Quantitative Analyse eines Synchronisationsalgorithmus für DB-Sharing

T. Härder, E. Rahm
Universität Kaiserslautern

Abstract
Als DB-Sharing bezeichnet man die gemeinsame Benutzung einer Datenbank durch Datenbank-Verwaltungssysteme auf einem System lose 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.
Nach einer Beschreibung des Aufbaus und der funktionellen Komponenten eines DB-Sharing-Systems werden seine speziellen Charakteristika für den Transaktionsbetrieb diskutiert. Daraus ergeben sich die wesentlichen Aspekte, die bei der Modellbildung eines Mehrrechner-Datenbanksystems zu berücksichtigen sind. Für ein DB-Sharing mit zwei Rechnern wird ein Synchronisationsprotokoll auf der Basis eines Token-Ring-Konzeptes vorgestellt, welches auch das sogenannte Veralterungsproblem löst. Mit auf Seitenreferenzstrings basierenden Simulationen konnte die Leistungsfähigkeit des Verfahrens quantifiziert werden. Bei des Meßläufen kamen einfache Maßnahmen zur Lastkontrolle zur Anwendung.

1. Einleitung

Bestehende DBS-Anwendungen in klassischen Bereichen wie Bankwesen, Lagerhaltung, Buchhaltung usw., die im Rahmen von Transaktionssystemen zu Auskunfts-, Buchungs- und Datenerfassungsaufgaben [HM85] eingesetzt werden, verlangen zunehmend die Erweiterung und Verbesserung existierender DBVS oder gar die Neuentwicklungen spezieller Systeme, um vor allem folgende Anforderungen erfüllen zu können:

- Hohe Transaktionsraten, die möglicherweise auf relativ kleinen Datenbanken abzuwickeln sind:
  Dieser Aspekt zielt auf Performance-Steigerungen des DBVS bei typischerweise kurzen und vorgeplanten Transaktionen des gleichen oder verschiedenen Typs ab, die hochgradig parallel auszuführen sind. Datenbanksysteme, die diese Art des Transaktionsbetriebs in optimaler Weise unterstützen, werden oft als **Hochleistungs-Datenbanksysteme** bezeichnet. Als Beispiel diene IMS/Fast Path, das auf einem 10-12 MIPS-Rechner bis zu 180 TA/sec vom Typ "Kontenbuchung" [Gr78] durchführen kann. 1000 TA/sec sind momentan konkretes Entwicklungsziel [Gr85].
- Hohe Verfügbarkeit:
  Das DBS muß während seiner gesamten Betriebszeit - im Extremfall bis zu 24 Stunden an 7 Tagen in der Woche - verfügbar sein. Je mehr Transaktionen in einem Transaktionssystem online abgewickelt werden, umso stärker macht sich ein Systemausfall als Störung im Betriebsablauf eines Unternehmens bemerkbar. Der Restart-Aufwand der Transaktionsanwendung hängt linear von der Anzahl der angeschlossenen Terminale ab. Bei einigen tausend Terminalen ist deshalb mit einer Restartzeit zu rechnen, die mehrere Stunden betragen kann. In vielen Anwendungen (z.B. Ein-/Auszahlung auf Konten, Fahrkartenverkauf usw.) sind jedoch höchstens 5-10 Minuten als Ausfallzeit zu tolerieren.

Um solche DBVS-Eigenschaften gewährleisten zu können, ist eine Systemstruktur zu wählen, die einerseits eine modulare Steigerung der Rechnerleistung zuläßt und andererseits geeignete Redundanzen zur Erhöhung der Fehlertoleranz besitzt. Einprozessorsysteme erbringen weder die geforderte Leistungsfähigkeit noch die notwendige Redundanz. Um den Ausfall eines Prozessors überstehen zu können, muß die Transaktionsanwendung auf einem Mehrprozessorsystem ablaufen. Eng gekoppelte Mehrprozessorsysteme (tightly coupled) erfüllen die Forderung nach erhöhter Rechnerleistung nur in begrenztem Maße (2-4 Prozessoren). Problematisch ist eine solche Systemstruktur aber vor allem unter dem Aspekt der Fehlertoleranz.
Die Realisierung der oben aufgestellten Entwurfsziele legt eine stärkere Separierung und Isolierung der Prozessoren und Speicher nahe. Die dafür geeigneten Systemstrukturen, die eine weniger enge Kopplung (closely und loosely coupled)

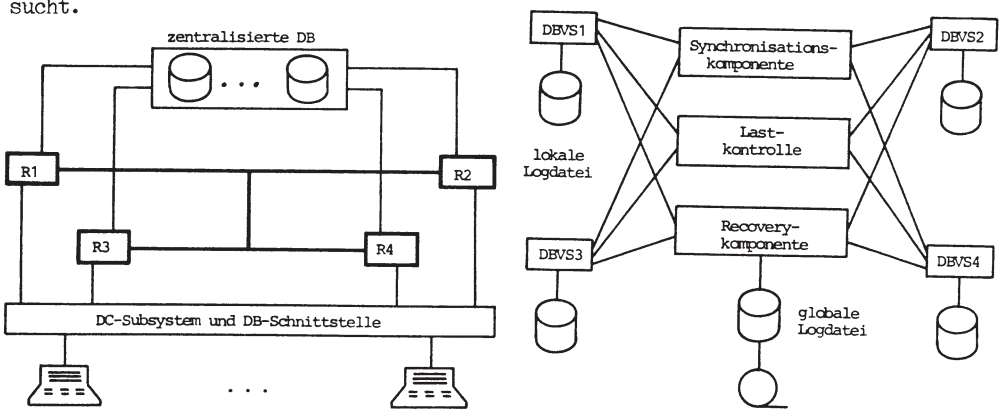
aufweisen, werden in [Ki84] und [HR85] untersucht. Für unseren Systemvorschlag betrachten wir ein lose gekoppeltes Mehrrechnersystem, in dem n Prozessoren mit eigenen Kopien des Betriebssystems und separaten Hauptspeichern mit Hilfe von Nachrichten über eine schnelle Kommunikationsverbindung zusammenarbeiten. Die einzusetzenden Protokolle und Algorithmen sollen eine Erhöhung der Fehlertoleranz und damit der Verfügbarkeit gewährleisten und ein modulares Wachstum des Systems unterstützen. Die größere Isolation zwischen den Komponenten erfordert jedoch einen höheren Kommunikationsaufwand für Kooperation und Synchronisation.

