NAH GEKOPPELTE RECHNERARCHITEKTUREN FÜR EIN DB-SHARING-SYSTEM

Erhard Rahm
Universität Kaiserslautern

Abstract
Als DB-Sharing bezeichnet man die gemeinsame Benutzung einer Datenbank durch Datenbank-Verwaltungssysteme (DBVS) auf einem System lose bzw. nahe gekoppelter Prozessoren. Das wesentliche Entwurfsziel solcher Mehrrechner-Datenbanksysteme ist die annähernd lineare Erhöhung des Durchsatzes im Transaktionsbetrieb bei nur wenig veränderten Antwortzeiten im Vergleich zum 1-Rechner-Fall. Weiterhin wird eine deutliche Verbesserung der Verfügbarkeit des Systems für die DB-Verarbeitung angestrebt.
Zunächst werden die Grobarchitektur eines DB-Sharing-Systems sowie die wesentlichen funktionellen Komponenten vorgestellt. Die Anordnung dieser Komponenten im System hat entscheidenden Einfluß auf die Performance und die Verfügbarkeit. Untersucht wird vor allem wie eine nahe Kopplung der Rechner über gemeinsame Speicherpartitionen gewinnbringend verwendbar ist. Weitere Kapitel befassen sich mit der Stellung der Lastkontrolle in einem DB-Sharing-System sowie dem Einsatz von intelligenten Plattensteuerungen. Aus den einzelnen Punkten wird dann ein detaillierterer Architekturvorschlag für ein DB-Sharing-System abgeleitet.

1. Einführung

Bestehende DBS-Anwendungen in klassischen Bereichen wie Bankwesen, Lagerhaltung, Buchhaltung usw., die im Rahmen von Transaktionssystemen zu Auskunfts-, Buchungs- und Datenerfassungsaufgaben eingesetzt werden, verlangen zunehmend die Erweiterung und Verbesserung existierender DBVS oder gar die Neuentwicklung spezieller Systeme, um vor allem folgende Anforderungen erfüllen zu können:

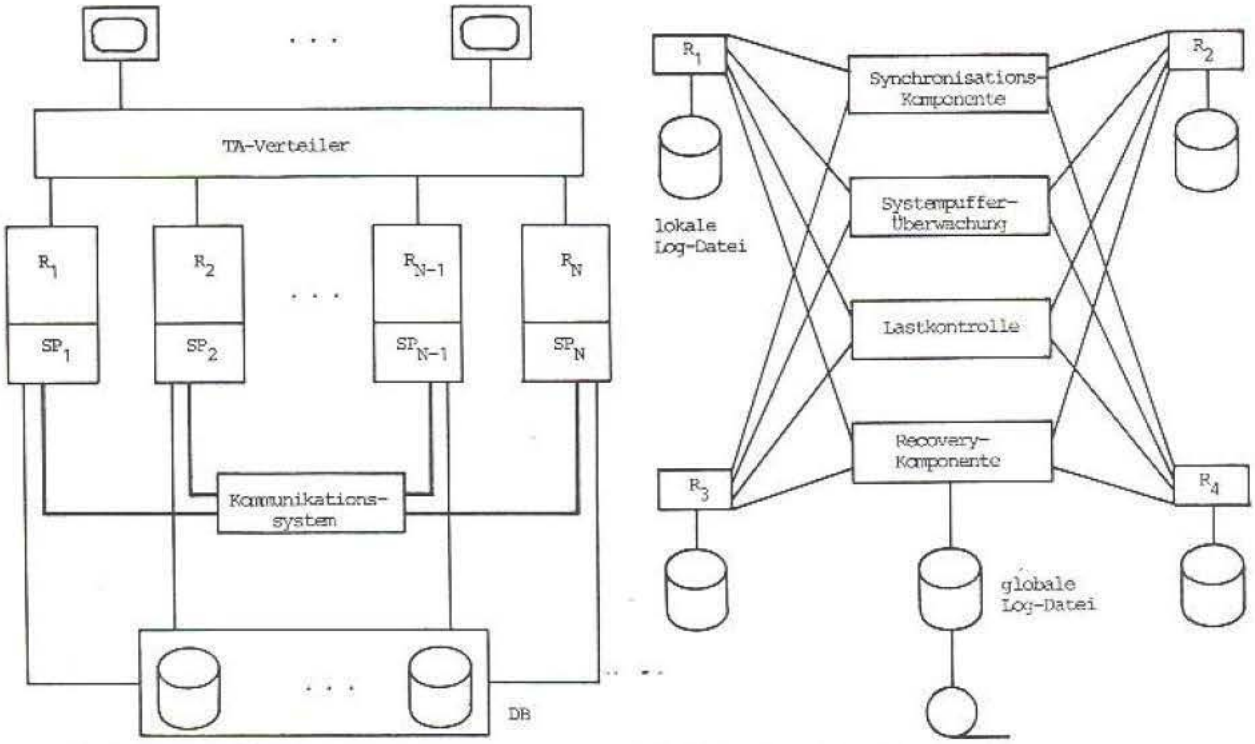
- **Abwicklung hoher Transaktionsraten**
  Der Betrieb von DB/DC-Systemen läßt sich charakterisieren durch die hochparallele Abwicklung von typischerweise kurzen und vorgeplanten Transaktionen (z.B. Kontenbuchung oder Flugreservierung). Datenbanksysteme, die diese Art des Transaktionsbetriebs in optimaler Weise unterstützen, werden oft als Hochleistungs-Datenbanksysteme bezeichnet. Während derzeitige Systeme maximal ca. 200 TA/sec erreichen (IMS/Fast Path), sind bereits 1000 TA/sec konkretes Entwicklungsziel [Gr85].
- **Modulares Systemwachstum mit annähernd linearer Durchsatzerhöhung**
- **Hohe Verfügbarkeit**
  Bei Transaktionssystemen ist diese Systemeigenschaft besonders wichtig. Je mehr TA in einem Transaktionssystem online abgewickelt werden, umso stärker macht sich ein Systemausfall als Störung im Betriebsablauf eines Unternehmens bemerkbar. Bereits heute werden oftmals lediglich 5 Minuten Ausfallzeit pro Jahr als tolerierbar angesehen [Gr85].
  Zur Erfüllung solcher Anforderungen muß jede Hauptkomponente (Prozessor, Kanal, E/A-Kontroller, Plattenlaufwerk, Rechnerkopplung) mindestens doppelt ausgelegt sein, um einen Ausfall verkraften zu können. Weiterhin müssen Fehler transparent für den Benutzer bleiben und Änderungen in der Hardware- oder Software-Konfiguration müssen dynamisch durchgeführt werden können.

In eng gekoppelten Mehrprozessorsystemen teilen sich alle Rechner einen gemeinsamen Hauptspeicher und es existiert nur eine Kopie von Betriebssystem und DBVS. Der gemeinsame Hauptspeicher birgt die Gefahr der Fehlerfortpflanzung, so daß oft ein Systemausfall den anderen nach sich zieht (verseuchte Datenstrukturen, permanent gesperrte kritische Abschnitte). Außerdem bedingt der gemeinsame Hauptspeicher mit zunehmender Rechneranzahl stark wachsende Zugriffskonflikte, so daß meist nur wenige Recher eng

gekoppelt werden (i.a. zwei oder vier) und ein lineares Durchsatzwachstum unmöglich wird.
Eine hohe Verfügbarkeit wird am besten durch ein lose gekoppeltes Mehrrechnersystem unterstützt, in dem n Prozessoren mit eigenen Kopien des Betriebssystems und separaten Hauptspeichern mit Hilfe von Nachrichten zusammenarbeiten [Ki84]. Die Tatsache, daß mehrere Rechner kooperieren, muß nach 'außen' hin verborgen bleiben, um die Benutzung und Wartung des Systems möglichst einfach (und damit wenig fehleranfällig) zu machen. Ein solches 'single system image' ist durch eine komplette Software-Abbildung zu realisieren. Bei einer losen Kopplung ist außerdem selbst bei einer schnellen Rechnerkopplung die Kommunikation über Nachrichten sehr teuer, da sie i.a. mit Prozeßwechseln verbunden ist. Dies macht es schwierig, ähnliche Antwortzeiten wie im zentralisierten Fall zu garantieren, was jedoch für die Akzeptanz solcher Systeme notwendig ist.
Die Lösung dieser Performance-Probleme könnte eine nahe Kopplung der Rechner sein, bei der ein gemeinsames Speichersegment eine effiziente Kommunikation für gewisse Teilaufgaben (z.B. Synchronisation des DB-Zugriffs) erlauben soll. Eine ausreichend hohe Verfügbarkeit soll dennoch ermöglicht werden, da wie bei der losen Kopplung jeder Rechner über eine eigene Betriebssystem-Kopie sowie einen eigenen Hauptspeicher verfügt [HR85b].
Wie aus den bisherigen Ausführungen hervorgeht, erfordert die Realisierung eines Hochleistungs-DBS mehrere lokal angeordnete Rechner, die lose oder nahe gekoppelt sind. Dabei läuft auf jedem Rechner ein selbständiges Datenbankverwaltungssystem (DBVS) ab, das in steter Koordination und Kommunikation mit den übrigen DBVS stehen muß. Alle DBVS müssen gemeinsam die Daten der Datenbank (DB) verarbeiten. Hierzu sind zwei Möglichkeiten der physischen Zuordnung der Daten denkbar:
- alle DBVS können gemeinsam auf alle Daten direkt zugreifen. Diesen Ansatz bezeichnen wir als DB-Sharing.
- ein DBVS verwaltet eine Partition der DB und kann jeweils nur auf seine Partition zugreifen. Daten aus fremden Partitionen müssen explizit angefordert und ausgetauscht werden. Ein solcher Ansatz heißt auch DB-Distribution.

