mit pessimistischen (Sperr-) Verfahren, um z. B. einer mit optimistischer Synchronisation gescheiterten TA durch Setzen von Sperren bei der zweiten Ausführung ein erfolgreiches Durchkommen zu ermöglichen. Interessanterweise läßt sich auch für optimistische Verfahren die Konflikthäufigkeit (hier: Anzahl von Rücksetzungen) erheblich reduzieren, wenn anstatt Serialisierbarkeit nur Konsistenzebene 2 zu gewährleisten ist. In diesem Fall brauchen nämlich nur noch Änderungs-TA zu validieren, während Lese-TA stets bei der ersten Ausführung erfolgreich zu Ende kommen (s. [38]).

Der Hauptanreiz, optimistische Verfahren für DB-Sharing zu untersuchen, liegt darin, daß sie gegenüber Sperrverfahren potentiell weniger Nachrichten erfordern und der Kommunikationsbedarf zur Synchronisation nicht so stark von Lastprofil, Lastverteilung und Rechneranzahl abhängt. Denn während der Ausführung einer TA fällt keine Synchronisationsnachricht an, sondern nur bei TA-Ende zur globalen Validierung. Somit wird nur eine Synchronisationsnachricht pro TA-Ausführung erforderlich (bei Konsistenzebene 2 sogar nur für Änderungs-TA), während Sperrverfahren mehrere globale Sperranforderungen pro TA verursachen können.

Die einfachste Realisierungsform eines optimistischen Protokolls wird möglich bei einer *zentralen Validierung*, wo bei TA-Ende die Validierungsaufforderung zu einem zentralen Validierungsknoten (ZVK) gesendet wird. Die Validierung im ZVK besteht im wesentlichen darin zu überprüfen, ob die von einer TA referenzierten Objektversionen zum Validierungszeitpunkt noch gültig sind, was durch Verwendung von Zeitstempeln einfach und effizient realisierbar ist [37, 38]. Wurde ein Objekt referenziert, das zwischenzeitlich von einer erfolgreich validierten TA geändert wurde, scheitert die Validierung und die betroffene TA wird zurückgesetzt.

Um ein mehrfaches Zurücksetzen einer TA zu vermeiden, kann vorgesehen werden, daß eine gescheiterte TA im ZVK für jedes bei der ersten Ausführung referenzierte Objekt eine Sperre setzt. Diese Sperren, die im ZVK unmittelbar nach der gescheiterten Validierung und vor der erneuten TA-Ausführung beantragt werden, verhindern, daß die Objekte (erneut) von anderen TA geändert werden. Die zweite Ausführung der TA ist somit erfolgreich, wenn auf keine weiteren Objekte zugegriffen wurde. Das Setzen der Sperren, das einem *preclaiming* gleichkommt, kann offenbar ohne zusätzliche Kommunikationsvorgänge und unter Vermeidung von Deadlocks vorgenommen werden. Eine (aufwendige) Alternative besteht darin, gefährdete TA (z. B. lange Änderungs-TA) von vorneherein pessimistisch zu synchronisieren, also bereits in der ersten Ausführung vor jedem Objektzugriff eine entsprechende Sperre im zentralen Knoten zu setzen.

Mit einer *verteilten Validierung* sollen v. a. die mit den zentralen Verfahren einhergehenden Verfügbarkeitsprobleme umgangen werden. Allerdings sind nun die lokalen Validierungen einer TA derart zu koordinieren, daß die globale Serialisierbarkeit gewährleistet ist. Dies wird am einfachsten dadurch erreicht, daß die Transaktionen in jedem Knoten in der gleichen Reihenfolge validiert werden, was z. B. mit einem Token-Ring-Protokoll oder unter Nutzung eines Broadcast-Mediums geschehen kann [38]. Während mit dem Token-Ring-Verfahren die Validierungen systemweit sequentialisiert werden, werden bei der *Broadcast-Validierung* [36] alle Validierungen einer TA durch eine Broadcast-Nachricht gleichzeitig gestartet und parallel von allen Rechnern bearbeitet (kürzere Antwortzeiten). Allerdings ergibt sich dabei ein relativ hoher Validierungs- und Kommunikations-Overhead, da jede TA an jedem Rechner validiert wird. Dieser Overhead läßt sich jedoch durch ein PCA-artiges Validierungsschema i. a. deutlich senken, mit dem eine TA nur noch an den Knoten zu validieren hat, welche die PCA für von der TA referenzierte Objekte halten [36]. Das modifizierte Verfahren kann zur Reduzierung von Rücksetzungen auch mit dem Primary-Copy-Sperrverfahren kombiniert werden, wobei dann eine TA entweder pessimistisch (lange TA, bereits gescheiterte TA) oder optimistisch synchronisiert wird.

3. Behandlung von Pufferinvalidierungen

Zur Reduzierung von physischen E/A-Vorgängen (Plattenzugriffen) verwalten DBVS einen Datenbank- oder Systempuffer im Hauptspeicher, in dem referenzierte DB-Seiten für weitere Zugriffe gepuffert werden [13]. Da bei DB-Sharing jeder Rechner einen solchen Systempuffer führt und jeder Knoten direkten Zugriff auf die gesamte Datenbank hat, kann eine bestimmte DB-Seite gleichzeitig in mehreren Puffern vorliegen (dynamische Replikation der Daten). Die Folge davon ist das sogenannte *Veralterungs-* oder *Pufferinvalidierungsproblem*, da die Änderung einer DB-Seite in einem der Rechner die Kopien der Seite in anderen Systempuffern sowie auf Platte invalidiert. Die Behandlung dieses Invalidierungsproblems erfordert daher die Erkennung (bzw. Vermeidung) veralteter Seitenkopien sowie die Bereitstellung der aktuellen Objektversionen beim Objektzugriff.[4]

Die Behandlung von Pufferinvalidierungen hängt bei DB-Sharing vor allem von der Wahl des Synchronisationsprotokolls, von der Größe des Synchronisationsgranulates sowie von der Ausschreibstrategie für geänderte DB-Seiten ab:

- *Einfluß des Synchronisationsverfahrens*
Bei Sperrverfahren ist stets zu gewährleisten, daß nur aktuelle DB-Objekte referenziert werden; da vor jedem Objektzugriff eine Sperre zu erwerben ist, kann dies durch eine geeignete Erweiterung des Sperrprotokolls erreicht werden. Bei optimistischen Protokollen dagegen erfolgen die Objektreferenzen unsynchronisiert, so daß Zugriffe auf veraltete Objekte oft erst während der Validierung am TA-Ende erkannt werden. Zur Begrenzung von Rücksetzungen ist hier auch dafür zu sorgen, daß geänderte Objektversionen in allen Knoten des DB-Sharing-Systems schnell verfügbar gemacht werden.

- *Wahl des Synchronisationsgranulats*
Bei einer Synchronisation auf Satzebene ist es möglich, daß zwei verschiedene Sätze derselben DB-Seite zur gleichen Zeit in den Systempuffern verschiedener Knoten geändert werden. Dies hat zur Folge, daß in beiden Knoten nur eine teilweise aktuelle Seitenversion vorliegt und die Seitenkopie auf Platte (zunächst) keine der Änderungen enthält. Um eine aktuelle Version der DB-Seite zu erhalten, ist daher spätestens vor dem Ausschreiben auf Platte ein Mischen der Änderungen erforderlich, was nicht nur zusätzliche Kommunikationsvorgänge verursacht, sondern bei Änderungen der Seitenstruktur (Einfügungen, Löschvorgängen, Splitten von Index-Seiten u.ä.) nicht immer möglich ist. Zur Vermeidung dieser Schwierigkeiten müssen daher i. a. zumindest Schreibvorgänge zwischen den Rechnern auf Seitenebene synchronisiert werden (Änderungszugriffe innerhalb eines Knotens sowie Lesevorgänge können dagegen auf Satzebene synchronisiert werden). Wir gehen hier von einer Synchronisation auf Seitenebene für alle DB-Zugriffe aus, da so die Behandlung von Pufferinvalidierungen vereinfacht wird und ein verringerter Verwaltungsaufwand, auch zur Synchronisation (weniger Sperranforderungen), entsteht. Andererseits ist natürlich bei einer Synchronisation auf Seitenebene die Konfliktwahrscheinlichkeit höher als bei satzorientierten Verfahren.