2. Konzept für ein Mehrrechner-DBS

Für die DB-Verwaltung in einem lose gekoppelten Mehrrechnersystem ist ein Mehrrechner-DBS zu entwickeln, das die verfügbaren Ressourcen und Fehlertoleranzmöglichkeiten optimal ausnutzt. Nach diesem Konzept läuft auf jedem Rechner ein selbständiges DBVS ab, das in steter Koordination und Kommunikation mit den übrigen DBVS stehen muß. Alle DBVS müssen gemeinsam die Daten der DB verarbeiten. Zwei Möglichkeiten der physischen Zuordnung der Daten sind 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 erfolgsversprechenden Ansatz, der weiter betrachtet werden soll. Seine prinzipielle Systemstruktur ist in Bild 1a gezeigt. Allgemeine Kriterien und Unterscheidungsmerkmale zu DB-Distribution werden in [HR85] untersucht.



a.) Systemstruktur b.) funktionelle Komponenten

Bild 1: Aufbau eines DB-Sharing-Systems

Einheit der Aufgabenverteilung ist die Transaktion. Sie wird vom DC-System einem Rechner (DBVS) zur Bearbeitung zugeteilt. Falls kein Fehler auftritt, wird die Transaktion (TA) durch das ausgewählte DBVS vollständig abgewickelt; im Fehlerfall wird sie nach ihrem Zurücksetzen von einem anderen DBVS erneut gestartet. Zur TA-Bearbeitung sind alle dazu notwendigen Betriebsmittel zu erwerben und insbesondere die erforderlichen Datenseiten in den lokalen Systempuffer des DBVS zu holen. Als wesentliche DBVS-Anforderungen [RS84], die neuartige Lösungen verlangen, treten dabei

- die **Synchronisation** beim Zugriff der verschiedenen DBVS auf die gemeinsamen Daten
- das **Logging** lokaler und globaler Daten und die **Recovery** beim Ausfall eines DBVS
- die **Lastkontrolle** zur optimalen Zuordnung und Verteilung von Transaktionen zu Prozessoren

auf. Diese Aufgaben, die alle zentral oder verteilt realisiert werden können, sind in Bild 1b noch einmal durch Zuordnung zu den Systemkomponenten verdeutlicht.

3. Modellbildung für ein Mehrrechner-DBS

Beim Entwurf eines solchen Systems ist die Bestimmung seiner (zu erwartenden) Leistungsfähigkeit von besonderer Wichtigkeit. Insbesondere sind **genaue Aussagen über Durchsatz und Antwortzeit** entscheidend, da es den Transaktionsbetrieb unterstützen soll:

- Der Durchsatz gemessen in TA/sec soll idealerweise linear mit der Anzahl der beteiligten Rechner wachsen ("modulares" Wachstum auch in Bezug auf Leistung). Für Koordinationsaufgaben ist natürlich ein gewisser Aufwand (10 - 20 %) in Kauf zu nehmen.
- Die Beibehaltung der Antwortzeit für eine TA - verglichen zu einer Lösung mit einem zentralen DBVS - ist wesentlich für die Akzeptanz des Systems. Ihr Anstieg um mehr als 10 % [RS84] würde in vielen Einsatzumgebungen nicht mehr toleriert werden. Der Grund dafür ist nicht nur die längere Wartezeit am Terminal. Wenn die Antwortzeit stark ansteigt, muß zur Erhaltung des Durchsatzes die Parallelität erhöht werden. Das bedeutet aber, daß "von außen her" mehr TA gestartet werden müssen, was mehr Terminals und mehr Personal zur Bewältigung der vorgegebenen Aufgabe impliziert.

Bei der genauen Modellierung des Mehrrechner-DBS zur Beantwortung der Fragen nach seiner Leistungsfähigkeit **spielt die Synchronisation eine kritische Rolle.** Sie besitzt einen dominierenden Einfluß auf Durchsatz und Antwortzeit und bestimmt die Struktur des Systemmodells. Da jeder Zugriff auf ein Datum (Datenseite) synchronisiert werden muß, kann bei der Modellierung gleichzeitig die Verarbeitung der Daten (Verarbeitungskosten und ggf. E/A-Kosten) und das Logging redundanter Informationen berücksichtigt werden. Da in jedem Fall eine gewisse Zuordnung und Regulierung der Last vorgenommen werden muß, sind **zumindest einfache Verfahren der Lastkontrolle** vorzusehen. Es ist klar, daß zunächst nur das Verhalten des Systems im Normalbetrieb interessiert und daß deshalb bei der Modellbildung die Fragen der Recovery mit ihren vielen Sonderfällen und die damit verbundenen Probleme der Lastverteilung vernachlässigt werden können.

Warum ist die Synchronisation für die Leistungsfähigkeit so entscheidend? In DBVS werden typischerweise Sperrverfahren eingesetzt. Im zentralen Fall ist das DBVS durch einen oder mehrere Prozesse mit einer gemeinsamen Speicherpartition repräsentiert. Die gewährten Sperren werden durch eine über eine Hashfunktion adressierbare Datenstruktur im Hauptspeicher dargestellt und durch den Sperrverwalter geführt. Die Überprüfung, Freigabe oder Gewährung einer Sperre (wenn nicht gewartet werden muß) kosten 200 bis 700 Maschineninstruktionen [RS84]. Da eine typische kurze TA 10 bis 50 Sperren anfordert und freigibt, erzeugen 20 TA/sec schon bis zu 1000 Wartungsoperationen pro sec. auf der Sperrtabelle.

In einem DB-Sharing-System ist das DBVS auf jedem Rechner durch einen oder mehrere Prozesse repräsentiert. Kommunikation zwischen Rechnern kann nur über Nachrichten stattfinden. Bei Synchronisation über Sperren sind folgende Aspekte zusätzlich zu beachten:

- Eine Sperranforderung, über die nicht lokal entschieden werden kann, impliziert eine oder mehrere Nachrichten zur Synchronisation. Die anfallende Wartezeit verlängert die Antwortzeit der TA.
- Eine solche nur "global" entscheidbare Sperranforderung bedeutet eine Deaktivierung der TA (synchroner Aufruf als Nachricht) wegen der relativ langen Wartezeit. Die Deaktivierung und spätere Aktivierung der TA verlangen je nach Betriebssystem und Prozeßstruktur 2 Task- oder Prozeßwechsel, wobei ein Prozeßwechsel bis zu 5000 Instruktionen bedingen kann. Zusätzlich müßte zur Erhaltung des Durchsatzes der Grad der internen Parallelität (= der TA) erhöht werden, was wiederum zu Lasten der Antwortzeit einer TA geht (größere Blockierungsgefahr, geringerer Rechenzeitanteil).