Wir halten DB-Sharing für einen erfolgversprechenden Ansatz, der weiter betrachtet werden soll. Seine prinzipielle Systemstruktur ist in Bild 1a gezeigt. Allgemeine Kriterien und Unterscheidungsmerkmale zu DB-Distribution werden in [Hä85] und [HR85b] untersucht.
In einem DB-Sharing-System werden die an den Terminals gestarteten TA über einen TA-Verteiler auf die Rechner aufgeteilt. Jede TA kann vollständig auf einem Rechner abgearbeitet werden, da jeder Prozessor direkten Zugriff zu allen Daten hat. Dies impliziert, daß alle Rechner räumlich benachbart angeordnet sein müssen, erlaubt jedoch sehr leistungsfähige Rechnerkopplungen (1 - 100 MB/sec).
In Bild 1b sind die wesentlichen Funktionen eines DB-Sharing-Systems gezeigt, für die im Vergleich zu zentralisierten DBVS neue Lösungsansätze erforderlich sind:
- Die Synchronisationskomponente koordiniert den Zugriff zur zentralisierten DB. Bei einer losen Kopplung ist hierbei darauf zu achten, daß dies mit geringstmöglichem Kommunikationsaufwand zwischen den Rechnern möglich wird, da ansonsten die Antwortzeiten stark ansteigen können. Denn neben der eigentlichen Übertragungszeit wird die Wartezeit auf die Antwort zu einer Synchronisationsnachricht (z.B. Lock-Request) dadurch verlängert, daß solche Nachrichten vor der Übertragung erst 'gebündelt' werden, da in heutigen Betriebssystemen das Senden und Empfangen von Nachrichten teuere Operationen sind (ca. 10^4 Instruktionen). Um unter diesen Bedingungen den Durchsatz halten zu können, muß in jedem Rechner der Parallelitätsgrad der TA entsprechend erhöht werden, was jedoch wiederum die Konfliktwahrscheinlichkeit und den BS-Overhead für Scheduling, Paging u.ä. erhöht. Daher ist

hoher Durchsatz bei vertretbaren Antwortzeiten nur zu erreichen, wenn die durchschnittliche Anzahl von Synchronisationsnachrichten pro TA minimal bleibt. Eine weitergehende Diskussion der Synchronisationsproblematik bei DB-Sharing sowie mögliche Algorithmen findet man in [RS84, Ra84, HR85a, Ra85a, Ra85b].
In Kapitel 2 wird gezeigt, wie das Synchronisationsproblem durch den Einsatz eines gemeinsamen Speichersegmentes entschärft werden kann.



a) Grobarchitektur b) funktionelle Komponenten

Bild 1: Aufbau eines DB-Sharing Systems

- Wenn jeder Prozessor eines DB-Sharing-Systems einen eigenen Systempuffer (SP) benutzt, kommt es zu dem sogenannten Veralterungsproblem (Pufferinvalidierung), das durch eine Systempufferüberwachung behandelt werden muß. Denn da eine Modifikation eines DB-Objektes zunächst nur im lokalen Systempuffer vorgenommen wird, wird eine Kopie dieses Objektes in einem der anderen Puffer invalidiert. Die Systempufferüberwachung muß sowohl verhindern, daß auf solchermaßen veraltete Objekte zugegriffen wird, und sie muß ggf. einem Rechner die neueste Version eines Objektes zur Verfügung stellen. Wenn jede Änderungstransaktion die modifizierten Objekte bei EOT in die DB ausschreibt (FORCE-Strategie [HR83]), dann kann die neueste Version eines Objektes stets von der Platte gelesen werden. Bei einer NOFORCE-Strategie kann der Austausch der Änderungen über Platte, über die Koppelstrecke zwischen den Rechnern oder - bei einer nahen Kopplung - über ein gemeinsam benutztes Speichersegment erfolgen. In Kapitel 2 werden weitere Möglichkeiten untersucht, wie mit einer nahen Kopplung das Veralterungsproblem entschärft bzw. gänzlich vermieden werden kann.

- Die Lastkontrolle hat die Aufgabe, die aktuelle TA-Last auf die verfügbaren Rechner aufzuteilen, so daß eine möglichst gute Systemauslastung erreicht wird und die vorgegebenen Anforderungen (Durchsatz, Antwortzeiten) erfüllt werden. Sie muß daher sicherstellen, daß keiner der Rechner überlastet ist und daß die TA so verteilt werden, daß möglichst wenig (Synchronisations-)

Nachrichten zwischen den Rechnern erforderlich sind. Außerdem muß die Lastkontrolle dynamisch auf Änderungen in der Systemkonfiguration (z.B. Ausfall oder Wiedereinbringen eines Rechners) reagieren. In Kapitel 3 wird diese wichtige Systemkomponente genauer vorgestellt sowie ihre Stellung innerhalb des Systems untersucht.

- Logging und Recovery sind Aufgabe der Recovery-Komponente. Während das Zurücksetzen einzelner TA sowie die Recovery nach Ausfall eines Rechners mit lokalen Log-Dateien durchgeführt werden kann, erfordert die Platten-Recovery einen globalen Log, der durch geeignetes Mischen der lokalen Log-Daten erstellt wird. Für dieses Mischen kann wiederum ein gemeinsames Speichersegment nützlich sein (siehe Kapitel 2).
  Die Recovery-Komponente hat sicherzustellen, daß der Ausfall eines Rechners erkannt und daß die Recovery durch die überlebenden Prozessoren schnellstmöglich durchgeführt wird, um die TA-Verarbeitung möglichst wenig zu behindern. Die gescheiterten TA sind unter Zuhilfenahme eines Nachrichten-Logs auf den verfügbaren Rechnern erneut zu starten, so daß der Rechnerausfall für den Benutzer transparent bleibt.

Im nächsten Kapitel werden mögliche Organisationsformen einer nahen Kopplung untersucht. Hierbei konzentrieren wir uns auf die Verwendung von gemeinsam benutzten Speichersegmenten, die, wie bereits mehrfach angedeutet, für wesentliche Funktionskomponenten eines DB-Sharing-Systems nutzbringend einsetzbar sind. In Kapitel 3 folgt eine genauere Diskussion über die Rolle der Lastkontrolle sowie über mögliche Realisierungsformen. Danach sollen noch sogenannte 'intelligente Plattensteuerungen' im Hinblick auf ihre Tauglichkeit in einem DB-Sharing System betrachtet werden. In der Zusammenfassung wird dann ein genauerer Architekturvorschlag präsentiert, der aus der vorangegangenen Diskussion abgeleitet ist.