- *Wahl der Ausschreibstrategie für geänderte DB-Seiten*
Hier ist generell zwischen einer sogenannten FORCE- und einer NOFORCE-Strategie zu unterscheiden [21]. Bei FORCE werden am Ende einer Update-TA sämtliche geänderte Seiten vom Systempuffer in die materialisierte Datenbank auf Platte ausgeschrieben. Dies hat für DB-Sharing (bei Synchronisation auf Seitenebene) den Vorteil, daß die aktuelle Version einer DB-Seite, in der alle Änderungen erfolgreich beendeter TA reflektiert sind, stets durch Einlesen von Platte erreicht werden kann. Allerdings ist der FORCE-Ansatz für Hochleistungssysteme nicht tolerierbar, da die Schreibvorgänge einen hohen E/A-Overhead verursachen und die Antwortzeiten der Änderungs-TA erheblich verlängern (erhöhte Wahrscheinlichkeit von Synchronisa-

[4] Ein analoges Invalidierungsproblem gibt es in eng gekoppelten Multiprozessoren für die Prozessor-Caches [53], das dort oft mit spezieller Hardware-Unterstützung gelöst wird. Auch in Workstation-Server-Umgebungen bzw. verteilten Dateisystemen ergibt sich ein ähnliches Veralterungsproblem wie bei DB-Sharing, wenn Daten (z. B. Seiten) zur Einsparung von Kommunikationsvorgängen mit den DB- bzw. Datei-Servern in den Workstations gepuffert werden können [31, 43]

tionskonflikten). Es ist somit eine NOFORCE-Strategie zu unterstützen, wenngleich dadurch die Behandlung von Pufferinvalidierungen und Rechnerausfällen (Kap. 5) komplizierter wird.

Da bei NOFORCE die auf Platte vorliegenden DB-Seiten möglicherweise veraltet sind, ist in Hauptspeicher-Datenstrukturen (repliziert in jedem Knoten oder in Erweiterung der globalen Sperrtabelle) zu vermerken, wo für geänderte Seiten eine aktuelle Kopie erhältlich ist. Anstelle eines Einlesevorganges von Platte muß dann zur Bereitstellung einer geänderten Seite ggf. eine explizite *Seitenanforderung* an einen Rechner geschickt werden, der im Besitz einer aktuellen Kopie ist. Zur Übergabe der Seite kommen mehrere Möglichkeiten in Frage; bei loser Kopplung ist eine direkte Übertragung der Seite über die Kommunikationsleitungen einem Seitenaustausch über Platte vorzuziehen. Bei einer schnellen Rechnerkopplung ist das Anfordern einer Seite in einem anderen Rechner so i.a. mindestens um einen Faktor 10 schneller als ein Einlesevorgang von Platte.

Abbildung 3 zeigt eine Übersicht über die wichtigsten Techniken zur Behandlung von Pufferinvalidierungen bei DB-Sharing; insbesondere für Sperrverfahren sind unterschiedliche Strategien möglich. Die Verfahren, die Pufferinvalidierungen entweder vermeiden oder erkennen, werden im folgenden näher beschrieben bevor eine abschließende Bewertung der Protokolle vorgenommen wird.

Broadcast-Invalidierung

Bei diesem Ansatz wird am Ende einer Änderungs-TA eine Broadcast-Nachricht an alle Knoten geschickt mit der Angabe, welche Seiten von der Transaktion geändert wurden. Diese Nachricht erlaubt es, veraltete Seiten in den Systempuffern sofort zu erkennen und zu entfernen.

Bei Sperrverfahren ist zur Vermeidung von Zugriffen auf invalidierte Seiten die Freigabe der Schreibsperren zu verzögern, bis alle Rechner den Empfang der Broadcast-Nachricht quittiert haben. Da die Änderungs-TA synchron auf das Eintreffen dieser Quittierungen warten müssen, bezeichnen wir diesen Ansatz auch als *synchrone Broadcast-Invalidierung*. Alle existierende DB-Sharing-Systeme verwenden (soweit bekannt) diese einfache Vorgehensweise - in Kombination mit FORCE - zur Behandlung von Pufferinvalidierungen. Trotzdem ist die synchrone Broadcast-Invalidierung nicht empfehlenswert, da neben den Nachteilen einer FORCE-Strategie ein zusätzlicher Kommunikations-Overhead (Broadcast-Meldungen, Quittierungen) eingeführt wird, der mit wachsender Rechneranzahl steigt. Außerdem erhöhen sich die Antwortzeiten für Update-TA (Sperrdauer) durch das synchrone Warten auf die Quittierungen.

Bei optimistischen Verfahren kann die Invalidierungsnachricht nach Beendigung der Update-TA verschickt werden (*asynchrone Broadcast-Invalidierung*) da der Zugriff auf veraltete Objekte hier ohnehin nicht

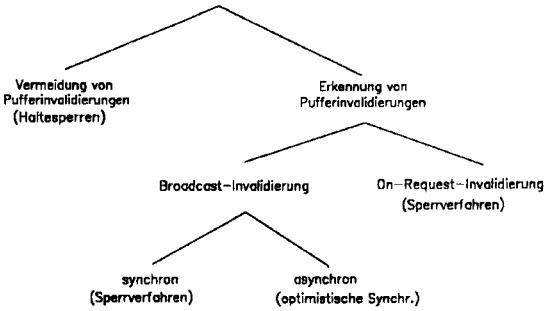

Abb. 3. Techniken zur Behandlung von Pufferinvalidierungen bei DB-Sharing

vermeidbar ist und bei der Validierung entdeckt wird. Die Broadcast-Nachrichten führen so nicht zur Erhöhung der Antwortzeiten; zudem eignen sich die asynchronen Meldungen besser zur gebündelten Übertragung (Verringerung des Kommunikations-Overheads). Außer zum Entfernen veralteter Seiten aus den Systempuffern kann eine Invalidierungsnachricht bei optimistischer Synchronisation auch dazu benutzt werden, laufende TA sofort zurückzusetzen, wenn sie bereits auf ein invalidiertes Objekt zugegriffen haben [36]. Da diese TA zum Scheitern verurteilt sind, kann so eine unnötige Weiterbearbeitung vermieden werden.

Um die Zahl der Rücksetzungen möglichst gering zu halten, ist bei NOFORCE ein Austausch geänderter Seiten zwischen den Rechnern vorzunehmen. Hierzu führt jeder Rechner eine replizierte Datenstruktur, in der vermerkt wird, welcher Rechner eine Seite zuletzt geändert hat; diese Angaben können den Invalidierungsnachrichten entnommen werden. Die Tabelle zeigt nun an, an welchen Rechner für geänderte Seiten eine Seitenanforderung zu richten ist, um möglichst die aktuelle Seitenversion zu erhalten. Das Ausschreiben geänderter Seiten auf Platte wird asynchron mitgeteilt, um die Zahl der Kontrolleinträge und ‚erfolglosen' Seitenanforderungen (die angeforderte Seite wurde bereits aus dem Systempuffer verdrängt) zu reduzieren [36, 38].

Vermeidung von Pufferinvalidierungen

Bei Sperrverfahren können nur solche Seiten invalidiert werden, für die keine Sperre gehalten wird, da eine Sperre verhindert, daß eine Transaktion in einem anderen Rechner eine Schreibsperre erwirbt und die Seite ändert (invalidiert). Pufferinvalidierungen können daher vollkommen vermieden werden, wenn alle Seiten vor Freigabe der letzten lokal gehaltenen Sperre (bei TA-Ende) aus dem Systempuffer entfernt werden (buffer purge). Dieser einfache Vermeidungsansatz ist allerdings von keiner praktischen Bedeutung, da er nicht nur eine FORCE-Ausschreibstrategie impliziert (hoher Schreibaufwand), sondern auch vermehrte Lesevorgänge von Platte, da Lokalität im Referenzverhalten nicht mehr über TA-Grenzen hinweg genutzt werden kann.

Diese Nachteile können mit einer anderen Vermeidungsstrategie umgangen werden, bei der Seiten nach

dem TA-Ende im Puffer verbleiben, jedoch durch spezielle *Haltesperren* vor einer Invalidierung durch einen anderen Rechner geschützt werden. Ein solcher Ansatz wurde zuerst für ein Pass-the-Buck-Protokoll vorgeschlagen [18], kann jedoch auch auf andere Sperrverfahren für DB-Sharing übertragen werden (z. B. mit zentralem Sperrverwalter).