Aus diesen Überlegungen kann klar der Schluß gezogen werden, daß bei einem Synchronisationsalgorithmus für DB-Sharing unbedingt die **Anzahl der globalen Sperranforderungen zu minimieren** ist. Dieses Ziel läßt sich sicher nur in enger Zusammenarbeit mit der Lastkontrolle erreichen; es muß nämlich sichergestellt sein, daß neu ankommende TA dorthin geleitet werden, wo (mit großer Wahrscheinlichkeit) die Betriebsmittel - vor allem Sperren und Datenseiten - vorhanden sind, die zu ihrer Bearbeitung benötigt werden. Auf diese Weise läßt sich eine Art

der Lokalität (Affinität) der Verarbeitung gewinnen, die leistungsmindernde Thrashing-Situationen verhindern hilft.
Das sogenannte **Veralterungsproblem** von Seiten wird ebenfalls durch die Lokalität der Verarbeitung entschärft. Es tritt bei DB-Sharing aus folgenden Gründen auf:

- Seiten können sich in den Systempuffern verschiedener DBVS (Rechner) befinden und parallel gelesen werden. Bei großer Referenzhäufigkeit versucht der Pufferverwalter solche Seiten möglichst lange im Puffer zu halten, selbst wenn momentan keine Anforderung der Seite vorliegt.
- Wird eine solche Seite von einem DBVS geändert, so sind die Kopien der Seite in den anderen Puffer veraltet (buffer invalidation). Zur Vermeidung von inkonsistenten Operationen ist eine Mitteilung an die übrigen DBVS erforderlich.
- Bei einer erneuten Referenz muß die aktuelle Version der Seite zur Verfügung gestellt werden. Das impliziert, daß die geänderte Seite entweder zum anfordernden Rechner übertragen wird oder daß sie zwangsweise auf die DB ausgeschrieben wird, damit sie vom anfordernden Rechner erneut eingelesen werden kann.
- Bei direkter Übertragung einer aktuellen Version muß darauf geachtet werden, daß beim späteren Ausschreiben nicht eventuell eine neuere Version der Seite überschrieben wird.

Der Synchronisationsalgorithmus und die gewählte Ausschreibstrategie (FORCE/¬FORCE [HR83]) üben einen wesentlichen Einfluß auf die skizzierte Problemstellung aus. Die Integration von Maßnahmen zur Erkennung der Veralterung oder zu ihrer Vermeidung im Rahmen der Synchronisation sollte zur Leistungssteigerung der Mehrrechner-DBS beitragen, wenn Lokalität ausgenutzt werden kann.

In diesem Aufsatz wird ein DB-Sharing-Modell untersucht, das auf einem Synchronisationsalgorithmus auf der Basis eines **Token-Ring-Konzeptes** beruht. Die Grundidee dieses Algorithmus wurde bereits für "Data Sharing" in IMS/VS 1.2 für ein hierarchisches DB-Konzept [IMS81, Ke81, SUW82] ausgenutzt und implementiert. Wir schneiden jedoch diesen Algorithmus auf die Anforderungen von netzwerkartigen DB-Konzepten [CODA73] zu und optimieren ihn auf die besonderen Bedürfnisse von UDS [UDS84]. Dazu gehört vor allem die **Berücksichtigung einer Sperrhierarchie, spezieller Seitentypen sowie von Maßnahmen zur Ausnutzung von Lokalität.**
Der ursprüngliche Algorithmus ist für 2 lose gekoppelte Rechner entworfen, wobei jeder Rechner wiederum ein eng gekoppeltes Mehrprozessorsystem darstellen kann. Wir untersuchen den Algorithmus für diesen wichtigen Spezialfall, der sich schon als sehr komplex erweist. Obwohl eine Erweiterung prinzipiell möglich ist, wird sie aus Gründen der Komplexität nicht weiter verfolgt.

Die Art der Modellbildung hat entscheidenden Einfluß auf die Genauigkeit der Ergebnisse. Da eine zuverlässige Bestimmung von Durchsatz und Antwortzeit gefordert wird, scheiden analytische Modelle aus. Probabilistische Modelle zur Darstellung der TA-Last und der referenzierten DB-Daten sind **bei allen DB-Anwendungen sehr ungenau** und damit unbrauchbar. Durch die Verwendung von Seitenreferenzstrings verschiedenen Typs, die aus einer Reihe realer DB-Anwendungen gewonnen wurden, gelingt eine verläßliche Darstellung der TA-Last und ihrer DB-Referenzen. Ein aus einem TA-Mix abgeleiteter Seitenreferenzstring verkörpert deshalb **das aus TA bestehende Lastmodell und zugleich das Referenzmodell für die DB.** Durch eine referenzstringgetriebene Simulation, bei der die wichtigsten Systemkomponenten nachgebildet werden, sind deshalb recht genaue Leistungsdaten für das DB-Sharing-System zu erwarten.

4. Sperrprotokoll für ein DB-Sharing mit zwei Rechnern

In diesem Abschnitt wird ein Verfahren zur Synchronisation in einem DB-Sharing-System mit zwei Rechnern angegeben, wobei auf jedem Rechner ein CODASYL-artiges DBS eingesetzt wird. Eine Lösung des Veralterungsproblems wird über sogenannte Haltesperren in das Konzept integriert. Besonders berücksichtigt wird der Spezialfall, bei dem nur einer der beiden Rechner Update-TA ausführen darf. Die **Grundidee** des Verfahrens besteht darin, daß die beiden Rechner abwechselnd 'Master' des Systems sind. Jeder Rechner darf nur als Master globale Sperrinformationen, die von beiden Prozessoren geführt werden, modifizieren. Am Ende einer Masterphase, deren Dauer bei Systemstart eingestellt werden kann, wird ein sogenannter **Buck** zum anderen Rechner übertragen. Er enthält

- Änderungen an den globalen Sperrinformationen, die in der zu Ende gegangenen Masterphase vorgenommen wurden
- Sperranforderungen (Lock-Requests), die lokal nicht entscheidbar sind, und Antworten (Lock-Responses) auf solche für den anderen Rechner
- sonstige Nachrichten, wie Kommandos von globaler Bedeutung u.ä.

Der Empfang eines Bucks signalisiert dem Rechner den Beginn seiner Masterphase. Die Rechner befinden sich somit zyklisch in folgenden Zeitabschnitten:
- Slavephase
- Empfangen des Bucks
- Verarbeiten des Bucks
- Masterphase
- Senden des Bucks
- Slavephase usw.

Eines der Hauptziele bei der Entwicklung des Sperrkonzeptes war die Minimierung der Sperranforderungen an den anderen Rechner, da sie wegen der Notwendigkeit des synchronen Wartens die Antwortzeit der TA entscheidend beeinflußen. Zu den Wartezeiten kommen zudem noch die Verzögerungen bis zur aktuellen Buckübertragung, da die Nachrichten immer nur im Buck übertragen werden (Eine ´Bündelung´ von Nachrichten ist schon aus Kostengründen unabdingbar).