2. Nutzung von gemeinsamen Speicherpartitionen

Die lose Kopplung zwischen den Verarbeitungsrechnern eines DB-Sharing-Systems garantiert zwar eine weitestgehende Autonomie der Prozessoren, jedoch bewirkt die teuere Kooperation und Kommunikation zwischen den Rechnern über Nachrichten erhebliche Performance-Probleme. So bedeutet für kurze TA, wie sie in DB/DC-Anwendungen typisch sind, selbst eine Kommunikation oft eine deutliche Erhöhung der Antwortzeit im Vergleich zu zentralisierten DBS. Der Austausch der Nachrichten ist dabei vor allem zur Synchronisation der DB-Zugriffe notwendig, die daher in einem lose gekoppelten DB-Sharing-System zur leistungsbestimmenden Komponente wird.
Als offensichtliche Lösungsmöglichkeit für diese Probleme bietet sich eine nahe Kopplung der Prozessoren an, die beste Performance bei ausreichender Verfügbarkeit bieten soll. In diesem Kapitel soll untersucht werden, wie gemeinsame (d.h. von allen Rechnern erreichbare) Halbleiterspeichersegmente für eine nahe Kopplung eingesetzt werden können. Nicht betrachtet werden also sogenannte "Lückenfüllertechnologien" (Magnetblasen- oder CCD-Speicher), da sie nahezu keine praktische Relevanz besitzen. Ebenso ausgeklammert bleibt eine mögliche Verwendung spezieller Hardware-Einheiten, wie etwa eine 'Lock-Box' zur globalen Synchronisation [HR85b].
Bei der Verwendung eines gemeinsamen Halbleiterspeichers ist natürlich darauf zu achten, daß damit kein Engpaß erzeugt wird. Daher soll die Kommunikation zwischen den Rechnern nicht nur über solche Speichersegmente vorgenommen werden, sondern sie sollte sich auf wesentliche Funktionen beschränken. Weiterhin muß man sich darüber im klaren sein, daß eine gemeinsame Speicherpartition zur Ausbreitung von Fehlern führen kann und daß einem Ausfall geeignet vorgebeugt werden muß (z.B. durch ein Ersatz-Speichersegment).
Wie die Nutzung eines gemeinsamen Halbleiterspeichers aussehen kann, ist im wesentlichen bestimmt durch die

- Eigenschaften des Halbleiterspeichers sowie die

- Verwendung innerhalb des Systems.

Bezüglich der Eigenschaften sind folgende drei Möglichkeiten zu betrachten:

1. Der Halbleiterspeicher ist flüchtig und für alle Rechner instruktionsadressierbar. Dies bedeutet, daß das gemeinsame Speichersegment für jeden Rechner als Teil seines Hauptspeichers angesehen werden kann. Da von allen Rechnern auf den Speicherbereich zugegriffen wird, sollte er nicht seitenwechselbar sein (Realspeicher).
2. Der Halbleiterspeicher ist flüchtig und für alle Rechner seitenadressierbar.
3. Der Halbleiterspeicher ist nichtflüchtig und für alle Rechner seitenadressierbar. (Solche Speicher wurden bereits von den Firmen Fujitsu und NAS angekündigt, wobei die Kapazität jeweils bis zu 512 MB betragen soll bei einer Zugriffszeit von 1 ms. Die Nicht-Flüchtigkeit wird durch eine Batterie und angeschlossenen Festplattenspeicher gewährleistet, so daß bei Stromausfall die im Halbleiterspeicher befindlichen Seiten noch gesichert werden können.)

Ein technisches Problem für die genannten Speicherarten scheint die Anzahl der anschließbaren Rechner zu sein. So können derzeit meines Wissens nur maximal 4 (Groß-)Rechner solchermaßen gekoppelt werden.

Als Verwendungsformen solcher gemeinsamer Halbleiterspeicher kommen in einem DB-Sharing-System folgende Möglichkeiten in Betracht:

1. Synchronisation
   Die Synchronisation und die Behandlung von Deadlocks kann quasi wie in einem zentralisierten DBVS erfolgen, wenn eine globale Sperrtabelle in dem gemeinsamen Halbleiterspeicher geführt wird. In jedem Rechner arbeitet ein Serrverwalter mit der globalen Sperrtabelle, wobei z.B. über Semaphore die Zugriffe auf diese gemeinsame Datenstruktur geregelt werden. Diese Vorgehensweise setzt Instruktionsadressierbarkeit auf dem gemeinsamen Halbleitspeicher voraus (Speichertyp 1), da sonst Seiten in den lokalen Hauptspeicher (und bei Änderungen wieder zurück) übertragen werden müßten. Bei dieser Strategie dürften selbst bei hohen TA-Raten keine nennenswerten Wartesituationen beim Zugriff auf die globale Sperrtabelle entstehen. Denn wenn pro TA 20 Zugriffe auf die globale Sperrtabelle zu jeweils 200 Instruktionen notwendig sind, dann ergeben sich für 1000 TA/sec 20000 Zugriffe in der Sekunde. Für 20-MIPS-Rechner dauert jeder Zugriff 10µs, so daß nur in 20% der Zeit ein Zugriff auf die Sperrtabelle vorliegt. Ein Ausfall der Sperrtabelle würde allerdings Systemstillstand bedeuten, so daß eine Kopie der globalen Sperrtabelle in einem unabhängigen Speichersegment geführt werden sollte, die jederzeit auf dem aktuellsten Stand ist.
2. Unterstützung der Systempufferüberwachung
   Wenn jeder Rechner seinen eigenen Systempuffer hat, dann muß das in Kapitel 1 erwähnte Veralterungsproblem in Kauf genommen werden. Dieses Problem entfällt, wenn ein globaler Systempuffer in einem gemeinsamen Halbleiterspeicherbereich eingerichtet wird und auf lokale Puffer verzichtet wird. In dem gemeinsamen Speicherbereich sind auch die Datenstrukturen zu führen, die zur Pufferverwaltung benötigt werden. Um ein ständiges Kopieren von Seiten zwischen globalem Puffer und lokalem Hauptspeicher zu vermeiden, muß auch hier Instruktionsadressierbarkeit vorliegen (Speichertyp 1). Der globale Puffer sollte sehr groß sein (z.B. >100 MB), um die Anzahl der physischen E/A's zu begrenzen. Während die Zugriffshäufigkeit zum globalen Puffer ähnlich der für die globale Sperrtabelle sein dürfte, ist i.a. mit längeren Zugriffszeiten zu rechnen. Dies macht es zwingend erforderlich, daß mehrere Rechner (TA) gleichzeitig auf den globalen Puffer zugreifen können, um Engpaßsituationen zu vermeiden. Problematisch dürfte es jedoch sein, den Ausfall des globalen Puffers zu verkraften, da das Führen einer Kopie kaum realistisch sein dürfte.

Eine weitere Organisationsmöglichkeit ist, einen globalen Puffer und lokale Systempuffer zu führen, wobei der globale Puffer dann seitenadressierbar sein kann (Speichertyp 2 oder 3). Der globale Puffer entschärft das Veralterungsproblem, wenn geänderte Seiten über ihn ausgetauscht werden, was i.a. wesentlich schneller gehen dürfte als über Leitungen oder gar über Platte. Wenn der globale Puffer eine echte Erweiterung der Speicherhierarchie zur Reduzierung der Platten-E/A sein soll, ist darauf zu achten, daß nur aktuelle Seiten in ihm enthalten sind. Dies jedoch macht i.a. eine write-through-Strategie auf den globalen Puffer erforderlich, um (freigegebene) Änderungen für alle sichtbar zu machen. Die write-through-Strategie vereinfacht die Behandlung des Veralterungsproblems, da die neueste Version einer Seite stets vom globalen Puffer bzw. von Platte gelesen werden kann (es muß jedoch nach wie vor der Zugriff auf veraltete Seiten in den lokalen Systempuffern verhindert werden). Eine benötigte Seite wird zuerst im lokalen Puffer gesucht, dann im globalen Puffer und erst wenn sie auch da nicht vorhanden ist, wird die Seite von der Platte gelesen. Bei einer FORCE-Ausschreibstrategie, bei der bisher geänderte Seiten einer Update-TA bei EOT auf Platte geschrieben wurden, ist durch einen globalen Puffer eine deutliche Beschleunigung dieser Schreibvorgänge zu erwarten; jedoch muß der globale Puffer nichtflüchtig sein (Speichertyp 3).
Für NOFORCE dagegen kann das Durchschreiben von Änderungen auf den globalen Puffer eine merkliche Verlangsamung im Vergleich zu zentralisierten DBVS bewirken. Diese Beeinträchtigungen können umgangen werden, wenn man nicht nur auf Platte sondern auch im globalen Puffer veraltete Seiten zuläßt. Der Zugriff auf solche Seiten ist dann durch geeignete Zusatzmaßnahmen zu verhindern, ähnlich wie in einem System ohne globalen Puffer der Zugriff auf veraltete Seiten auf der Platte zu vermeiden ist. Mögliche Verfahren hierzu wurden bereits vorgeschlagen (z.B. durch Verwendung von Haltesperren in [Ra84, HR85a] oder eines Invalidierungsvektors in [Ra85b]).
Während bei FORCE der globale Puffer nichtflüchtig sein muß, hat man bei NOFORCE die Wahl zwischen einem flüchtigen und einem nichtflüchtigen Halbleiterspeicher. Ein flüchtiger Halbleiterspeicher (Speichertyp 2) dürfte billiger und wahrscheinlich schneller als ein nichtflüchtiger sein. Um jedoch einen Ausfall eines solchen Speichers tolerieren zu können, darf eine gegenüber der Platte geänderte Seite nicht nur im globalen Puffer sein, sondern zusätzlich in einem der lokalen Puffer. Ein Verdrängen einer geänderten Seite aus dem lokalen Puffer erfordert also ein Ausschreiben auf Platte. Dies ist bei einem nichtflüchtigen globalen Puffer nicht erforderlich, wodurch ein systemweites Akkumulieren von Änderungen im globalen Puffer möglich wird (auch für FORCE!). Insgesamt gesehen ist ein globaler Puffer besonders für eine FORCE-Ausschreibstrategie gewinnbringend. Für NOFORCE sind zwar auch mit einem flüchtigen Halbleiterspeicher Performance-Gewinne möglich, jedoch in geringerem Ausmaß.