Durch die Verwendung von zwei Typen von Haltesperren (HR und HX) kann nicht nur das Veralterungsproblem bei NOFORCE gelöst, sondern auch ein Sole-Interest-Konzept sowie eine Leseoptimierung auf Seitenebene realisiert werden, mit denen globale Sperranforderungen eingespart werden können. Obwohl die Einzelheiten dazu [18] den Rahmen dieses Berichtes sprengen würden, sollen zur Verdeutlichung der Grundideen einige Kernpunkte des Verfahrens angegeben werden:

- Für jede DB-Seite im Systempuffer, für die keine reguläre TA-Sperre vorliegt, hält der Rechner eine Haltesperre. Für eine geänderte Seite wird dabei eine HX-Sperre geführt, die Sperren sowie Haltesperren (und damit Seitenkopien) in anderen Rechnern ausschließt. Eine HX-Sperre kennzeichnet daher eine ‚sole interest'-Situation, die eine lokale Sperrvergabe von Lese- und Schreibsperren erlaubt.
- Für ungeänderte DB-Seiten im Systempuffer wird eine HR-Sperre gehalten, die in mehreren Rechnern zur gleichen Zeit möglich ist und die Existenz einer Schreib- oder HX-Sperre ausschließt. Eine HR-Sperre kommt daher einer Leseautorisierung gleich und ermöglicht die lokale Vergabe von Lesesperren.
- Soll auf eine im lokalen Puffer nicht vorliegende Seite zugegriffen werden, so zeigt die globale Sperrtabelle an, ob die Seite bei einem anderen Rechner anzufordern ist (HX-Sperre). Der notwendige Entzug der Haltesperre kann in diesem Fall mit der Seitenanforderung kombiniert werden. Die Durchführung einer Änderung erfordert den vorherigen Entzug aller Haltesperren (Aufgabe der Leseautorisierungen bzw. Sole-Interest-Zuweisung). Mit Aufgabe einer Haltesperre wird die zugehörige Seite aus dem Puffer entfernt und damit eine Invalidierung vermieden.
- Das Setzen der Haltesperren erfolgt i. a. nach Freigabe der regulären TA-Sperren, die Freigabe entweder durch Aufforderung seitens TA anderer Rechner oder nach Verdrängen der zugehörigen DB-Seite aus dem Systempuffer.

Die Haltesperren erlauben so eine integrierte Lösung für Synchronisation und Behandlung von Pufferinvalidierungen, die v. a. für Sperrverfahren mit Sole-Interest-Konzept von Interesse ist.

On-Request-Invalidierung

Dieser Ansatz erlaubt ebenfalls eine integrierte Lösung des Veralterungsproblems im Rahmen des Synchronisationsprotokolls (für Sperrverfahren). Im Gegensatz zur Vermeidungsstrategie können hier jedoch DB-Seiten im Systempuffer invalidiert werden; die Gültigkeit einer Seite wird dann vor dem Zugriff, bei Bearbeitung der zugehörigen Sperranforderung, überprüft (on-request invalidation [22]). Veraltete Seiten werden dabei mittels geeigneter Erweiterungen der globalen Sperrinformation erkannt, z. B. beim PCA-Sperrverwalter oder ZLM. Für NOFORCE wird in der globalen Sperrtabelle zusätzlich vermerkt, bei welchem Rechner eine geänderte Seite angefordert werden kann (i. a. bei dem Knoten, an dem die letzte Änderung der Seite vorgenommen wurde).

Zur Realisierung einer On-Request-Invalidierung kommen verschiedene Möglichkeiten in Betracht [12, 38], die sich vor allem darin unterscheiden, welche Angaben zur Erkennung veralteter Seitenkopien verwendet werden. Wir wollen hier nur kurz die einfachste Strategie unter Verwendung von *Zeitstempeln* oder Versionsnummern skizzieren. Hier wird in jeder DB-Seite ein Zeitstempel (Zähler) geführt, der bei jeder Änderung erhöht wird. Bei Änderung der Seite wird mit Freigabe der Schreibsperre der aktuelle Wert des Zeitstempels dem globalen Sperrverwalter mitgeteilt, der somit stets den Zeitstempel der aktuellen Seitenversion in der globalen Sperrtabelle hält. Bei einer Sperranforderung wird nun zuerst überprüft, ob im lokalen Systempuffer bereits eine Kopie der zu referenzierenden Seite vorliegt und, wenn ja, mit welchem Zeitstempel. Durch Vergleich der Zeitstempel kann der globale Sperrverwalter dann bei der Bearbeitung einer Sperranforderung die Gültigkeit einer Seitenkopie feststellen. Wird eine Pufferinvalidierung entdeckt, so teilt dies der globale Sperrverwalter bei Gewährung der Sperre mit, ebenso (bei NOFORCE) bei welchem Rechner ggf. eine aktuelle Seitenversion angefordert werden kann.

Ein Hauptvorteil einer On-Request-Invalidierung ist, daß im Gegensatz zur Broadcast-Invalidierung Pufferinvalidierungen ohne jegliche Zusatznachrichten und ohne Antwortzeiterhöhung für Änderungs-TA erkannt werden. Dies wird möglich durch die zusätzlichen Angaben in der globalen Sperrtabelle sowie deren Nutzung bzw. Anpassung im Rahmen von Sperranforderungen bzw. bei der Freigabe von Schreibsperren. Andererseits werden nun invalidierte Seiten später als bei der Broadcast-Invalidierung entdeckt und aus dem Puffer entfernt, wodurch sich i. a. etwas verringerte Trefferraten im Puffer ergeben. Quantitative Leistungsuntersuchungen [12, 22, 38] zeigten jedoch, daß dies im Vergleich zu den Vorteilen einer On-Request-Invalidierung von untergeordneter Bedeutung ist.

Obwohl die Erkennung von Pufferinvalidierungen bei einer On-Request-Invalidierung keine eigenen Nachrichten verursacht, werden für NOFORCE wie auch bei den zuvor diskutierten Strategien neben den Synchronisationsnachrichten i. a. zusätzliche Kommunikationsvorgänge zur Anforderung geänderter Seiten erforderlich. Für das *Primary-Copy-Sperrverfahren* kann dieser Austausch geänderter Seiten ebenfalls mit regulären Synchronisationsnachrichten gekoppelt werden, so daß hier die *Behandlung von Pufferinvalidierungen* (auch

bei NOFORCE) *vollkommen ohne zusätzliche Nachrichten* erfolgen kann. Dies wird dadurch möglich, daß mit Freigabe einer Schreibsperre die geänderte Seite selbst auch zum PCA-Rechner geschickt wird (wenn die Änderung in einem anderen Rechner vorgenommen wurde). Damit besitzt der PCA-Rechner stets die aktuellste Kopie für Seiten seiner Partition, oder sie kann von Platte eingelesen werden, wenn der PCA-Rechner die Seite bereits ausgeschrieben hat. Wenn der PCA-Rechner nun bei einer Sperrbearbeitung eine Pufferinvalidierung entdeckt (oder wenn die Seite noch nicht im Systempuffer des anfordernden Rechners vorliegt), kann er eine aktuelle Kopie der Seite zusammen mit der Sperrgewährung zustellen bzw. mitteilen, daß die Seite von Platte einzulesen ist. Es entfallen also die zusätzlichen Antwortzeitverzögerungen sowie der Kommunikationsaufwand für eigene Seitenanforderungen. Ein weiterer Vorteil ist, daß Pufferinvalidierungen nur noch für solche Seiten möglich sind, für die ein anderer Rechner die PCA hält. Eine ausführlichere Beschreibung des Verfahrens mit Beispiel findet sich in [38].

Zusammenfassende Wertung

Zur TA-Verarbeitung in lose gekoppelten DB-Sharing-Systemen werden Kommunikationsvorgänge zur globalen Synchronisation sowie zur Behandlung von Pufferinvalidierungen (einschließlich Austausch geänderter Seiten zwischen den Rechnern) benötigt. Die in existierenden DB-Sharing-Systemen vorherrschende Broadcast-Invalidierung in Verbindung mit einer FORCE-Ausschreibstrategie ist für Hochleistungssysteme nicht tolerierbar. Wie gezeigt wurde, lassen sich auch bei NOFORCE (geringerer E/A-Overhead, kürzere Antwortzeiten für Update-TA) zusätzliche Nachrichten aufgrund des Veralterungsproblems durch eine integrierte Behandlung im Rahmen des Synchronisationsprotokolls weitgehend vermeiden.

Die effizienteste Behandlung von Pufferinvalidierungen wird möglich mit Sperrverfahren (und insbesondere dem Primary-Copy-Sperrverfahren), da durch eine Erweiterung der Sperrbehandlung Pufferinvalidierungen entweder vermieden oder ohne zusätzliche Nachrichten entdeckt werden können. Die Vermeidungsstrategie (mit Haltesperren) scheint dabei z.B. für Pass-the-Buck-Verfahren und zentrale Sperrprotokolle (ZLM) empfehlenswert zu sein, da damit zugleich ein Sole-Interest-Konzept sowie eine Leseoptimierung realisiert werden kann. Für das Primary-Copy-Sperrverfahren, das kein Sole-Interest-Konzept anwendet, ist dagegen eine On-Request-Invalidierung angebracht (sowie eine Leseoptimierung). Ein solches Primary-Copy-Verfahren zeigte in [38] von mehreren Protokollen die besten Simulationsergebnisse, wofür neben dem Nutzen der PCA-Verteilung sowie der Leseoptimierung zur Reduzierung von Synchronisationsnachrichten auch die effiziente Lösung des Veralterungsproblems ohne jegliche Zusatznachrichten (s.o.) ausschlaggebend war.