Die Minimierung der Sperranforderungen kann zum Teil durch Verwendung **hierarchischer Sperren** erreicht werden. Sinnvolle Granulate wären dabei etwa
- Datenbank
- Area
- Seite (Block)
- Eintrag (z.B. für Hot-Spot-Objekte).

Wenn z.B. nur ein Rechner Interesse (sole interest) an Area A hat, können alle Zugriffe auf Seiten von A durch Vergabe einer hierarchischen Sperre auf A wie im zentralisierten Fall (ohne Kommunikation) synchronisiert werden. Im vorliegenden Konzept wurde eine Synchronisation auf Area- und auf Blockebene vorgesehen. Die Behandlung des Veralterungsproblems wird auf Blockebene ´miterledigt´.

Im folgenden wird das Protokoll genauer vorgestellt, das im wesentlichen durch die Aktionen des Sperrverwalters bei READY und FINISH (Areaebene), bei LOCK und UNLOCK (Blockebene) sowie bei Abarbeitung der Nachrichten im Buck festgelegt ist. Aus Platzgründen müssen jedoch bei der Beschreibung Abstriche in der Vollständigkeit und Genauigkeit in Kauf genommen werden. Die genauen Algorithmen sind in [Ra84] zu finden.

4.1 Synchronisation auf Areaebene

Bei CODASYL-artigen DBS gibt eine TA beim READY an, auf welche Areas des Subschemas und mit welchem Modus sie zugreifen will. Die Synchronisation auf Areaebene geschieht über eine Areatabelle, die in jedem Rechner geführt wird und deren Größe durch die maximal mögliche Anzahl von Areas begrenzt ist. Wenn der Sperrverwalter mittels der Areatabelle feststellt, daß ein READY mit den bereits zugelassenen TA nicht verträglich ist, wird die zugehörige TA deaktiviert und in eine areaspezifische Warteliste gebracht. Das Verfahren ist so konzipiert, daß auf Areaebene nur Nachrichten zur Aktualisierung der Areatabelle benötigt werden. Somit kann ein READY ohne expliziten Nachrichtenaustausch bearbeitet werden; es ist höchstens ein Warten bis zur nächsten Masterphase notwendig, um die Aktualität der Areatabelle zu gewährleisten.

In der Areatabelle wird für jede Area vermerkt, welche Rechner "Interesse" darauf haben, wie der höchste Areamodus aller zugelassenen TA ("globaler Zustand") lautet sowie welche TA zugelassen bzw. vorerst abgewiesen wurden. Auf Areas von gemeinsamem Interesse werden dabei auch TA des anderen Rechners in der Warteliste aufgenommen, um eine faire Bedienung der Warter zu ermöglichen (Vermeidung des Starvation-Problems). Die Abarbeitung einer Warteliste erfolgt durch den Rechner, bei dem durch das FINISH einer TA eine Verringerung des globalen Zustandes eintritt, so daß wartende TA zugelassen werden können.

4.2 Synchronisation auf Blockebene

Hierzu verwaltet jeder Rechner eine Sperrtabelle (**Global Lock Table:** GLT), in der die Sperrkontrollblöcke (Blockeinträge) über eine Hashfunktion eingebracht werden, sowie eine **Global Hash Table** (GHT) mit einer gleichen Anzahl von Einträgen (Bild 2). In der GLT sind zu jeder Hashklasse die Blockeinträge, die in diese Klasse fallen, untereinander verkettet. In der GHT werden für jede Hashklasse zwei Bit geführt, wobei Bit 1 (Bit 2) angibt, ob Rechner 1 (Rechner 2) Interesse auf einem Objekt der Hashklasse hat oder nicht. Interesse bedeutet dabei, der Rechner ist in Besitz einer Sperre bzw. verlangt eine Sperre zu einem Block der Hashklasse.

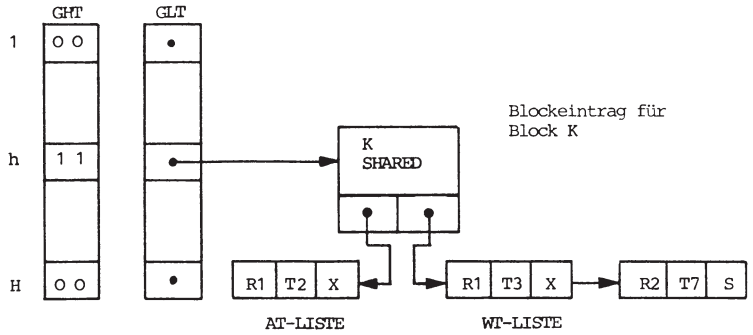


Bild 2: GHT und GLT (Beispiel für Rechner 1)

Eine GHT wurde bereits beim Data Sharing in IMS eingesetzt und bildete dort die einzige Maßnahme, Sperranforderungen an den anderen Rechner einzusparen. Wenn nämlich eine Sperranforderung in Rechner 1 einen GHT-Eintrag 10 vorfindet (bzw. 00 und der Rechner ist Master), dann kann die Synchronisation lokal erfolgen, da ein Konflikt mit Rechner 2 ausgeschlossen ist. Nur für einen GHT-Eintrag 01 bzw. 11 muß die Sperranforderung im Buck weitergeleitet werden. Änderungen an der GHT kann nur der jeweilige Master vornehmen; er teilt sie dem anderen Rechner im Buck mit.
Da durch die GHT allein noch viele lange Sperranforderungen anfallen, wurden im vorliegenden Konzept auch in den Blockeinträgen globale Sperrinformationen aufgenommen. Ein Blockeintrag enthält u.a. folgende Informationen:

```
SHARED : Boolean; (* gibt an, ob beide Rechner auf dem Block Interesse haben *)
HS, HX : Boolean;  (* Haltesperren *)
AT-Liste, WT-Liste : verkettete Liste von TA-Einträgen. (* ein TA-Eintrag besteht
     aus der Rechner- und der TA-Identifikation sowie dem Sperrmodus (S oder X) *)
```

In der AT-Liste stehen solche TA, denen eine Sperre auf dem Block gewährt wurde; in der WT-Liste stehen lokale und externe TA, deren Sperranforderung abgelehnt wurde. Da die SHARED-Information wesentlich genauer ist als die GHT-Angaben für ganze Hashklassen, wird die GHT nur benutzt, falls noch kein Blockeintrag vorhanden ist, um ggf. die Sperre ohne Kommunikation gewähren zu können.