3. Unterstützung der Lastkontrolle
Die Verwendung eines gemeinsamen, instruktionsadressierbaren Halbleiterspeichers zur Unterstützung der Lastkontrolle wird im nächsten Kapitel ausführlich diskutiert. Ein solcher Speicherbereich kann einerseit für gemeinsam benutzte Datenstrukturen genutzt werden, wenn die Lastkontrolle auf mehreren Rechnern abläuft. Andererseits können über ihn effizient TA zur Ausführung in einen anderen Rechner weitergeleitet werden, wenn dies sinnvoll erscheint.

4. Unterstützung des Logging
Generell kann das Logging durch Verwendung von nichtflüchtigen Halbleiterspeichern beschleunigt werden, da die Zugriffszeiten auf solche Speicher (auch im Vergleich zu chained-I/O) höhere Ausschreibraten als auf Platten

zulassen. Auch die Erstellung eines globalen Log kann erheblich erleichtert werden, wenn man einen gemeinsamen seitenadressierbaren Halbleiterspeicher als globalen Logpuffer einsetzt. Hierbei ist ein nichtflüchtiger Halbleiterspeicher sicher sinnvoll, jedoch nicht zwingend notwendig, da die Log-Daten in den lokalen Log-Dateien gesichert sind. Ein Ausschreiben auf den globalen Log-Puffer ist dann wegen des schnellen Zugriffs parallel zum lokalen Logging möglich, so daß das Mischen der lokalen Log-Daten sehr einfach werden dürfte. Es muß jedoch dafür Sorge getragen werden, daß die Log-Daten nur kurz im Log-Puffer verbleiben, um einem Überlauf vorzubeugen, der bei dem zu erwartenden Log-Umfang leicht möglich ist. Denn bei 1000 TA/sec und 50% Änderungstransaktionen, ergeben sich 2000 Änderungen/sec, wenn jede ändernde TA durchschnittlich 4 DB-Objekte modifiziert. Bei einer Log-Puffer-Größe von 512 MB und physischem Logging auf Seitenebene wäre der Puffer bei einer Seitengröße von 2 KB innerhalb von 128 sec gefüllt. Dies macht klar, daß bei Verwendung eines globalen Log-Puffers eintragsweises Logging und wahrscheinlich Gruppencommit [Gr85] dringend erforderlich sind, da damit das Datenvolumen drastisch reduziert werden kann (Faktor 20).

In Bild 2 sind die angesprochenen Einsatzformen eines gemeinsamen Halbleiterspeichers zusammengestellt.

Verwendungsform / Speichertyp	Synchronisation	Systempufferkontrolle ohne lokale Puffer	Systempufferkontrolle mit lokalen Puffern	Lastkontrolle	Logging
1 instruktionsadressierbar	X	X	-	X	(X)
2 seitenadressierbar, flüchtig	-	-	X	-	(X)
3 nichtflüchtig	-	-	X	-	X

X = geeignet (X) = bedingt geeignet - = nicht sinnvoll

Bild 2: Einsatzformen von gemeinsam benutzten Halbleiterspeichern in einem nah gekoppelten DB-Sharing-System

Wie gezeigt wurde, kann also für die vier wesentlichen Komponenten eines DB-Sharing-Systems (Synchronisation, Systempufferverwaltung, Lastkontrolle, Logging) eine nahe Kopplung über einen instruktionsadressierbaren, flüchtigen Halbleiterspeicher verwendet werden. Dieser Speichertyp birgt jedoch ähnliche Gefahren bezüglich der Verfügbarkeit wie der gemeinsame Hauptspeicher in eng gekoppelten Mehrrechnersystemen. Daher muß durch geeignete Vorkehrungen und Protokolle dafür gesorgt werden, daß ein Rechnerausfall keine "unerwünschten" Spuren in dem gemeinsamen Speicherbereich hinterläßt. Insbesondere muß durch gegenseitige Überwachung verhindert werden, daß kritische Abschnitte endlos gesperrt bleiben.

Ob und in welchem Umfang gemeinsame Halbleiterspeicher in einem DB-Sharing-System eingesetzt werden sollten, ist stark von folgenden Faktoren abhängig:

- geforderte Verfügbarkeitsanforderungen
- Kosten
- Anzahl der Rechner, die an den gemeinsamen Halbleiterspeicher anschließbar sind.

Eine hohe Verfügbarkeit ist umso schwieriger zu realisieren, je mehr Funktionen zentralisiert werden, v.a. bei Verwendung von flüchtigen Halbleiterspeichern. Wenn z.B. bei einem gemeinsam benutzten flüchtigen Halbleiterspeicher der Ausfall eines Rechners den Ausfall dieses Speichers bewirkt, besitzt dieser gemeinsame Speicher eine sehr hohe Ausfallwahrscheinlichkeit. Nichtflüchtige Halbleiterspeicher dagegen sind sogar wesentlich teurer als Magnetplatten, so daß sie aus Kostengründen nur begrenzt eingesetzt werden dürften. Die Engpaßgefahr ist als weniger problematisch anzusehen; sie wäre jedoch gegeben, wenn man mehrere der genannten Ver-

wendungsformen mit einem gemeinsamen Halbleiterspeicher realisieren wollte.

Aus diesen Überlegungen heraus sollen nun abschließend noch einmal einige Architekturbeispiele (ohne Lastkontrolle und Logging) gezeigt werden:

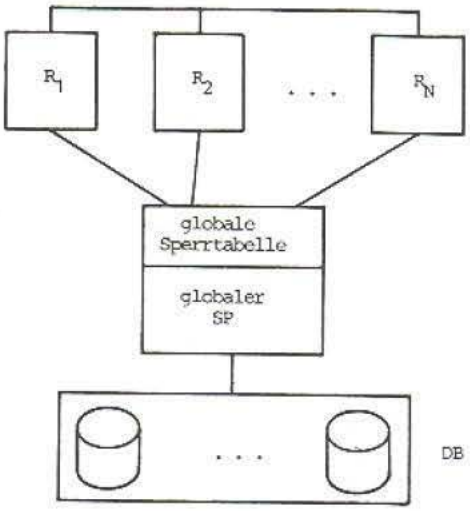


Bild 3: Architekturbeispiel 1

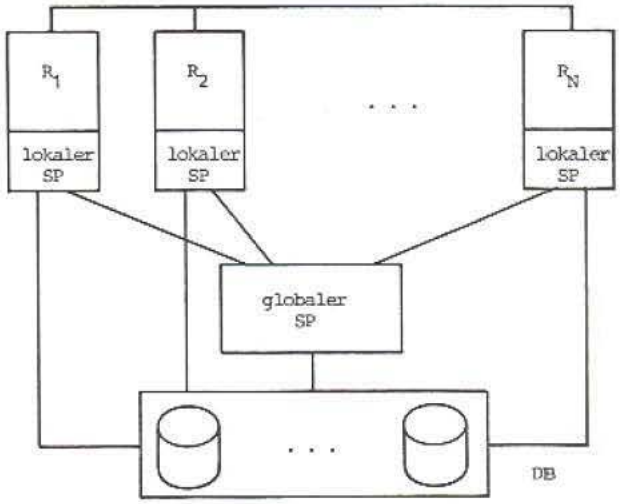


Bild 4: Architekturbeispiel 3

Beispiel 1: Globale Sperrtabelle, globaler Systempuffer ohne lokale Puffer (Bild 3)
In diesem System kann die Synchronisation gleichsam wie in einem zentralen System durchgeführt werden und es existiert kein Veralterungsproblem. Da der (die) gemeinsame(n) Halbleiterspeicher wegen der Instruktionsadressierbarkeit flüchtig ist (sind) ergeben sich v.a. wegen des globalen Puffers kaum tolerierbare Verfügbarkeitsprobleme. Liegen die globale Sperrtabelle und der globale Puffer in demselben Speichersegment, dürften die hohen Zugriffsraten merkliche Wartezeiten verursachen (Engpaß). Der nächste Schritt wäre daher entweder nur eine globale Sperrtabelle oder nur einen globalen Puffer (ohne lokale Puffer) vorzusehen. Letztere Möglichkeit soll jedoch wegen der Verfügbarkeitsprobleme nicht betrachtet werden.