Die Simulationen in [38] ergaben auch, daß optimistische Protokolle v.a. mit wachsender Rechneranzahl wegen der geringeren Abhängigkeit von der Erreichbarkeit rechnerspezifischer Lokalität meist weniger Kommunikationsvorgänge zur Synchronisation als Sperrverfahren verursachen. Andererseits wird das Veralterungsproblem weniger effizient gelöst (asynchrone Broadcast-Invalidierung), was vor allem mit wachsendem Anteil an Änderungs-TA negativ zu Buche schlägt. Noch nachteiliger sind jedoch die vielfachen Rücksetzungen, die trotz verschiedener Optimierungen bei Lasten mit höherer Änderungshäufigkeit oder langen TA nicht auf einem akzeptablen Niveau gehalten werden konnten. Eine Kombination mit Sperrverfahren ist zwar generell realisierbar, jedoch ergeben sich komplexe Protokolle, die nicht notwendigerweise Leistungsvorteile verschaffen [38].

Unter Leistungsgesichtspunkten ist das durch die Datenreplikation in den Systempuffern eingeführte Veralterungsproblem kein signifikanter Nachteil für DB-Sharing. Denn durch diese Form der Datenreplikation kann - im Gegensatz zu DB-Distribution - ein Objekt parallel in verschiedenen Knoten gelesen werden. Simulationen [18, 22, 38] zeigten zudem, daß Pufferinvalidierungen bei geeigneter Lastverteilung und der typischerweise relativ kleinen Rechneranzahl (< 20) i.a. vergleichsweise selten vorkommen; auch läßt sich das Veralterungsproblem ohne eigene Nachrichten lösen. Darüber hinaus können bei NOFORCE geänderte Seiten auch für den ersten Zugriff (also ohne vorherige Pufferinvalidierung) direkt von einem anderen Rechner erhalten werden, während im Ein-Rechner-Fall hierzu ein (wesentlich langsamerer) Lesevorgang von Platte notwendig ist. Dies trug in Simulationen [38] zu einem besseren E/A-Verhalten bei DB-Sharing als in zentralisierten DBS bei, wodurch sich Antwortzeiterhöhungen aufgrund von Kommunikationsvorgängen zumindest teilweise kompensieren ließen.

4. Lastkontrolle bei DB-Sharing

Um das Leistungspotential einer DB-Sharing-Architektur zur Einhaltung vorgegebener Antwortzeit- und Durchsatzanforderungen nutzen zu können, sollte eine Lastkontrolle und -balancierung auf wenigstens zwei Ebenen stattfinden [38]:

- Primäre Aufgabe der *globalen Lastkontrolle* ist die Lastverteilung (z.B. über ein Front-End-System), d.h. die Zuordnung ankommender TA-Aufträge zu Verarbeitungsrechnern. Die Routing-Strategie sollte dabei eine möglichst hohe rechnerspezifische Lokalität im Referenzverhalten anstreben sowie die Überlastung einzelner Knoten verhindern (Lastbalancierung). Eine denkbare Option ist auch, die aktuelle Anzahl der benutzten Rechner je nach Lastaufkommen zu variieren (Abstellen bzw. Hinzunehmen einzelner Rechner) [47].

• Auf Ebene der Verarbeitungsrechner ist eine *lokale Lastkontrolle* vorzusehen, die eine möglichst effiziente Verarbeitung der zugeordneten TA gewährleisten soll und die statistische Meßgrößen (CPU-Auslastung, Häufigkeit von Paging- und E/A-Vorgängen oder Sperrkonflikten u.ä) sammelt, um potentielle Leistungsengpässe erkennen zu können bzw. für globale Routing-Entscheidungen benötigte Angaben zum Referenzverhalten zu erhalten.
• Gemeinsame Aufgaben von globaler und lokaler Lastkontrolle, die innerhalb einer Feedback-Schleife zusammenwirken [40], sind darüber hinaus die Überwachung vorgegebener Leistungsanforderungen (z.B. Antwortzeitrestriktionen) sowie das Ergreifen geeigneter Korrekturmaßnahmen, wenn eine Verfehlung von Sollgrößen zu befürchten bzw. bereits eingetreten ist. Dazu zählt auch eine Anpassung der Routing-Strategie nach einem Rechnerausfall bzw. bei hinzugekommenem Rechner oder signifikant geändertem Lastprofil (Referenzverhalten).

Bisherige Simulationen von DB-Sharing-Systemen [38, 55] belegen die Wichtigkeit eines sogenannten *Affinitäts-basierten TA-Routing*, bei dem TA-Typen mit ähnlichem Referenzverhalten (d.h. mit Affinität bezüglich denselben DB-Bereichen) möglichst demselben Rechner zugeordnet werden (Unterstützung von rechnerspezifischer Lokalität). Eine solche Lastverteilung wird durch das Vorherrschen vorgeplanter Transaktionen möglich, für die i.a. ausreichende Angaben bezüglich des Referenzverhaltens bekannt sind bzw. gesammelt werden können. Die Affinitäts-basierte Lastverteilung erlaubt nicht nur die Einsparung globaler Synchronisationsnachrichten (v.a. für Sperrverfahren), sondern bewirkt auch verbesserte Trefferraten in den Systempuffern sowie weniger Pufferinvalidierungen und damit weniger physische Lesevorgänge bzw. Seitenanforderungen. Darüber hinaus kann damit die Häufigkeit von Sperrkonflikten (und Deadlocks) zwischen TA verschiedener Rechner reduziert werden, die aufgrund von Kommunikationsverzögerungen i.a. langsamer als Konflikte zwischen lokalen TA aufgelöst werden können. Im Vergleich zu einer wahlfreien Lastverteilung wurden in [38] so signifikante Antwortzeit- und Durchsatzverbesserungen bei Affinitäts-basiertem TA-Routing beobachtet, wobei die Verbesserungen mit wachsender Rechneranzahl zunahmen.

Allerdings sind auch einer Affinitäts-basierten Lastverteilung Grenzen gesetzt, v.a. hinsichtlich modularer Wachstumsfähigkeit, bei

• dominierenden DB-Bereichen (die von der Mehrzahl der TA-Typen referenziert werden),
• dominierenden TA-Typen (die nicht vollständig in einem Rechner abgearbeitet werden können, ohne diesen zu überlasten),
• geringer Lokalität innerhalb eines TA-Typs (wenn TA eines TA-Typs nahezu alle wesentlichen DB-Bereiche referenzieren).

Diese Probleme lassen sich bei DB-Sharing mit einem geeigneten Synchronisationsverfahren (Leseoptimierung) zumindest teilweise entschärfen, solange Lesezugriffe vorherrschen. Ansonsten muß z.B. versucht werden, dominierende TA-Typen durch Berücksichtigung von Eingabeparametern in Sub-TA-Typen zu zerlegen (bei der ‚Kontenbuchung' etwa über die Kontonummer). Eine andere Möglichkeit ist ein angepaßter Entwurf der TA-Typen, bei dem beispielsweise ein allgemeiner TA-Typ durch mehrere zugeschnittene TA-Typen ersetzt wird.

In Anlehnung an [9] können Routing-Strategien ferner in *dynamische, statische* und *semi-dynamische* (oder *adaptive*) *Verfahren* eingeteilt werden. Vollkommen dynamische Verfahren versuchen, für jede ankommende TA unter Berücksichtigung des aktuellen Systemzustandes den Rechner zu bestimmen, der die effizienteste Bearbeitung der TA erlaubt. So wurden z.B. in [54] Verfahren vorgeschlagen, bei denen pro TA für jeden Rechner eine Antwortzeitschätzung berechnet und die TA dem Rechner zugeordnet wird, für den die geringste Antwortzeit prognostiziert wurde. Solche Verfahren sind viel zu aufwendig und für die zu bewältigenden TA-Raten nicht praktikabel. Dieser Nachteil soll bei den statischen bzw. semi-dynamischen Routing-Verfahren vermieden werden, bei denen die TA-Verteilung über eine *Routing-Tabelle* vorgenommen wird, die eine Zuordnung der TA-Typen zu den Rechnern festlegt. Im Gegensatz zur statischen Strategie, die für DB-Sharing nicht sinnvoll ist, wird bei der semi-dynamischen Lastverteilung eine Anpassung der Routing-Tabelle in den oben erwähnten Fällen (geändertes Referenzverhalten, Rechnerausfall etc.) vorgesehen.

Eine tabellengesteuerte Lastverteilung wird dadurch gerechtfertigt, daß in realen TA-Systemen vorwiegend kurze und vorgeplante TA-Typen ausgeführt werden, deren Referenzverhalten zumindest grob bekannt ist, und daß die Lastzusammensetzung oft über Stunden hinweg keinen signifikanten Änderungen unterworfen ist [40]. Zur Bestimmung der Routing-Tabelle, die eine Affinitäts-basierte Lastverteilung sowie eine möglichst gleichmäßige Rechnerauslastung gewährleisten soll, müssen für die wichtigsten TA-Typen hinreichend genaue Angaben hinsichtlich des Referenzverhaltens, der Ankunftsraten sowie des Instruktionsbedarfs bekannt sein. Beim Primary-Copy-Sperrverfahren sollte zudem mit der Routing-Tabelle auch die PCA-Verteilung bestimmt werden, um zu einer abgestimmten Lösung mit möglichst geringem Kommunikationsbedarf zur Synchronisation zu kommen. Dazu muß zunächst eine ausreichende Anzahl von DB-Fragmenten festgelegt werden, die dann mit geeigneten Algorithmen zu Partitionen zusammengefaßt und den Rechnern zugeordnet werden.