Einsatz von Haltesperren

Durch die Haltesperren wird nun erreicht, daß die Blockeinträge länger gehalten bleiben und damit die genauere Information entsprechend länger zur Verfügung steht. Haltesperren, mit denen ja in erster Linie das Veralterungsproblem gelöst werden soll, sind dabei keine Sperren im üblichen Sinn, die von TA erworben und freigegeben werden. Sie beschreiben vielmehr den Zustand des Blocks bzgl. eines Rechners. Es werden zwei Arten von Haltesperren unterschieden, wobei zu einem Zeitpunkt pro Rechner und Block höchstens eine existieren darf:

- Eine **HS-Sperre** wird gesetzt, wenn die letzte lokale S-Sperre freigegeben wird und keine TA auf eine Sperre wartet (Warteliste leer). Beide Rechner können gleichzeitig eine HS-Sperre auf demselben Block haben. Eine HS-Sperre bleibt nur gesetzt, wenn keine TA auf den Block wartet und keine lokale TA eine Sperre auf dem Block besitzt.

- Eine **HX-Sperre** wird nach Freigabe einer X-Sperre gesetzt, falls danach keine externe TA aktivierbar wird. Eine HX-Sperre kann nur in einem der beiden Rechner existieren und zwar in dem, der die letzte X-Sperre auf dem Block hatte und daher auch die letzte Änderung durchgeführt hat. Die Existenz einer HX-Sperre bedeutet, daß der Block mit Änderungsvermerk im Puffer steht oder bereits auf Platte ausgeschrieben wurde. Die HX-Sperre darf nicht aufgegeben werden, solange die geänderte Seite noch nicht in die Datenbank eingebracht wurde.

Die folgende Matrix zeigt die Verträglichkeit der Sperren auf einem Block zwischen zwei Rechnern:

	S	HS	X	HX
S	+	+	-	-
HS	+	+	-	-
X	-	-	-	-
HX	-	-	-	-

Mit Haltesperren kann das Veralterungsproblem behandelt werden, da sie sicherstellen, daß die neueste Version eines Blockes nicht vom anderen Rechner geholt werden muß. Es werden im folgenden der kompliziertere Fall einer ¬**FORCE**-Ersetzungsstrategie (kein Einbringen einer geänderten Seite bei EOT) sowie die Existenz eines Blockeintrages für jede im Puffer befindliche Seite angenommen. Das Verfahren funktioniert unabhängig davon, ob geänderte Seiten direkt zwischen den Rechnern oder über Platte ausgetauscht werden. Wenn nämlich bei Anforderung einer Sperre ein Blockeintrag existiert, dann befindet sich die aktuelle Version des Blockes meist im lokalen Puffer oder auf Platte (dies ist immer dann gewährleistet, wenn eine lokale TA eine Sperre besitzt oder eine Haltesperre gesetzt ist). Existiert kein Blockeintrag, dann trifft dies auch zu, wenn aufgrund der GHT festgestellt wird, daß der andere Rechner kein Interesse an der Hashklasse (also keinen Blockeintrag führt und damit nach Voraussetzung auch die Seite nicht in seinem Puffer hat) besitzt.

Erfordert der GHT-Eintrag einen LOCK-REQUEST an den anderen Rechner, so wird dort die Verträglichkeit mit lokalen TA überprüft. Ist die Sperranforderung zulässig und befindet sich die Seite noch mit Änderungsvermerk im Puffer (in diesem Fall muß HX gesetzt sein), so wird die Seite entweder direkt (mit der LOCK-RESPONSE) oder über Platte ausgetauscht. In letzterem Fall muß dann der andere Rechner nach Eintreffen der LOCK-RESPONSE die Seite einlesen.
Ist der LOCK-REQUEST nicht zulässig, so kommt die TA in die Warteliste zu dem Block, für den jetzt SHARED = true gilt, und es wird eine ablehnende LOCK-RESPONSE übertragen. Wenn dann die geforderte Sperre zu einem späteren Zeitpunkt zuerkannt werden soll, dann muß ggf. auch vorher noch die neueste Version der Seite in den Rechner gebracht werden.

Neben der Lösung des Veralterungsproblems unterstützen die Haltesperren auch Lokalität in den Sperranforderungen eines Rechners, da die Blockeinträge länger gehalten werden. Es können nämlich viele Sperranforderungen aufgrund der Informationen im Sperrkontrollblock ohne Kommunikation mit dem anderen Rechner gewährt werden:
- Falls auf einem Block HS gesetzt ist, kann eine S-Sperre gewährt werden, da dann keine TA warten und auf dem anderen Rechner keine X-Sperre gewährt ist. Eine X-Sperre kann bei HS nur gewährt werden, wenn der Block entweder in lokalem Besitz ist oder aber der andere Rechner auch eine HS-Sperre hat und der eigene Rechner Master ist. In letzterem Fall wird dem anderen Rechner durch eine RELEASE-AUFFORDERUNG mitgeteilt, seinen Blockeintrag aufzugeben.
- Eine HX-Sperre stellt sicher, daß auf dem Block keine externe TA eine Sperre besitzt. Eine (S- oder X-)Sperre kann also sofort zugelassen werden, wenn sie mit lokal gewährten Sperren verträglich ist und die Warteliste leer ist. Im Gegensatz zur HS-Sperre bleibt HX auch nach Gewährung lokaler Sperren gesetzt, da sonst die Information, daß die letzte Änderung der Seite in diesem Rechner stattfand (und die gültige Version möglicherweise noch nicht auf Platte ist), verlorenginge. Eine HX-Sperre muß erst dann aufgegeben werden, wenn dem anderen Rechner eine Sperre auf dem Block gewährt werden soll.

Aktionen bei einer Lock-Bearbeitung
Wenn eine TA T eine Sperre für einen Block B benötigt, richtet sie einen LOCK-Aufruf an den lokalen Sperrverwalter. Dieser überprüft zunächst, ob ein Blockeintrag für B in der GLT vorhanden ist oder nicht. Wenn noch kein Blockeintrag existiert, kann die Sperre dennoch lokal gewährt werden (möglicherweise nach einem Warten bis zur nächsten Masterphase), wenn die GHT Konflikte im anderen Rechner ausschließt (s.o.) oder wenn der andere Rechner kein Interesse an der Area, zu der B gehört, hat. In diesen Fällen wird ein Blockeintrag für B angelegt, wobei die TA T in die AT-Liste gebracht wird und SHARED = false gesetzt wird. Kann die Sperre nicht lokal gewährt werden, wird eine LOCK-REQUEST-Nachricht zum anderen Rechner geschickt, auf deren Antwort die TA in der Warteliste des neu angelegten Blockeintrags wartet.