Beispiel 2: globale Sperrtabelle, kein globaler Systempuffer
Dieser Ansatz hat im Vergleich zu Architekturbeispiel 1 geringere Verfügbarkeitsprobleme und dürfte keinen Engpaß erzeugen (s.o.). Die Synchronisation ist relativ einfach und effizient durchführbar, jedoch muß das Veralterungsproblem behandelt werden. Dies kann zusammen mit der Synchronisation geschehen, wenn man z.B., wie in [Ra85b] gezeigt, die Sperrkontrollblöcke der globalen Sperrtabelle um einen sogenannten Invalidierungsvektor erweitert.

Beispiel 3: seitenadressierbarer, flüchtiger globaler Systempuffer (Bild 4)
Dieser Ansatz ist nur für eine NOFORCE-Ausschreibstrategie von Interesse. Für ungeänderte Seiten kann der globale Puffer Lesezugriffe auf die Platten einsparen; geänderte Seiten können über ihn effizient zwischen den Rechnern ausgetauscht werden. Der Vorteil der Architektur liegt in der möglichen Performancesteigerung, ohne Verfügbarkeitsprobleme zu verursachen, da nach einem Ausfall Änderungen wieder direkt oder über Platte ausgetauscht werden.

Beispiel 4: nichtflüchtiger globaler Systempuffer
Bei diesem Ansatz kommt neben dem beschleunigten Austausch von Änderungen der Vorteil der Nichtflüchtigkeit hinzu, der v.a. bei einer FORCE-Ausschreibstrategie wichtig ist. Weiterhin wird auch ein "normales" Verdrängen einer geänderten Seite aus dem lokalen Puffer beschleunigt, da hierzu keine Platten-E/A mehr erforderlich ist. Damit wird ein systemweites Akkumulieren von Änderungen im globalen Puffer ermöglicht. Diese

beiden letztgenannten Vorteile sind natürlich auch bei einer NOFORCE-Strategie relevant, für die dann Änderungen auf zwei Ebenen akkumuliert werden könnten, bevor sie in der materialisierten DB [HR83] sichtbar würden. Nachteilig gegenüber Architekturbeispiel 3 sind die höheren Kosten sowie die vermutlich langsameren Zugriffszeiten.

3. Stellung der Lastkontrolle in einem DB-Sharing-System

Wie bereits in der Einführung erwähnt, hat die Lastkontrolle in einem DB-Sharing-System im wesentlichen folgende Aufgaben:

1. Verteilung der von den Terminals gestarteten TA auf die verfügbaren Rechner (TA-Routing), so daß die TA möglichst schnell bearbeitet werden können.
2. Erreichung einer relativ gleichen Auslastung der verfügbaren Rechner (Lastbalancierung) und Gewährleistung des notwendigen Durchsatzes. Es muß hierzu u.a. verhindert werden, daß ein einzelner Rechner überlastet wird.
3. Zur Erfüllung der beiden genannten Aufgaben, muß die Lastkontrolle auf Änderungen in der Last (z.B. stark wechselndes Referenzverhalten) sowie in der Systemkonfiguration (z.B. bei Ausfall oder Hinzufügen eines Prozessors) geeignet reagieren.

Da bei DB-Sharing jeder Rechner auf alle Daten direkt zugreifen kann, ist es für ein effektives TA-Routing ausschlaggebend, insbesondere bei einer losen Kopplung der Rechner, die Anzahl der Nachrichten zur globalen Synchronisation zu minimieren. Diese Aufgabe kann natürlich nur in enger Zusammenarbeit mit dem gewählten Synchronisationsalgorithmus durchgeführt werden, jedoch ist es hierzu i.a. erforderlich, in den Rechnern eine möglichst hohe Lokalität im Referenzverhalten der dort ablaufenden TA zu erreichen [Re85]. Eine solche Lokalität bewirkt i.a. neben der Synchronisationsunterstützung auch eine Reduktion von physischen E/A-Vorgängen, da viele der benötigten Objekte bereits im lokalen Systempuffer vorhanden sind.

Um nun die ersten zwei der oben genannten Aufgaben wirkungsvoll durchführen zu können, benötigt die Lastkontrolle die

- Vorhersage des Referenzverhaltens einer TA sowie die
- Vorhersage der Auswirkung einer TA auf die aktuelle Last.

Zur Vorhersage des Referenzverhaltens kann die Lastkontrolle die Informationen benutzen, die bei Ankunft einer TA bekannt sind. Dies sind i.a.: TA-Typ, Subschema, Terminal-Identifikation und möglicherweise Eingabedaten. Damit kann dann grob abgeschätzt werden, welche Datenbanken, Satztypen usw. mit welchem maximalen Zugriffsmodus wahrscheinlich referenziert werden, falls über den betreffenden TA-Typ ausreichende Informationen bezüglich des Referenzverhaltens bekannt sind. Ist dies nicht der Fall, müssen diese Informationen möglicherweise erst im laufenden Betrieb gesammelt werden.

Um die Auswirkungen einer TA auf die aktuelle Last vorhersagen zu können, muß die Lastkontrolle neben dem wahrscheinlichen Referenzverhalten der TA auch die Auslastung der einzelnen Rechner kennen sowie möglicherweise Informationen bezüglich des globalen Synchronisationszustandes und der Lokalität von Daten besitzen. Die Auslastung der Rechner ist der Lastkontrolle zumindest grob bekannt, da sie weiß, welche TA sie auf welchen Rechner geleitet hat. Weitergehende Informationen zur Rechnerauslastung erfordern i.a. explizite Kommunikation.

Da in einem System von beispielsweise 1000 TA/sec das TA-Routing sehr häufig durchzuführen ist und sich die dafür benötigte Zeit auch in der Antwortzeit niederschlägt, muß die Lastkontrolle die Zuordnung einer TA zu einem Rechner sehr schnell treffen können. Reuter argumentiert in [Re85], daß hierzu eine TA-Typ-bezogene Routing-Tabelle benutzt werden kann, da das Referenzverhalten in Transaktionssystemen über längere Zeiträume (Stunden) homogen sei ("hyper-locality"). In einer solchen Tabelle werden zu jedem TA-Typ ein oder mehrere Rechner genannt, in denen TA dieses Typs ausgeführt werden sollen. Welcher Rechner ausgewählt wird, hängt dann von der aktuellen Auslastung bei

Eintreffen der TA ab.
Bei Erstellung einer solchen Routing-Tabelle wird eine bestimmte Anzahl von Rechnern sowie ein bestimmtes Lastprofil vorausgesetzt. Ändert sich die Rechnerkonfiguration oder das Lastprofil, muß i.a. die Routing-Tabelle geändert werden. Solche Änderungen sind jedoch relativ selten, so daß das TA-Routing weitgehend tabellengesteuert durchgeführt werden kann. Änderungen im Lastverhalten können relativ einfach durch Überwachung der Ankunftsraten von TA-Typen erkannt werden, wenn TA-Typen als 'stabil' angesehen werden (d.h. homogenes Referenzverhalten innerhalb eines TA-Typs).
Bisher wurde noch nicht diskutiert, wo die Lastkontrolle in einem DB-Sharing-System sinnvollerweise anzusiedeln ist. Da alle von Terminals gestarteten TA über die Lastkontolle auf die Rechner verteilt werden, kann die Lastkontrolle als logische Erweiterung des DC-Systems angesehen werden. Bezüglich der Zuordnung von Lastkontrolle zu Rechnern sind folgende Alternativen zu betrachen:

1.) Die Lastkontrolle wird von einem Rechner (zentral) oder von mehreren (dezentral) ausgeführt.
2.) Die Lastkontrolle wird in einem oder mehreren Front-Ends (FE) durchgeführt oder von einem oder allen Verarbeitungsrechnern (VAR).