Zur Berechnung einer Routing-Tabelle (und PCA-Verteilung) aus den genannten Angaben wurden verschiedene ‚optimale' und approximative Verfahren vorgeschlagen. Eine Formalisierung des Optimierungs-

problems sowie die Bestimmung der besten Lösung mittels numerischer Hilfsmittel wird z. B. in [11] verfolgt; dieser Ansatz verursacht jedoch einen enormen Rechenaufwand, so daß er für eine adaptive Anpassung der Tabellen im laufenden Betrieb ausscheidet. Zudem ist eine wesentliche Einschränkung dabei, daß die TA-Typen immer vollkommen einem Rechner zugeordnet werden müssen, was bei dominierenden TA-Typen nicht akzeptabel ist. Einen geeigneteren Ansatz stellen z. B. die effizient ausführbaren Heuristiken in [34] zur Bestimmung der Routing- und PCA-Tabellen dar, die auch ein sogenanntes *probabilistisches TA-Routing* erlauben, bei dem ein TA-Typ mehreren Rechnern zugeordnet werden kann.

Allerdings ist klar, daß mit einer Routing-Tabelle allein kein balancierter Systemzustand erreicht werden kann, weil die bei deren Berechnung verwendeten Angaben dafür zu ungenau oder zu unvollständig sind. So ist z. B. die tatsächliche Auslastung der Rechner von vielen Faktoren abhängig (z. B. E/A- oder Sperrkonfliktverhalten), die mit einer Routing-Tabelle nicht berücksichtigt werden können. Daher müssen zumindest in einigen Ausnahmesituationen (z. B. Überlastung eines Rechners) von der Routing-Tabelle abweichende Verteilentscheidungen getroffen bzw. geeignete Änderungen der Tabelle vorgenommen werden (Umdirigieren eines TA-Typs vom überlasteten Knoten auf einen wenig ausgelasteten Rechner u. ä). Die Tauglichkeit solcher Maßnahmen kann i. a. jedoch nur im praktischen Betrieb bzw. im Rahmen einer Prototyp-Implementierung unter Verwendung realer Lasten nachgewiesen werden.

Die Forderung nach einzuhaltenden Antwortzeit- und Durchsatzvorgaben verlangt auch die Bereitstellung geeigneter Lastkontrollmaßnahmen innerhalb der Verarbeitungsrechner. Zentrale Betriebsparameter innerhalb des DB/DC-Systems, deren automatische Anpassung an verschiedene Lastsituationen wünschenswert ist, sind u. a. die Parallelität (Anzahl parallel bearbeiteter TA-Aufträge), die Systempuffergröße und die Anzahl von Server-Prozessen zur Bearbeitung der DB-Operationen. Z. B. kann es zur Begrenzung von Synchronisationskonflikten sinnvoll sein, jeweils nur eine bestimmte Anzahl gleichzeitiger Aktivierungen von TA-Typen mit ‚ähnlichem' Referenzverhalten (und damit erhöhter Konfliktgefahr) zuzulassen. Andere Maßnahmen, wie dynamische Änderung von Systempuffergröße oder Prioritäten, erfordern eine enge Zusammenarbeit mit den Resource-Managern des Betriebssystems (CPU-Scheduling, Speicherverwaltung etc.). In diesem Bereich sind noch viele Probleme ungelöst, die offenbar auch für zentralisierte DBS von Bedeutung sind.

5. Recovery-Aspekte

Das Rücksetzen einzelner TA im laufenden Betrieb wirft keine neuen Probleme im Vergleich zu zentralisierten DBS auf. Dies geschieht innerhalb der jeweiligen Verarbeitungsrechner ggf. unter Verwendung der *lokalen Log-Datei*, in der alle Änderungen eines Rechners sowie die Ein- und Ausgabenachrichten der dort ausgeführten TA protokolliert werden. Durch asynchrones Mischen der lokalen Log-Dateien kann eine *globale Log-Datei* erstellt werden [46], welche sämtliche DB-Änderungen des Systems in chronologischer Reihenfolge enthält. Zur Begrenzung des E/A- und des Log-Umfangs ist dabei ein Logging auf Seitenebene nicht tolerierbar, vielmehr ist ein sogenanntes Eintrags-Logging [21] vorzunehmen, bei dem nur die (After-Images der) geänderten Teile einer DB-Seite protokolliert werden.

Die Recovery nach einem *Rechnerausfall* muß, um eine möglichst störungsfreie Fortsetzung der TA-Verarbeitung zu erlauben (Freigabe gehaltener Sperren u. ä.), von den überlebenden Rechnern vorgenommen werden. Dabei sind, wie im zentralen Fall, alle unvollendeten TA des ausgefallenen Rechners zurückzusetzen sowie ggf. die von ihnen gehaltenen Sperren freizugeben. Bei NOFORCE (s. o.) sind weiterhin alle in dem ausgefallenen Rechner vorgenommenen und durch den Rechnerausfall verlorengegangenen Änderungen erfolgreich beendeter TA aus den Log-Daten zu rekonstruieren und in die aktuelle Datenbank einzubringen (REDO-Recovery). Weitere Aktionen im Rahmen der Crash-Recovery betreffen die Anpassung der Routing-Strategie sowie eine Rekonstruktion verlorengegangener Datenstrukturen, die zur korrekten Fortsetzung der Synchronisation bzw. Behandlung des Veralterungsproblems benötigt werden. Gescheiterte TA können unter Verwendung der protokollierten Eingabenachrichten auf den verbleibenden Rechnern wiederholt werden, um den Rechnerausfall für den Benutzer möglichst transparent zu halten. Voraussetzung dazu ist, daß die Verbindungen zwischen den Terminals und dem DB-Sharing-System trotz des Rechnerausfalls aufrechterhalten bleiben.

Die Realisierung der angesprochenen Aktionen, die von den verwendeten Protokollen zur Synchronisation und Behandlung des Veralterungsproblems abhängt, kann sehr komplex und aufwendig werden. Die bei NOFORCE vorzunehmende REDO-Recovery wird z. B. bei DB-Sharing dadurch kompliziert, daß jedes DB-Objekt in jedem Rechner geändert werden kann, so daß auch die zugehörigen After-Images i. a. über mehrere lokale Log-Dateien verstreut sind. Daher ist bei der REDO-Recovery für einen Rechner sicherzustellen, daß keine veralteten After-Images von dessen lokaler Log-Datei angewendet werden. Für die REDO-Recovery sind zur Vermeidung von Interferenzen mit laufenden TA auch möglicherweise Sperren durch die Recovery-Prozesse anzufordern, wobei etwaige Sperrkonflikte oder Deadlocks schnell zu beseitigen sind.

Für Sperrverfahren können nach Ausfall einer Sperrtabelle Angaben über gewährte oder wartende

Sperranforderungen relativ leicht aus lokalen Tabellen der überlebenden Rechner rekonstruiert werden [38], nicht jedoch die Angaben zur integrierten Behandlung des Veralterungsproblems. Werden diese Informationen aus Aufwandsgründen nicht ausfallsicher gespeichert, kann z. B. bei einem On-Request-Invalidierungsverfahren die Aktualität von Seiten in den Systempuffern oder auf Platte nicht mehr festgestellt werden, und es ist unbekannt, wo eine aktuelle Kopie der Seite erhältlich ist. Bei einem zentralen Sperrverfahren müssen daher nach Ausfall des ZLM sämtliche Seiten, für die keine lokale Sperre (oder Autorisierung zur lokalen Sperrvergabe) vorliegt und damit die Aktualität feststeht, als invalidiert angesehen und aus den Puffern entfernt werden. Ebenso ist eine REDO-Recovery für die gesamte Datenbank notwendig, um die aktuellen Seitenversionen aus den (globalen) Log-Daten zu rekonstruieren. Die damit verbundenen Verzögerungen dürften jedoch i.a. aus Verfügbarkeitsgründen nicht tolerierbar sein. Beim Primary-Copy-Sperrverfahren kann diese REDO-Recovery auf die Partition des ausgefallenen Rechners beschränkt werden [38]; eine weitere Eingrenzung der zu rekonstruierenden (und damit während der Recovery zu blockierenden) Seitenmenge kann durch Checkpoint-Mechanismen [21] erreicht werden.