War zum Zeitpunkt des LOCK-Aufrufs bereits ein Blockeintrag vorhanden, so kann die Sperranforderung stets lokal behandelt werden. Für SHARED = true muß der Rechner dazu i.a. Master sein, während bei nur lokalem Interesse an dem Block immer sofort über die Zulässigkeit der Sperre entschieden werden kann. Auf einem Block mit SHARED = true kann als Master eine S-Sperre zugelassen werden, wenn die Warteliste leer ist (d.h. es existieren auch keine externen Warter) und lokal keine X-Sperre auf dem Block gewährt ist. Eine X-Sperre auf einem "gemeinsamen" Block kann nur gewährt werden, wenn der Rechner Master ist und in beiden Rechnern HS gesetzt ist (s.o.). In diesem Fall wird der andere Rechner aufgefordert, seinen Blockeintrag aufzugeben (RELEASE-AUFFORDERUNG). Neben den erwähnten Nachrichtentypen LOCK-REQUEST, LOCK-RESPONSE und RELEASE-AUFFORDERUNG, werden auch auf Blockebene Nachrichten zur Aktualisierung der globalen Sperrinformationen (GHT, Blockeinträge mit SHARED = true) gesendet.

In den Simulationen wurde eine TA zurückgesetzt, wenn sie auf Grund einer Sperranforderung in einen Wartezustand versetzt wurde, der einen Deadlock zwischen lokalen TA verursachte. Auf eine globale Deadlockerkennung wurde verzichtet unter der Annahme, daß es selten zu einer globalen Verklemmung kommt. Es wurde daher lediglich eine maximale Wartezeit auf eine globale Sperre vorgegeben, nach deren Überschreiten die TA zurückgesetzt wurde, da unterstellt wurde, sie sei in einen globalen Deadlock verwickelt.
Bei der Freigabe von Sperren (UNLOCK) werden möglicherweise wartende TA aktiviert oder Haltesperren gesetzt. Für eine genaue Beschreibung der stattfindenden Aktionen, ebenso wie für den Aufbau und die Abarbeitung der einzelnen Nachrichtentypen, muß wiederum auf [Ra84] verwiesen werden.

4.3 Protokolländerungen bei nur einem Update-Rechner

Dieser Spezialfall bringt für DB-Sharing folgende Vorteile:
1. Der Retrieval-Rechner braucht keine Log-Daten zu führen.
2. Ein Ausfall des Retrieval-Rechners ist relativ einfach zu behandeln.
3. Für den Update-Rechner existiert kein Veralterungsproblem, da nur er Blöcke ändern kann.
4. Im Update-Rechner können Lesesperren ohne Kommunikation gewährt werden, falls kein lokaler Sperrkonflikt vorliegt. Ebenso können R-, PR- und U-TA im Update-Rechner höchstens mit lokalen TA in Konflikt geraten.

Im Update-Rechner werden keine HS-Sperren geführt, da für ihn kein Veralterungskonflikt existiert und da S-Sperren auch ohne vorhandenen Blockeintrag gewährt werden können. HS-Sperren würden also nur unnötige Behinderungen für den Retrieval-Rechner verursachen.

5. Die Simulation

Mit der Simulation sollten quantitative Aussagen über das entwickelte Sperrverfahren gewonnen werden. Die Simulationen wurden für den 1-Rechner-Fall durchgeführt sowie für ein DB-Sharing mit zwei Update-Rechnern bzw. für einen Update- und einen Retrieval-Rechner. Damit konnten die relativen Antwortzeit- und Durchsatzänderungen im Vergleich zum zentralisierten Fall ermittelt werden. Da wesentliche Funktionen (Recovery, Systempufferverwaltung) nicht simuliert wurden, konnten jedoch keine absoluten Werte, wie etwa Anzahl von TA pro Sekunde, bestimmt werden.

Die Simulationen wurden mit sogenannten **Seitenreferenzstrings** durchgeführt, die aus realen Anwendungen mit dem Datenbanksystem UDS gewonnen wurden. In den Referenzstrings kommen folgende Satzarten vor:

- BOT-Satz
  Er enthält u.a. die TA-Nummer, den TA-Typ, die referenzierten Areas sowie den benutzten Areamodus.
- Logischer Seitenreferenzsatz oder Logref
  In ihm ist neben der TA-Nummer und der Seitenidentifikation noch angegeben, ob die Seite geändert werden soll.
- Unfix-Satz
  Er beschreibt die Freigabe der Seite im Puffer durch die anfordernde TA.
- EOT-Satz
  Er vermerkt das Ende der TA, was die Freigabe ihrer gesamten Sperren impliziert.

In der Simulation mußte eine TA für einen Logref mit Änderungsabsicht eine X-Sperre erwerben, ansonsten genügte eine S-Sperre. X-Sperren wurden bis zum Ende der TA gehalten, während Lesesperren nur kurz (bis zum nächsten Unfix) in Besitz der TA blieben. Diese Verfahrensweise entspricht der in der Praxis üblichen Konsistenzebene 2.
Die Simulationen wurden mit sechs verschiedenen Referenzstrings (Mixes) durchgeführt, deren wichtigsten Charakteristika der Tabelle 1 zu entnehmen sind. Die ersten vier TA-Lasten stammen von kommerziellen DB/DC-Applikationen, während die letzten beiden aus einer Anwendung resultierten, die speziell für Leistungsuntersuchungen erstellt wurde [Re81].

	Mix 1	Mix 2	Mix 3	Mix 4	Mix 5	Mix 6
#der TA	2 288	669	860	2 014	39	262
Anteil U-TA(%)	45.7	46.5	33.6	13.4	12.8	86.3
#Logrefs	9 862	40 751	54 465	57 959	79 750	109 216
#Locks	8 272	38 457	44 981	54 595	79 701	101 454
Anteil X-Locks(%)	40.4	3.7	7.2	9.8	0.05	2.7
#Locks/TA	3.6	57.5	52.3	27.1	2043.6	387.2
Größe der DB (MB)	565	330	315	3 600	60	60
#ref. Seiten	2 188	3 025	6 389	8 515	5 235	3 543
Instr./Logref	10 700	7 100	8 500	8 900	6 900	8 300
Instr./TA	46 000	430 000	540 000	260 000	14 Mill.	3.5 Mill.

Tabelle 1: Übersicht über die wichtigsten Charakteristika der verwendeten TA-Mixes

Die Tabelle 1 zeigt, daß die Referenzstrings stark unterschiedliche Lastprofile repräsentieren. Dies betrifft sowohl Anzahl und Größe der TA als auch die Länge der Mixes und die Änderungsintensität.
Die Anzahl von Instruktionen pro Logref basiert zum Teil auf Pfadlängenmessungen und zum Teil auf Schätzungen. Es wurden dabei neben den eigentlichen Instruktionen im DBVS auch Kosten für Prozesswechsel, Ein/Ausgabe und BS-Overhead berücksichtigt. Diese Angaben beziehen sich auf einen bestimmten Parallelitätsgrad, der bei der Messung der realen Anwendungen erreicht wurde.