Wenn man die Lastkontrolle auf einem oder mehreren FE durchführt, kann man sich zudem noch überlegen, ob man das "eigentliche" DC-System auch auslagert oder weiterhin in den VAR beläßt. Dabei wird angenommen, daß das DC-System die TA-Programme (TAP) ausführt und die Kommunikation zwischen TAP und Terminal einerseits und TAP und DBVS andererseits durchführt. Außerdem führt die DC-Komponente einen Nachrichten-Log. Eine genauere Beschreibung der DC-Komponente sowie mögliche Realisierungsformen findet man in [HM85].
Im folgenden sollen die vier sich ergebenden Zuordnungsmöglichkeiten für die Lastkontrolle diskutiert werden:

a) Zentrale Lastkontrolle auf einem VAR
Da ein VAR typischerweise ein universeller Großrechner ist, sind das Senden und Empfangen von Nachrichten i.a. sehr teure Operationen (z.B. 10^3-10^4 Instruktionen). Daher wäre der Rechner, der die Lastkontrolle durchführt, kaum noch in der Lage eigene TA-Verarbeitung durchzuführen, d.h. sämtliche TA müßten weitergeleitet werden. Zur Verdeutlichung folgendes Zahlenbeispiel:
Wenn ein RECEIVE (SEND) 5000 Instruktionen kostet, sind bei 1000 TA/sec 5 MIPS für das Empfangen der Terminalnachrichten erforderlich und nochmals 5 MIPS zum Weiterleiten der TA-Nachrichten auf den Rechner, der die TA durchzuführen hat. Wenn der abarbeitende Rechner nicht direkt die Antwortnachricht an das Terminal schicken kann, ist nochmals Kommunikation mit dem VAR, auf dem die Lastkontrolle läuft, erforderlich. Für das Empfangen und Weiterleiten der Antwortnachrichten sind also ggf. weitere 10 MIPS notwendig.
Um den Kommunikationsoverhead zu reduzieren, kann man auf eine nahe Kopplung der VAR übergehen, wobei ein gemeinsamer, instruktionsadressierbarer Speicherbereich verwendet wird. Dann können die von einem Rechner R zu verarbeitenden TA-Nachrichten in einem festen Teil dieses Speicherbereichs abgelegt werden, der von R in periodischen Abständen auf neu angekommene Nachrichten überprüft wird. Umgekehrt können auch Antwortnachrichten über den gemeinsamen Speicherbereich zurück zur Lastkontrolle gebracht werden, die diese dann an das entsprechende Terminal weiterleitet.
Da nach Ausfall des Rechners ein anderer VAR die Lastkontrolle durchführen muß, müssen alle Terminals zumindest mit zwei VAR verbunden sein. Wenn die Antwortnachrichten nicht über die Lastkontrolle laufen, sondern direkt vom abarbeitenden Rechner an das Terminal geschickt werden sollen, muß jeder Rechner mit allen Terminals verbunden sein.

b) Dezentrale Lastkontrolle auf allen VAR
Bei diesem Ansatz ist jedes Terminal zunächst einem Rechner fest zuge-

ordnet; jedoch muß für den Fehlerfall vorgesehen werden, daß zumindest noch ein weiterer VAR erreichbar ist. Da die Lastkontrolle in jedem VAR läuft, wird das TA-Routing immer in dem einem Terminal zugeordneten Rechner durchgeführt. Dies hat den Vorteil, daß der Kommunikationsoverhead gleichmäßig auf alle Rechner verteilt wird.
Auch hier ist eine nahe Kopplung der Rechner sinnvoll, um TA schnell an andere VAR weiterleiten zu können. Ebenso können ggf. Antwortnachrichten für ein Terminal zwischen den VAR über den gemeinsamen Speicherbereich ausgetauscht werden, wenn nur der einem Terminal primär zugeordnete Rechner Nachrichten an das Terminal schicken kann. Zudem erlaubt erst die nahe Kopplung ein lastabhängiges TA-Routing, da sich bei hohen TA-Raten die Auslastung der Rechner sehr schnell ändern kann. Bei einer losen Kopplung wäre es daher i.a. nicht möglich, aktuelle Informationen zur Auslastung der anderen Rechner zu führen. Eine nahe Kopplung erlaubt es, diese Informationen in dem gemeinsamen Speicherbereich zu halten, ebenso wie die von allen Rechnern benutzte Routing-Tabelle. Wenn keine nahe Kopplung vorläge, wäre das relativ statische Verteilen der TA über eine Routing-Tabelle zumindest in Frage gestellt, da i.a. nicht vorhersehbar ist, in welchem Rechner eine TA-Nachricht eintrifft und da ein ggf. notwendiges Verschicken dieser Nachricht zu einem anderen VAR teurer sein kann, als mögliche Vorteile durch das Auslagern auf einen anderen Rechner.
Wegen der Dezentralisierung wird es schwieriger Änderungen im TA-Verhalten zu erkennen, da hierzu die Ankunftsraten der TA-Typen bei allen Rechnern zusammengefaßt werden müssen, was sinnvollerweise auch über eine nahe Kopplung geschieht. Die Durchführung dieser Aufgabe wird von einem festzulegenden Rechner vorgenommen, der dann auch die neue Routing-Tabelle berechnet, wenn ein stark verändertes Referenzverhalten registriert wird oder wenn ein Prozessor ausgefallen bzw. hinzugekommen ist. Wenn dieser 'ausgezeichnete' Rechner ausfällt, muß ein anderer seine Funktion übernehmen.

c) Zentrale Lastkontrolle auf einem FE
Bei dieser Zuordnung wird angenommen, daß der FE ein spezieller Kommunikationsrechner ist, der eine effiziente Kommunikation erlaubt. Weiterhin soll der FE mit allen Terminals und allen VAR verbunden sein (Bild 5). Alle Nachrichten von und zu den Terminals laufen über diesen FE.

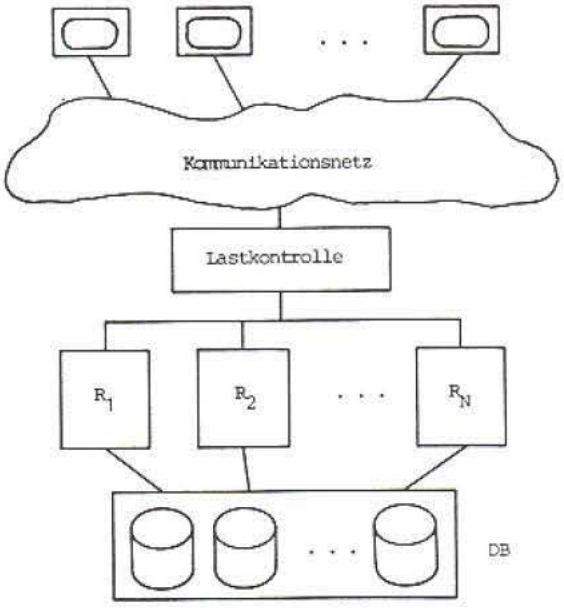


Bild 5: Zentrale Lastkontrolle auf einem Front-End

Wenn man 1000 TA/sec annimmt, muß der FE pro Sekunde jeweils 1000 Nachrichten vom Terminal empfangen, an die VAR weiterleiten, von den VAR empfangen und an die Terminals zurückschicken. Hierfür sind bei 500 Instruktionen für das Senden und Empfangen einer Nachricht 2 MIPS erforderlich. Das Routing selbst fällt dagegen kaum ins Gewicht, da es tabellengesteuert abläuft. Weil ansonsten im wesentlichen nur noch die Überwachung der TA-Raten sowie gelegentlich eine Neuberechnung der Routing-Tabelle durchzuführen ist, sollte ein Engpaß vermieden werden können. Auch der Ausfall des FE müßte relativ einfach durch einen Ersatz-FE behandelbar sein, der ebenfalls mit allen Terminals und Rechnern verbunden sein muß. Denn die notwendigen Informationen sind im wesentlichen die Routing-Tabelle sowie Daten zur Rechnerauslastung. Wenn man die Routing-Tabelle auf Platte sichert, braucht sie im Fehlerfall nur eingelesen werden; die Informationen zur Rechnerauslastung können bei jedem VAR neu angefordert werden.

d) Dezentrale Lastkontrolle auf mehreren FE
Dieser Ansatz hat gegenüber den vorherigen den Vorteil, daß ein Engpaß noch unwahrscheinlicher wird und daß der Ausfall eines FE reibungsloser überstanden werden kann. Die Engpaßgefahr für einen FE wäre z.B. gegeben, wenn die Lastkontrolle auch Teilaufgaben der DC-Systeme (z.B. Terminalabbildung, Nachrichten-Logging) übernehmen soll. Bei mehreren FE muß jedoch wieder ein ausgezeichneter FE bestimmt werden, der ggf. die Änderung der Routing-Tabelle übernimmt sowie die beobachteten TA-Ankunftsraten zusammenfaßt. Eine nahe Kopplung zwischen den FE kann benutzt werden, um die Routing-Tabelle sowie Informationen zur Auslastung der VAR abzulegen.