Die Behebung von Plattenfehlern kann mit Spiegelplatten oder ebenfalls mit der globalen Log-Datei (sowie Archiv-Kopien) erfolgen. Eine schnelle *Katastrophen-Recovery* (Bombenanschlag, Erdbeben etc.) verlangt jedoch eine ortsverteilte Verwaltung der Datenbestände und ist daher bei DB-Sharing schwieriger als für DB-Distribution zu erreichen. Eine kostspielige Möglichkeit wäre die Installierung eines passiven Stand-By-Systems in ausreichend großer Entfernung, dessen einzige Aufgabe im Normalbetrieb darin besteht, die Änderungen des Primärsystems auf einer DB-Kopie nachzufahren, mit der im Fehlerfall die Verarbeitung fortgesetzt wird.[5] Wenn im Katastrophenfall der Verlust einiger Transaktionen toleriert werden kann, genügt dazu eine asynchrone Übertragung der globalen Log-Daten zum Stand-By-System. Anderenfalls müßten Änderungs-TA verzögert werden, bis ihre Änderungen im entfernten System angekommen sind, was i.a. jedoch nicht tolerierbar sein dürfte.

Ein alternativer Ansatz wäre die Verwendung zweier ‚aktiver' DB-Sharing-Zentren, die jeweils eine komplette Kopie der Datenbank führen und die TA-Last im Normalbetrieb untereinander aufteilen. Bei einer asynchronen Übertragung der Änderungen sind hier zusätzliche Synchronisationsprobleme zu lösen, um einer TA jeweils die aktuellsten Änderungen zugänglich zu machen.

[5] Eine derartige Form der Katastrophen-Recovery unterstützt Tandem neuerdings mit der sogenannten Remote Database Facility (RDF) [29]

6. Einsatzformen einer nahen Rechnerkopplung

Bei der bisherigen Diskussion der DB-Sharing-Funktionen wurde eine lose Kopplung der Rechner unterstellt, bei der sämtliche Kommunikationsvorgänge über Nachrichten stattfinden. Diese Entkopplung der Rechner verspricht zwar die besten Verfügbarkeits- und Erweiterbarkeitsmerkmale, andererseits werden damit oft ein hoher Overhead (Prozeßwechsel, aufwendige Kommunikationsprotokolle) sowie signifikante Antwortzeitbeeinträchtigungen hervorgerufen. Bei räumlich benachbarter Anordnung der Rechner kann mit einer nahen Kopplung [19] eine effizientere Kommunikation angestrebt werden, etwa unter Verwendung von Spezialprozessoren oder gemeinsamer Halbleiterspeicher. Idealerweise sollten dabei die Zugriffszeiten auf solche gemeinsamen Systemkomponenten nur wenige Mikrosekunden betragen, so daß ein synchroner Zugriff (ohne Freigabe der CPU) zur Umgehung der Prozeßwechselkosten möglich wird. Wie bei der losen Kopplung sind die beteiligten Rechner autonom, d.h. sie besitzen einen separaten Hauptspeicher sowie eine eigene Betriebssystem- und DBVS-Kopie.

Als Einsatzformen einer nahen Kopplung bei DB-Sharing werden im folgenden die Nutzung einer ‚Lock-Engine' zur Synchronisation sowie die Verwendung gemeinsamer Halbleiterspeicher betrachtet.

Synchronisation mit Spezialprozessoren

Eine *Lock-Engine* [24] ist ein Spezialprozessor, der die Funktion eines zentralen Sperrverwalters (realisiert in Hardware bzw. Mikrocode) erfüllt. Die Lock-Engine soll eine schnellere Sperrbearbeitung als mit einem Software-Protokoll sowie kürzere Zugriffszeiten als mit einem allgemeinen Nachrichtenaustausch ermöglichen.

Eine erste Realisierung einer derartigen Lock-Engine stellt die *Limited Lock Facility* [5, 42] dar, die im Rahmen des auf TA-Verarbeitung zugeschnittenen Betriebssystems TPF (Transaction Processing Facility [44]) verwendet wird. Die Synchronisation (von bis zu acht TPF-Systemen) findet über einfache Sperrtabellen innerhalb der Plattenkontroller statt, in denen nur exklusive Sperren auf Rechnerebene unterschieden werden (für bis zu 512 Objekte pro Steuereinheit). Das Setzen und Freigeben von Sperren erfolgt mittels eigener Kanalbefehle, die wie ein E/A-Vorgang einen Prozeßwechsel verursachen. Die Gewährung einer Sperre wird über einen Interrupt mitgeteilt. Durch die Verwendung ausreichend vieler Steuereinheiten (sowie deren Ersatzeinheiten für den Fehlerfall) und eine entsprechende Datenverteilung auf die zugehörigen Plattenlaufwerke können Engpässe vermieden werden.

Mit einer solchen Methode kann eine Sperre zwar i.a. schneller und mit geringerem CPU-Overhead (für TPF ca. 500–800 Instruktionen [44]) als eine globale Sperranforderung in einem nachrichtenbasierten Protokoll gewährt werden, aber zusätzlich ist in den Verarbeitungsrechnern ein Sperrprotokoll zu realisieren, z. B. zur Verwaltung von Sperren auf TA-Ebene sowie zur

Behandlung von Pufferinvalidierungen, wartenden Sperranforderungen (Vermeidung von ‚starvation') und globalen Deadlocks. Mit einem solchen Verfahren, das die Komplexität des zunächst einfach erscheinenden Ansatzes erheblich vergrößert, kann dann auch eine lokale Synchronisation zur Reduzierung der Zugriffe auf die Lock-Engine unterstützt werden (z. B. über eine Leseoptimierung bzw. ein Sole-Interest-Konzept). In [42] wird ein solches Verfahren diskutiert.

Eine andere Realisierung einer zentralen Lock-Engine bietet das DCS [45], das intern aus acht Spezialprozessoren besteht, die die Sperranforderungen über eine Hash-Abbildung untereinander aufteilen. Jeder dieser Prozessoren führt seine Sperrtabelle doppelt in unabhängigen Speicherbereichen, so daß bei Ausfall eines Knotens die Synchronisation fortgeführt werden kann. Die Sperren werden dabei auf TA-Ebene gehalten, und es wird eine explizite Erkennung von Deadlocks vorgenommen. Da Zugriffe von den Host-Rechnern zur Lock-Engine über Nachrichten erfolgen, scheint dieser Ansatz eher unter Fehlertoleranzgesichtspunkten von Interesse zu sein. Alternative Techniken zur Realisierung fehlertoleranter Lock-Engines dieser Art werden in [24] untersucht.

Wünschenswert wäre natürlich eine Lock-Engine, die eine Sperranforderung in wenigen Mikrosekunden gewähren kann, so daß ein synchrones Warten (ohne Prozeßwechsel) möglich wird [41]. Dies erfordert jedoch nicht nur einen entsprechenden Spezialprozessor, sondern auch Unterstützung in der Architektur der Verarbeitungsrechner in Form einer speziellen Zugriffsschnittstelle. Damit könnten dann die Verzögerungen zur globalen Synchronisation vernachlässigbar klein (< 1 ms pro TA) und unabhängiger von den Lastcharakteristika gehalten werden.

Verwendung gemeinsamer Halbleiterspeicher

Die Nutzungsmöglichkeiten eines gemeinsamen Halbleiterspeichers sind im wesentlichen durch die Speicher-Charakteristika (Flüchtigkeit, Zugriffsgranulat, Zugriffsgeschwindigkeit, Speichergröße, ...) bestimmt. So ergeben sich z. B. für instruktionsadressierbare (flüchtige) Halbleiterspeicher die vielfältigsten Verwendungsmöglichkeiten [33], jedoch auch ähnliche Verfügbarkeits- und Erweiterbarkeitsprobleme wie mit dem gemeinsamen Hauptspeicher in eng gekoppelten Multiprozessoren. Eine größere Entkopplung ist bei seitenadressierbaren (flüchtigen oder nichtflüchtigen) Halbleiterspeichern gegeben, wie sie derzeit im Rahmen von Platten-Caches [30, 48] oder als ‚expanded storage' [2] eingesetzt werden. Bei DB-Sharing können solche Speicher v. a. zur Erstellung einer globalen Log-Datei, zum beschleunigten Austausch geänderter DB-Seiten zwischen Rechnern sowie als globaler DB- bzw. Systempuffer genutzt werden [33]; wir wollen hier nur die letztgenannten Verwendungsformen näher diskutieren.

Ein *globaler Systempuffer* entspricht einer Erweiterung der Speicherhierarchie zwischen den Hauptspeichern der Verarbeitungsrechner (mit den lokalen Systempuffern) und der physischen Datenbank auf Platte. Damit können Lesezugriffe auf Platte eingespart werden, wenn die zu referenzierende DB-Seite im globalen Puffer vorliegt; bei Nichtflüchtigkeit des gemeinsamen Speicherbereiches werden auch Ausschreibvorgänge beschleunigt, da ein Ausschreiben in den globalen Puffer genügt. Daneben kann der globale Systempuffer auch zum Austausch geänderter DB-Seiten verwendet werden.