Zusammenfassend ist festzuhalten, daß eine sinnvolle Durchführung der Lastkontrolle auf den VAR eine nahe Kopplung mit einem instruktionsadressierbaren Halbleiterspeicher erfordert. Dieser Speichertyp kann jedoch die Verfügbarkeit und Erweiterbarkeit des Systems erheblich beeinträchtigen. Daher ist eine Realisierung mit einem oder mehreren FE vorzuziehen, die jedoch eine effiziente Kommunikation ermöglich müssen. Eine zentrale Lastkontrolle auf einem FE erscheint die attraktivste Lösung zu sein, zumal ein Ausfall relativ einfach behandelbar und ein Engpaß vermeidbar ist. Die Zentralisierung unterstützt die Aufgaben der Lastkontrolle in natürlicher Weise (z.B. braucht die Überwachung der TA-Raten oder die Erstellung einer neuen Routing-Tabelle nicht zwischen mehreren Rechnern koordiniert zu werden). Die DC-Komponente kann weitgehend unverändert gegenüber herkömmlichen TA-Systemen in den VAR ablaufen; die Kommunikation erfolgt statt mit einem Terminal nun mit der Lastkontrolle auf dem FE. Die zentrale Lastkontrolle kann darüberhinaus auch dazu genutzt werden, die Funktionsfähigkeit der VAR zu überwachen und nach einem Ausfall die Recovery-Aktionen anzustoßen und zu koordinieren [Ra85b].

4. Einsatz intelligenter Plattensteuerungen

Zur Durchführung der Platten-E/A besitzt jeder Rechner mehrere Kanäle, die üblicherweise im Blockmultiplex-Modus arbeiten. Die Plattensteuerung (disk controller) ist zwischen Kanal und Platte angesiedelt, um den Kanal von geräteabhängigen Aufgaben, wie Codierung, mechanischer Steuerung oder Energieversorgung zu befreien. Jeder Plattensteuerung sind dabei i.a. mehrere Platten zugeordnet; umgekehrt kann eine Platte auch von mehreren Kontrollern aus erreicht werden, um Wartezeiten zu verringern, wenn eine Steuerung mit einem anderen Laufwerk beschäftigt ist. Ebenso ist eine Steuerung i.a. mit mehreren Kanälen (Rechnern) verbunden.

Vor allem im Bereich der Datenbankmaschinen wurde vielfach versucht, durch leistungsfähigere (intelligentere) Plattensteuerungen, die von den Platten kommenden Daten 'vor Ort' zu filtern. Dabei wird der ankommende Datenstrom hardwaremäßig auf Treffer einer Suchanfrage durchgesehen, wobei dann nur die gefundenen Treffersätze weitergeleitet werden. Im allgemeinen erlauben solche Kontroller (z.B. CAFS von ICL, [Sch85]) jedoch nur relativ einfache Such-

prädikate innerhalb eines Satztyps (keine Joins o.ä.). Für die in DB/DC-Anwendungen ablaufenden TA, die typischerweise sehr einfach sind, haben solche Filterprozessoren nahezu keine Relevanz, da sie v.a. auf das inhaltsorientierte Durchsuchen großer Datenmengen zugeschnitten sind. In Transaktionssystemen dagegen dominieren Einzelsatzzugriffe, die am besten durch vordefinierte Zugriffspfade unterstützt werden.
Eine weitere Einsatzform intelligenter Plattensteuerungen wird im Airline Control Program (ACP) von IBM genutzt. Hier synchronisieren die Plattenkontroller die parallelen Zugriffe auf die Daten (Blöcke) durch die sogenannte 'Limited Lock Facility', wobei über zusätzliche Kanalkommandos Sperren erworben und freigegeben werden können [Ro85]. Es werden jedoch nur exklusive Sperren auf Rechnerebene unterstützt. Für DB-Sharing ist eine Synchronisation in den Plattensteuerungen kaum relevant, da ja versucht wird durch Unterstützung von Lokalität in lokalen bzw. in einem globalen Systempuffer möglichst selten auf die Platte und damit auf den Kontroller zugreifen zu müssen. Außerdem wäre eine vollständige Synchronisation innerhalb der Kontroller ohnehin kaum realisierbar (globale Deadlocks, Transaktionskonzept).
Eine weitere Erweiterung herkömmlicher Plattenkontroller stellt die Verwendung von Platten-Caches dar. Es handelt sich dabei um flüchtige Halbleiterspeicher in den Kontrollern, mit denen das Lesen von Platte beschleunigt werden soll. Wegen der Flüchtigkeit müssen Schreiboperationen jedoch auf der Platte selbst stattfinden. Solche Kontroller werden z.B. von IBM bereits seit einigen Jahren angeboten (Modell 3880/13 bzw. neuerdings 3880/23), wobei die Speicherkapazität des Halbleiterspeichers bis zu 64 MB beträgt. Als Ersetzungsstrategie wird ein LRU-Algorithmus benutzt. Wenn beim Lesen eines Blocks dieser nicht im Platten-Cache gefunden wird, muß er von der Patte gelesen werden. Der Block wird dann in den Hauptspeicher des anfordernden Rechners übertragen und zusätzlich wird die gesamte Spur des Blocks im Platten-Cache gespeichert [Ca85]. Damit wird bei einem sequentiellen Lesen maximal ein Plattenzugriff pro Spur notwendig, da alle weiteren Blöcke aus dem Platten-Cache gelesen werden können (Prefetching). Dadurch sollen IBM zufolge die durchschnittlichen Zugriffszeiten von den üblichen 30-50 ms auf 8-9 ms bei Platten mit einem Platten-Cache reduziert werden.
Die Verwendung solcher Plattenkontroller ist natürlich auch für DB-Sharing von Interesse. Allerdings können die Platten-Caches, im Gegensatz zu den in Kapitel 3 genannten seitenadressierbaren globalen Puffern, nicht sinnvoll zum Austausch von Änderungen benutzt werden. Denn ein Schreiben in den Platten-Cache alleine ist nicht möglich, es wird immer auch auf die Platte geschrieben.
Da in einem DB-Sharing-System jeder Rechner auf jede Platte zugreifen kann, entstehen völlig ungeordnete Zugriffsreihenfolgen, weil jedes Betriebssystem nur die lokalen Zugriffe kennt. Ein Plattenkontroller kann jedoch diese Zugriffe so ordnen, daß die Zugriffsbewegungszeiten auf einer Platte möglichst minimal werden. Solche Optimierungen werden z.B. schon durch die Plattenkontroller bei der Datenbankmaschine IDM/500 von Britton-Lee vorgenommen [Ch84].

5. Zusammenfassung

Im Mittelpunkt dieser Arbeit standen Möglichkeiten einer nahen Rechnerkopplung über gemeinsam benutzte Halbleiterspeicher. Ziel einer solchen Kopplung ist eine verbesserte Leistungsfähigkeit gegenüber lose gekoppelten DB-Sharing-Systemen mit ausreichend hoher Verfügbarkeit. Als Haupteinsatzgebiet von gemeinsamen Speicherbereichen wurden die Synchronisation, die Systempufferkontrolle, die Lastkontrolle und das globale Logging ausgemacht. Unterschieden wurde ferner zwischen flüchtigen und nichtflüchtigen sowie seiten- und instruktionsadressierbaren Speichern. Um eine ausreichend hohe Verfügbarkeit gewährleisten zu können, sollte der Einsatz gemeinsamer Halb-