Eine implizite Realisierung eines globalen Systempuffers, transparent für die Verarbeitungsrechner sowie die DBVS, ergibt sich, wenn jeder Plattenkontroller einen Platten-Cache (disk cache) führt. Ein Platten-Cache entspricht dann einem Teil des globalen Systempuffers, in dem DB-Seiten der vom jeweiligen Kontroller verwalteten DB-Partition gepuffert werden. Die Verwaltung des globalen Puffers erfolgt somit durch die Plattenkontroller (z. B. LRU-artige Seitenersetzung). Nichtflüchtige Platten-Caches können auch bei NOFORCE zum Austausch geänderter Seiten zwischen den Rechnern genutzt werden: Bei einer Seitenanforderung wird die betreffende Seite (in den Cache) ausgeschrieben und danach dem anfordernden Rechner signalisiert, das die Seite (vom Cache) eingelesen werden kann. Das Puffern der Seite im Cache ist für die beteiligten DBVS völlig transparent; aus deren Sicht handelt es sich um einen (schnellen) Seitenaustausch ‚über Platte'.

Der Vorteil des Ansatzes liegt in der einfachen Realisierbarkeit unter Nutzung existierender Hardware. Andererseits sind die Cache-Zugriffe relativ langsam (1–4 ms [48]) und daher asynchron.

Für kürzere Zugriffszeiten ist eine direkte Anbindung der Rechner an den globalen Speicherbereich, unter Umgehung der Kanalschnittstelle, erforderlich. Eine solche Anordnung liegt beim ‚expanded storage' (ES) vor, wenn auch nur für einen Rechner (IBM 3090) und v. a. für Paging-Zwecke. Die Zugriffe sind dabei synchron und liegen im unteren Mikrosekundenbereich; die Verwaltung des Speicherbereiches erfolgt durch spezielle Software im Betriebssystem des zugreifenden Rechners. Eine Besonderheit ist, daß alle Seitentransporte zwischen ES und Platte über den Hauptspeicher erfolgen müssen [2].

Um einen solchen Speicher als globalen Systempuffer nutzen zu können, müssen natürlich alle Rechner eines DB-Sharing-Systems darauf zugreifen können, was die Bereitstellung elementarer Operationen zur Zugriffssynchronisation verlangt. Eine einfache Realisierungsform zur Verwaltung des globalen Puffers sieht eine Partitionierung des Speicherbereiches vor, so daß ein Rechner nur auf eine von ihm verwaltete Partition schreibend zugreifen kann, lesend jedoch auf alle Partitionen (ähnlich wie in [12]). Eine Speicherpartition entspricht dann im wesentlichen einer Erweiterung des lokalen Systempuffers, die jedes DBVS lokal verwalten kann (Angaben zur Seitenersetzung, Freispeicherverwaltung u. ä). Zur Realisierung eines Seitenaustausches

kann ein Rechner nach einer Seitenanforderung die Seite in seine Partition schreiben und dem anfordernden Knoten die Adresse mitteilen, von der er die Seite einlesen kann.

Ein Nachteil des Ansatzes ist die starre Partitionierung des globalen Puffers, die eine optimale Speicherplatznutzung i. a. nicht zuläßt, zumal eine Seite in mehreren Partitionen repliziert vorliegen kann. Letzteres kann mit einem Primary-Copy-Sperrverfahren vermieden werden, wenn jeder Rechner nur Seiten in den globalen Puffer schreibt, zu denen er die PCA besitzt. Hierbei kann dann auch mit Gewährung einer Sperre (zur Umgehung einer eigenen Seitenanforderung) ggf. die Adresse im globalen Systempuffer mitgeteilt werden, von der die aktuelle Version der Seite eingelesen werden kann.

Weitere Aspekte zur nahen Kopplung bei DB-Sharing werden in [12, 20, 33, 38] betrachtet.

7. Resümee und Ausblick

Trotz des allgemeinen Trends zur dezentralen Datenverarbeitung werden vor allem im Bereich von Transaktionssystemen die gemeinsamen Datenbanken weiterhin vorwiegend in zentralen Rechenzentren verwaltet werden [51]. Allerdings erfordern die zunehmenden Leistungs- und Verfügbarkeitsanforderungen die Verarbeitung der Transaktionslast durch Mehrrechner-DBS, wobei die lokale Rechneranordnung eine effiziente Kopplung ermöglicht. Der in diesem Aufsatz behandelte DB-Sharing-Ansatz stellt eine vielversprechende Architektur zur Realisierung von Hochleistungs-Transaktionssystemen dar, bei dem keine statische Partitionierung der Datenbestände erforderlich ist. Gegenüber sogenannten DB-Distribution-Systemen ergeben sich so Vorteile bezüglich Lastbalancierung, Datenverfügbarkeit und Erweiterbarkeit. Weiterhin läßt sich zur effizienten Kommunikation und Kooperation der Rechner beziehungsweise zur Optimierung des E/A-Verhaltens (globaler Systempuffer, Logging) eine nahe Kopplung in natürlicher Weise in die Systemarchitektur integrieren.

Der Aufsatz diskutierte Anforderungen und Realisierungsmöglichkeiten derjenigen Systemkomponenten, für die bei DB-Sharing neue und koordinierte Lösungen erforderlich sind. Insbesondere bezüglich der Synchronisation und der Behandlung von Pufferinvalidierungen wurden die wichtigsten Strategien überblickartig vorgestellt und unter Berücksichtigung vorliegender Leistungsuntersuchungen miteinander verglichen. Bei loser Rechnerkopplung ist das Primary-Copy-Sperrverfahren am vielversprechendsten, da es eine integrierte Behandlung von Pufferinvalidierungen (ohne zusätzliche Kommunikationsvorgänge), den Einsatz einer optimierten Synchronisation von Lesezugriffen sowie eine effektive Lastverteilung ermöglicht. Zentrale Sperrverfahren kommen allenfalls bei spezieller Hardware-Unterstützung in Betracht, wobei eine globale Sperranforderung in wenigen Mikrosekunden gewährleistet sein sollte (Kap. 6).

Neben dem quantitativen Leistungsvergleich verschiedener Synchronisationsverfahren für DB-Sharing zeigte die Simulationsstudie in [38] die Bedeutung folgender Punkte bei der Realisierung von Hochleistungs-Transaktionssystemen:

- NOFORCE-Strategie bezüglich des Ausschreibens geänderter DB-Seiten (Kap. 3)
- Affinitäts-basierte Lastverteilung (Kap. 4)
- Geringe Häufigkeit von Interprozessor-Kommunikationen

Dies erfordert bei DB-Sharing koordinierte Lösungen für Synchronisation, Behandlung von Pufferinvalidierungen sowie der Lastverteilung (wie z. B. mit dem Primary-Copy-Verfahren möglich):

- Effiziente Kommunikationsprimitive sowie ein schnelles Kommunikationssystem

Bei DB-Sharing und NOFORCE können unter dieser Voraussetzung auch ganze DB-Seiten wesentlich schneller zwischen den Rechnern ausgetauscht als von Platte eingelesen werden, womit ein verbessertes E/A-Verhalten erreichbar ist.

- Hinreichend geringe Häufigkeit von Synchronisationskonflikten

Diese Forderung kann mit einer Synchronisation auf Seitenebene i. a. nicht erfüllt werden, so daß auch bei DB-Sharing z. B. für Verwaltungsdaten oder andere Hot-Spot-Objekte eine Sonderbehandlung notwendig wird [38].

Die Diskussion in den Kapiteln 4–6 verdeutlichte einige Probleme, die in weiterer Forschungs- und Entwicklungsarbeit noch vertieft zu behandeln und zum Teil über den DB-Sharing-Ansatz hinaus von Bedeutung sind. Dazu zählen die Entwicklung und Bewertung adaptiver Mechanismen zur lokalen und globalen Lastkontrolle, Fehlertoleranz- und Recovery-Maßnahmen in verteilten DB/DC-Systemen sowie neue Architekturkonzepte (Hardware-Unterstützung) zur Realisierung von Hochleistungs-Transaktionssystemen.

Diese Arbeit basiert weitgehend auf meiner Tätigkeit im von der Siemens AG finanzierten Projekt ‚Mehrrechner-Datenbanksysteme' an der Universität Kaiserslautern. Mein besonderer Dank gilt dem Projektleiter Prof. Dr. T. Härder für viele hilfreiche Vorschläge und Diskussionen. Die konstruktiven Hinweise der anonymen Gutachter trugen wesentlich zur Verbesserung des Aufsatzes bei.

Literatur

1. AIM/SRCF functions and facilities. Facom OS Techn. Manual 78SP4900E, Fujitsu Limited 1986
2. IBM 3090 processor complex: expanded storage overview. IBM Int. Techn. Support Centers, ZZ81-0181 (1987)
3. Anon et al.: A measure of transaction processing power. Datamation 112-118, April 1985
4. Bayer, R., Elhardt, K., Kießling, W., Killar, D.: Verteilte Datenbanksysteme - eine Übersicht über den heutigen Entwicklungsstand. Info.-Spek. 7(1), 1-19 (1984)
5. Behman, S. B., DeNatale, T. A., Shomler, R. W.: Limited lock fa-

cility in a DASD control unit. Tech. report TR 02.859, IBM General Products Division, San Jose 1979
6. Bernstein, P.A.: Sequoia: a fault-tolerant tightly coupled multiprocessor for transaction processing. IEEE Comp. 37-45, Februar 1988
7. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency control and recovery in database systems. Addison-Wesley 1987
8. Boral, H., Redfield, S.: Database machine morphology. Proc. 11th Int. Conf. on Very Large Data Bases, 59-71 (1985)
9. Casavant, T.L., Kuhl, J.G.: A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Trans. Software Eng. *14* (2), 141-154 (1988)
10. Cellary, W., Gelenbe, E., Morzy, T.: Concurrency control in distributed databases. Amsterdam: North-Holland 1988
11. Cornell, D.W., Dias, D.M., Yu, P.S.: On multi-system coupling through function request shipping. IEEE Trans. Software Eng. *12* (10), 1006-1017 (1986)
12. Dias, D.M., Iyer, B.R., Robinson, J.T., Yu, P.S.: Design and analysis of integrated concurrency-coherency controls. Proc 13th Int. Conf. on Very Large Data Bases, 463-471 (1987)
13. Effelsberg, W., Härder, T.: Principles of database buffer management. ACM Trans. Database Syst. *9* (4), 560-595 (1984)
14. Gray.J., et al.: One thousand transactions per second. Proc. IEEE Spring CompCon, 96-101 (1985)
15. Gray, J.N., Anderton, M.: Distributed computer systems: four case studies. Proc. IEEE *75* (5), 719-726 (1987)
16. Gray, J.N., Lorie, R.A., Putzolu, G.R., Traiger, I.: Granularity of locks and degrees of consistency in a shared database. Proc. IFIP Working Conf. on Modelling in Data Base Management Systems, 365-394. Amsterdam: North-Holland 1976
17. Härder, T., Meyer-Wegener, K.: Transaktionssysteme und TP-Monitore - Eine Systematik ihrer Aufgabenstellung und Implementierung. Informatik Forsch. Entw. *1* (1), 3-25 (1986)
18. Härder, T., Rahm, E.: Quantitative Analyse eines Synchronisationsalgorithmus für DB-Sharing. Proc.3. GI/NTG-Fachtagung über Messung, Modellierung und Bewertung von Rechensystemen. Informatik-Fachberichte 110, S.186-201. Berlin-Heidelberg-New York: Springer 1985
19. Härder, T., Rahm, E.: Mehrrechner-Datenbanksysteme für Transaktionssysteme hoher Leistungsfähigkeit. IT Informationstechnik *28* (4), 214-225 (1986)
20. Härder, T., Rahm, E.: Hochleistungsdatenbanksysteme - Vergleich und Bewertung aktueller Architekturen und ihrer Implementierung. IT Informationstechnik *29* (3), 127-140 (1987)
21. Härder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Comput. Surv. *15* (4), 287-317 (1983)
22. Hsu, Y.-P.: Performance evaluation of data sharing transaction processing systems. Master's thesis, Univ. of Massachusetts at Amherst 1988
23. Iyer, B.R., Krishna, C.M., Lew, J.S.: Analysis of trade-offs in distributed locking for transaction processing systems. IBM Res. Rep. RC 12802, Yorktown Heights 1987
24. Iyer, B.R., Yu, P.S., Donatiello, L.: Analysis of fault tolerant multiprocessor architectures for lock engine design. Comput. Syst. Sci. Eng. *2* (2), 59-75 (1987)
25. Keene, W.N.: Data sharing overview. IMS/VS VI, DBRC and Data Sharing User's Guide. G320-5911-0 (1982)
26. Knapp, E.: Deadlock detection in distributed databases. ACM Comput. Surv. *19* (4), 303-328 (1987)
27. Kronenberg, N.P., Levy, H.M., Strecker, W.D.: VAX clusters: a closely coupled distributed system. ACM Trans. Comput. Syst. *4* (2), 130-146 (1986)
28. Kung, H.T., Robinson, J.T.: On optimistic methods for concurrency control. ACM Trans. Database Syst. *6* (2), 213-226 (1981)
29. Lyon, J.: Design considerations in replicated database systems for disaster protection. Proc. IEEE Spring CompCon, 428-430 (1988)
30. Menon, J., Hartung, M.: The IBM 3990 disk cache. Proc. IEEE Spring CompCon, 146-151 (1988)
31. Nelson, M.N., Welch, B.B., Ousterhout, J.K.: Caching in the Sprite network file system. ACM Trans. Comput. Syst. *6* (1), 134-154 (1988)
32. Peinl, P.: Synchronisation in zentralisierten Datenbanksystemen. Algorithmen, Realisierungsmöglichkeiten und quantitative Bewertung. Informatik-Fachberichte 161. Berlin-Heidelberg-New York: Springer 1987
33. Rahm, E.: Nah gekoppelte Rechnerarchitekturen für ein DB-Sharing-System. Proc. 9. NTG/GI-Fachtagung über Architektur und Betrieb von Rechensystemen, NTG-Fachberichte 92, S.166-180. VDE 1986
34. Rahm, E.: Algorithmen zur effizienten Lastkontrolle in Mehrrechner-Datenbanksystemen. Angew. Inform. 161-169, April 1986
35. Rahm, E.: Primary copy synchronization for DB-Sharing. Inform. Syst. *11* (4), 275-286 (1986)
36. Rahm, E.: Design of optimistic methods for concurrency control in Database Sharing systems. Proc. 7th IEEE Int. Conf. on Distributed Computing Systems, 154-161 (1987)
37. Rahm, E.: Optimistische Synchronisationskonzepte in zentralisierten und verteilten Datenbanksystemen. IT Informationstechnik *30* (1), 28-47 (1988)
38. Rahm, E.: Synchronisation in Mehrrechner-Datenbanksystemen. Konzepte, Realisierungsformen und quantitative Bewertung. Informatik-Fachberichte 186. Berlin-Heidelberg-New York: Springer 1988
39. Reuter, A.: Database Sharing. Info.-Spekt. *8* (4), 225-226 (1985)
40. Reuter, A.: Load control and load balancing in a shared database management system. Proc. 2nd IEEE Int. Conf. on Data Engineering, 188-197 (1986)
41. Reuter, A., Shoens, K.: Synchronization in a data sharing environment. Techn. Bericht, IBM San Jose Research Lab. 1984
42. Robinson, J.T.: A fast general-purpose hardware synchronization mechanism. Proc. ACM SIGMOD Conf., 122-130 (1985)
43. Rowe, L.A.: A shared storage hierarchy. Proc. Int. Workshop on Object-Oriented Database Systems 1986
44. Scrutchin, Jr., T.W.: TPF: performance, capacity, availability. Proc. IEEE Spring CompCon, 158-160 (1987)
45. Sekino, A., et al.: The DCS - a new approach to multisystem data sharing. Proc. Nat. Comput. Conf., 59-68 (1984)
46. Shoens, K., et al.: The AMOEBA project. Proc. IEEE Spring CompCon, 102-105 (1985)
47. Shoens, K.: Data sharing vs. partitioning for capacity and availability. IEEE Database Eng. *9* (1), 10-16 (1986)
48. Smith, A.J.: Disk cache - miss ratio analysis and design considerations. ACM Trans. Comput. Syst. *3* (3), 161-203 (1985)
49. Stonebraker, M.: The case for shared nothing. IEEE Database Eng. *9* (1), 4-9 (1986)
50. Thomas, R.H.: A majority consensus approach to concurrency control for multiple copies data bases. ACM Trans. Database Syst. *4* (2), 180-209 (1979)
51. Traiger, I.: Trends in systems aspects of database management. Proc. 2nd Int. Conf. on Databases (ICOD-2), 1-20 (1983)
52. West, J.C., Isman, M.A., Hannaford, S.G.: PERPOS fault-tolerant transaction processing. Proc. 3rd IEEE Symp. on Reliability in Distributed Software and Database Systems, 189-194 (1983)
53. Yen, W.C., Yen, D.W.L., Fu, K.: Data coherence problem in a multicache system. IEEE Trans. Comput. *34* (1), 56-65 (1985)
54. Yu, P.S., Balsamo, S., Lee, Y.: Dynamic load sharing in distributed database systems. Proc. Fall Joint Comp. Conf., 675-683 (1986)
55. Yu, P.S., Cornell, D.W., Dias, D.M., Iyer, B.R.: Analysis of affinity based routing in multi-system data sharing. Perform. Eval. *7* (2), 87-109 (1987)
56. Yu, P.S., et al.: On coupling multi-systems through data sharing. Proc. IEEE *75* (5), 573-587 (1987)

Eingegangen 29.8. 1988, in überarbeiteter Form 5.1. 1989

Erhard Rahm
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, 10598 NY
USA