leiterspeicher jedoch auf wenige Fälle beschränkt bleiben, insbesondere wenn Instruktionsadressierbarkeit erforderlich ist.
In Kapitel 3 wurden die Aufgaben und Realisierungsmöglichkeiten der Lastkontrolle, einer der wesentlichen Komponenten in einem DB-Sharing-System, untersucht. Diese Funktion kann zentral oder dezentral sowie auf den Verarbeitungsrechnern oder auf Front-End-Rechnern durchgeführt werden. Am geeignetsten erscheint eine Durchführung der Lastkontrolle auf einem Front-End, der sehr schnell mit den Terminals und den Verarbeitungsrechnern kommunizieren kann.
Im letzten Kapitel wurden noch bisher bekanntgewordene Verwendungsformen von (mehr oder weniger) intelligenten Plattenkontrollern auf ihre Eignung für DB-Sharing betrachtet. Als sinnvoll ist dabei eine globale Koordinierung der Plattenzugriffe durch den Kontroller zur Minimierung des Positionierungsaufwandes anzusehen; ebenso die Nutzung von Platten-Caches zur Beschleunigung von Lesezugriffen, wenngleich dieser Vorteil bei Verwendung von großen Puffern sowie eines globalen Systempuffers verringert werden dürfte.
Die Bewertung der vorgestellten Einsatzmöglichkeiten einer nahen Kopplung wird vor allem dadurch erschwert, da das Leistungs- und Ausfallverhalten der einzelnen Speichertypen noch nicht ausreichend bekannt sind (z.B. Zugriffszeiten, CPU-Kosten pro Zugriff, Auswirkungen eines Rechnerausfalles). Weiterhin ist unklar, wieviele Rechner tatsächlich solche Speicher gemeinsam nutzen können (wichtig für die Erweiterbarkeit des Systems), wo die technischen Grenzen liegen und welche Kostenentwicklung zu erwarten ist. Trotz dieser und anderer offener Punkte wird in Bild 6 ein Architekturvorschlag für ein DB-Sharing-System gezeigt, bei dem etwas mehr Details als in Bild 1a aufgenommen wurden.

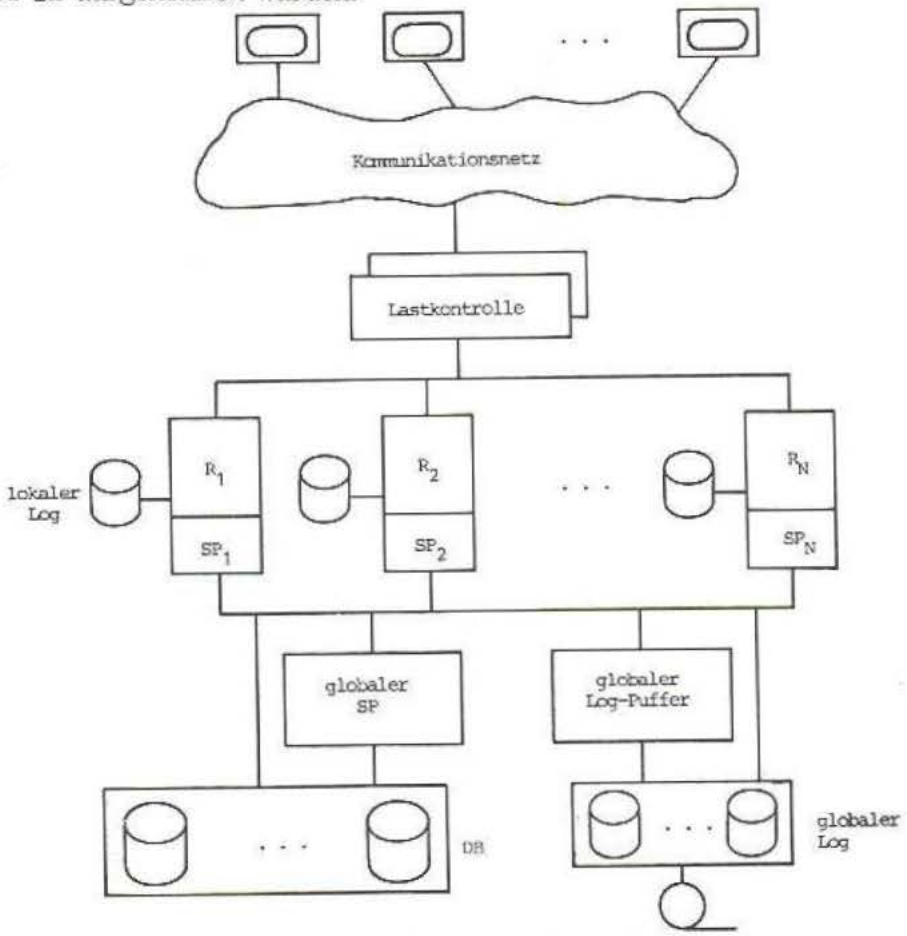


Bild 6: Architekturvorschlag für ein DB-Sharing-System

Wegen der zu erwartenden Verfügbarkeitsprobleme wurde auf eine nahe Kopplung mit instruktionsadressierbaren Speicherbereichen verzichtet. Zur Abwicklung der Lastkontrolle wurde gemäß Bild 3 ein Front-End sowie sein Stand-by eingebaut. Weiterhin wurde ein globaler Log-Puffer, für den Nichtflüchtigkeit unterstellt wird, aufgenommen, um das globale Logging effizient durchführen zu können (vergl. Kapitel 2). Nicht explizit gezeigt wurde die Instanz zur Verwaltung des globalen Log-Puffers. Ein globaler Systempuffer erscheint sinnvoll, da er eine Reduzierung der Plattenzugriffe sowie einen effizienten Austausch geänderter Seiten erlaubt. Falls Nichtflüchtigkeit vorliegt, werden zudem Ausschreibvorgänge aus den lokalen Systempuffern erheblich beschleunigt (besonders wichtig bei einer FORCE-Ausschreibstrategie). Weiterhin können bei Nichtflüchtigkeit systemweit Änderungen im globalen Systempuffer akkumuliert werden.

6. Literaturverzeichnis

Ca85 3880 Cache Control Units, Part 1, in: Candle Computer Report, Vol. 7, No. 11, Juni 1985, S. 1-2.

Ch84 Christmann, H.P.: Rechnerarchitekturen zur Unterstützung von nichtkommerziellen Anwendungen, Forschungsbericht SFB 124, Nr. 08/84, Univ. Kaiserslautern, Mai 1984.

Hä85 Härder, T.: DB-Sharing vs. DB-Distribution - die Frage nach dem Systemkonzept zukünftiger DB/DC-Systeme, FB Informatik, Univ. Kaiserslautern, 1985.

HM85 Härder, T., Meyer-Wegener, K.: Transaktionssysteme, TP-Monitore, DB/DC-Systeme - Eine Einführung, Interner Bericht, FB Informatik, Univ. Kaiserslautern, 1985.

HR83 Härder, T., Reuter, A.: Principles of Transaction-Oriented Database Recovery, in: ACM Computing Surveys, Vol. 15, No. 4, Dez. 1983, S. 287-317.

HR85a Härder, T., Rahm, E.: Quantitative Analyse eines Synchronisationsalgorithmus für DB-Sharing, in: Proc. 3. GI/NTG-Fachtagung "Messung, Modellierung und Bewertung von Rechensystemen", IFB 110, Dortmund, Okt. 1985, S. 186-201.

HR85b Härder, T., Rahm, E.: Klassifikation und Eigenschaften von Mehrrechner-Datenbanksystemen, FB Informatik, Univ. Kaiserslautern, 1985.

Ki84 Kim, W.: Highly Available Systems for Database Applications, in: ACM Computing Surveys, Vol. 16, No. 1, März 1984, S. 71-98.

Ra84 Rahm, E.: Quantitative Analyse eines Synchronisationsprotokolls für Mehrrechner-Datenbanksysteme, Diplomarbeit, FB Informatik, Univ. Kaiserslautern, Nov. 1984.

Ra85a Rahm, E.: Primary Copy Synchronization for DB-Sharing, Interner Bericht 137/85, FB Informatik, Univ. Kaiserslautern, Juli 1985.

Ra85b Rahm, E.: A Reliable and Efficient Synchronization Protocol for DB-Sharing, IB 139/85, FB Informatik, Univ. Kaiserslautern, August 1985.

Re85 Reuter, A.: Load Control and Load Balancing in a Shared Database Management System, IB 129/85, FB Informatik, Univ. Kaiserslautern, März 1985.

RS84 Reuter, A., Shoens, K.: Synchronization in a Data Sharing Environment, IBM San Jose Research Laboratory, August 1984 (vorläufige Fassung).

Ro85 Robinson, J.T.: A Fast General-Purpose Hardware Synchronization Mechanism, in: Proc. ACM SIGMOD 1985, S. 122-129.

Sch85 Schweppe, H.: Hardwareunterstützung für Datenbanken in Büro, Technik und Wissenschaft, in: Proc. GI-Fachtagung "Datenbanksysteme für Büro, Technik und Wissenschaft", IFB 94, Karlsruhe, März 1985, S. 287-308.

Diese Arbeit wurde von der SIEMENS AG finanziell unterstützt.