5.1 Das Simulationsprogramm

Das Simulationsprogramm besteht aus drei Moduln (Bild 3):

- dem Referenzmodul oder Reference-Manager (RM)
- der TA-Verwaltung oder Transaction-Manager (TM) und
- dem Sperrverwalter oder Lock-Manager (LM).

Die zentrale Komponente ist die TA-Verwaltung, die Funktionen der anderen beiden Moduln aufruft.

Der Sperrverwalter führt die Synchronisation für beide Rechner durch und wurde gemäß dem entwickelten Protokoll realisiert. Er enthält Einträge für READY, FINISH, LOCK und UNLOCK sowie zur Abarbeitung eines Bucks und zum Zurücksetzen einer TA. Die TA-Verwaltung liest für jeden Simulationslauf zunächt die Parameter ein und führt danach das Scheduling und die eigentliche TA-Verarbeitung durch. Am Ende des Laufes werden dann die Ergebnisse ausgegeben. Neben dem Mix wurden bei den Messungen folgende Parameter variiert:

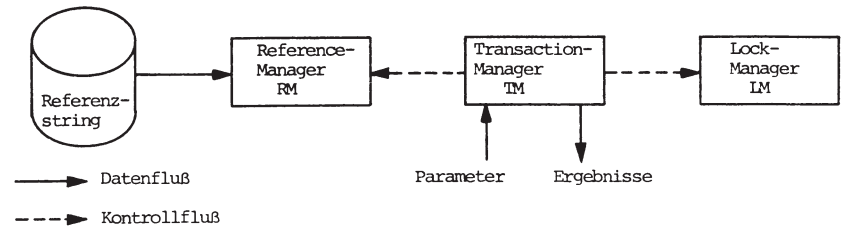


Bild 3: Aufbau des Simulationsprogramms

- Anzahl der Rechner (1 oder 2)
- Parallelität (n pro Rechner)
- Anzahl der Update-Rechner (1 oder 2)
- maximale Dauer der Masterphase (M)

Die Masterphase eines Rechners wird kürzer als der eingestellte Maximalwert, wenn alle n aktivierten TA des Rechners in einen Wartezustand geraten sind. Folgende Parameter wurden u.a. fest gewählt, um die Anzahl der Simulationsläufe zu begrenzen:

- CPU-Leistung eines Rechners, die für die TA-Verarbeitung zur Verfügung steht (3 MIPS)
- Kapazität der Übertragungsstrecke (800 KB/sec)
- Kosten für SEND/RECEIVE eines Bucks (10 000 Instruktionen).

In den Simulationen wurde unterstellt, daß ein Austausch von geänderten Seiten, der aufgrund des Veralterungsproblems notwendig wird, immer über Platte geschieht. Die dadurch entstehenden Verzögerungen wurden explizit berücksichtigt, die sonstige physische E/A wurde jedoch nicht simuliert.
Zur Bestimmung der Ergebnisse führte die TA-Verwaltung für jeden der Rechner eine Simulationsuhr. Sie wird beim Senden, Empfangen und Verarbeiten der Bucks sowie beim Bearbeiten der TA fortgeschaltet. Der Zeitbedarf für die TA-Verarbeitung wurde bei jeder Logref-Bearbeitung durch Erhöhung der Uhr um das Zeitäquivalent der in Tabelle 1 genannten Instruktionsanzahl (Instr./Logref) berücksichtigt.
Die TA-Verwaltung liest nicht direkt den Referenzstring, sondern bekommt nach einer Transformation vom Referenzmodul die zu verarbeitenden Sätze so geliefert, als wenn der String mit der Parallelität n aufgezeichnet worden wäre. Diese Indirektion erlaubt eine Variation von n. In RM wird zudem noch das TA-Routing (Lastkontrolle) durchgeführt, da dort nach Aufforderung seitens des Schedulers eine TA ausgewählt wird. Dabei muß sichergestellt werden, daß dem Retrieval-Rechner keine Änderungstransaktion zugeteilt wird. Für zwei Update-Rechner wurden folgende zwei Möglichkeiten in Betracht gezogen, die Behinderungen zu reduzieren:

1. TA, die auf eine bestimmte Area A zugreifen, werden nur in einem Rechner bearbeitet. Damit sind alle Sperrkonflikte bezüglich dieser Area lokaler Natur, und der Rechner braucht kein Master zu sein, um die Sperranforderungen zu bearbeiten.
2. TA eines bestimmten TA-Typs werden nur auf einem der Rechner ausgeführt. Diese Strategie lohnt sich jedoch nur, wenn innerhalb eines TA-Typs eine hohe Lokalität vorliegt.

Beide Vorgehensweisen sind nur zulässig, wenn dadurch nicht einer der Rechner überlastet wird. Für Mix 1, 2 und 4 konnte Methode 1 und für Mix 6 Methode 2 angewendet werden.

5.2 Bewertungsmodell

Mit den Simulationen sollten Aussagen über die Leistungsfähigkeit des Sperrverfahrens gemacht werden. Hierzu wurden die Änderungen im Durchsatz- und Antwortzeitverhalten bei DB-Sharing im Vergleich zum 1-Rechner-Fall quantifiziert. Vorausgesetzt wurde dabei, daß innerhalb eines Parallelitätsgrades für jeden der DB-Sharing-Rechner die E/A-Häufigkeit (ohne den Austausch geänderter Seiten über Platte) und die Kosten für Inter-Task-Kommunikation, Logging und BS-Overhead denen beim zentralisierten Fall entspechen. Da hierfür in den Simulationen für alle Parallelitätsgrade der gleiche CPU-Bedarf pro Logref eingesetzt wurde, durften die

Ergebnisse nicht zwischen verschiedenen Parallelitäten verglichen werden. Diese Einschränkung gilt unter der Annahme, daß diese Kosten stark vom jeweiligen Parallelitätsgrad abhängen können.
Dem zentralisierten Fall wurde der Durchsatz 1 zugeordnet, da dort immer mindestens eine TA aktiv ist, d.h. nicht auf eine Sperre bzw. auf eine Area wartet. Bei DB-Sharing wirken sich die Zeitanteile durchsatzmindernd aus, in denen keine TA-Verarbeitung stattfindet. Dies sind zum einen die Anteile für das Senden, Empfangen und Verarbeiten der Bucks und zum anderen die Zeiten, in denen alle vom Scheduler gestarteten TA in einen Wartezustand geraten sind. Das Simulationsprogramm bestimmte nun für die beiden Rechner einen gemeinsamen Zeitanteil A (in Prozent), der angibt, in welchem Ausmaß die beiden CPU s zur TA-Verarbeitung genutzt werden konnten. Der Quotient $2 \cdot A/100$ ist dann eine erste Näherung für die Durchsatzänderung D.
Für eine realistischere Berechnung von D muß jedoch noch der Einfluß von zurückgesetzten TA berücksichtigt werden, da diese nichts zum Durchsatz beitragen. Hierzu wurde sowohl im zentralisierten Fall wie bei DB-Sharing ein sogenannter **Wiederholungsfaktor** bestimmt, der ein Maß für die unnötigerweise erledigte Arbeit darstellt. Der Wiederholungsfaktor ist dabei der Quotient zwischen der Anzahl ausgeführter Logrefs und der Anzahl auszuführender Logrefs (>= 1). Wenn q der Wiederholungsfaktor im zentralisierten Fall ist und Q der bei DB-Sharing, dann gilt:

$$D = 2 \cdot A \cdot q \; / \; (100 \cdot Q)$$

Wurden z.B. in jedem Rechner 7 Prozent der Zeit für Senden, Empfangen und Verarbeiten der Bucks verbraucht und waren für 11 Prozent der Zeit alle TA eines Rechners in einem Wartezustand, dann ergibt sich (für $q = Q = 1$)

$$A = 82\,\% \qquad \text{und} \qquad D = 1.64 \; .$$

Um die Antwortzeitänderung R (genauer: Änderung der TA-Bearbeitungszeiten) zu bestimmen, wurden die mittleren Bearbeitungsdauern von TA im zentralisierten Fall mit denen bei DB-Sharing in Beziehung gesetzt. Etwas problematisch ist hierbei jedoch die Berücksichtigung zurückgesetzter TA, da deren Antwortzeiten stark von der Wiederaktivierungs-Strategie abhängen. Bei uns wurden nur die mittleren Bearbeitungszeiten von nicht zurückgesetzten TA ermittelt und anschließend wiederum eine Relativierung über die Wiederholungsfaktoren vorgenommen. Wenn t die mittlere Bearbeitungsdauer einer TA im 1-Rechner-Fall ist und T die bei DB-Sharing, dann gilt:

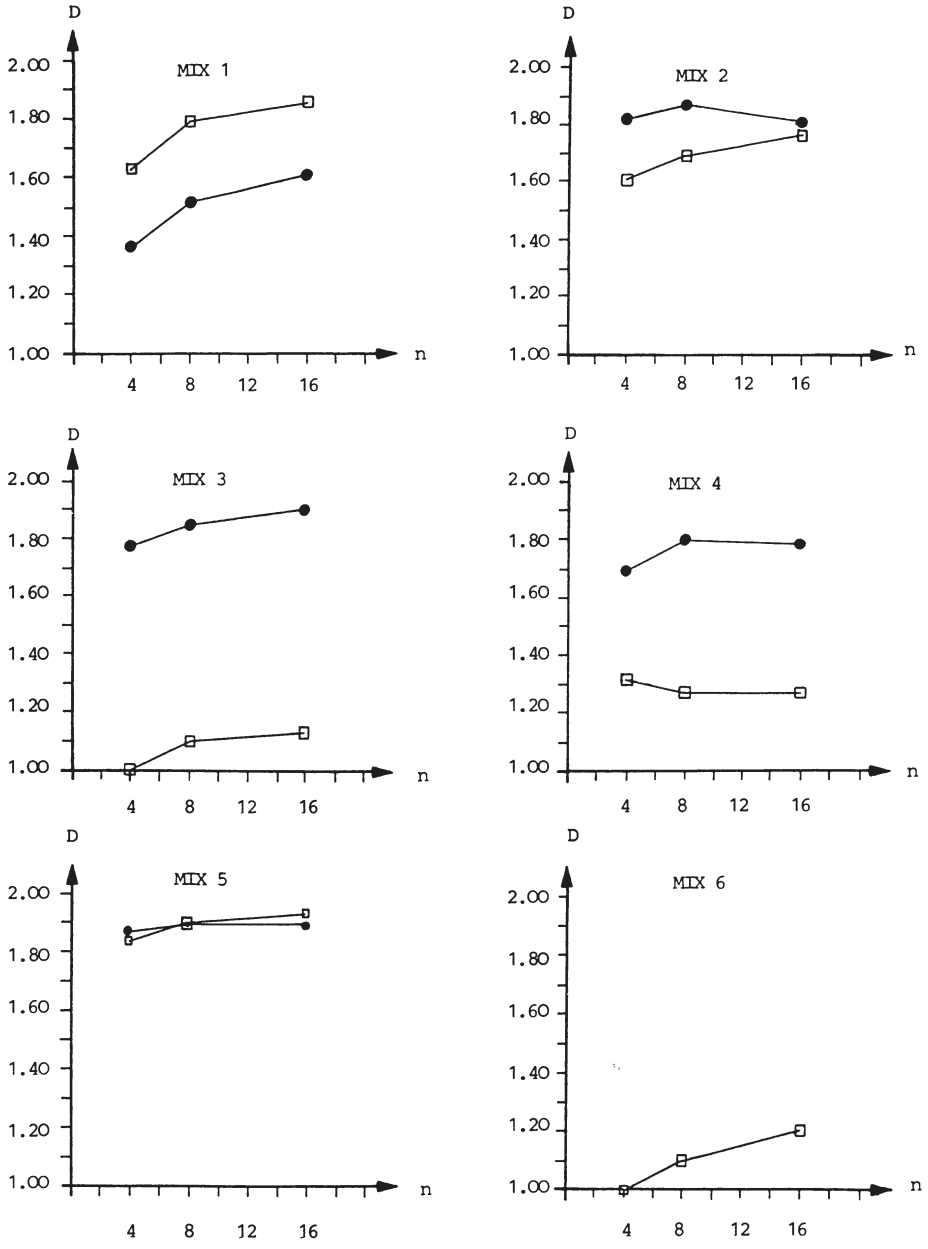
$$R = T \cdot Q \; / \; (t \cdot q)$$

Der Wert $R = 1.15$ bedeutet beispielsweise eine Verschlechterung der TA-Bearbeitungszeiten von 15 Prozent.

6. Simulationsergebnisse

Die Simulationen wurden mit drei Parallelitätsgraden (4, 8 und 16) und bei den DB-Sharing-Messungen mit drei unterschiedlichen Werten für die maximale Dauer M einer Masterphase (20, 40 und 60 msec) durchgeführt. Für Mix 6 wurden keine DB-Sharing-Simulationen mit nur einem Update-Rechner vorgenommen, da für ihn der Anteil der Lesetransaktionen zu gering ist (weniger als 15%).
Bild 4 zeigt die Durchsatz- und Bild 5 die Antwortzeitänderungen für jeden der Referenzstrings im Vergleich zu dem 1-Rechner-Fall. Die Diagramme enthalten je eine Kurve für die Werte bei zwei Update-Rechnern (U-Rechner) bzw. mit einem Update- und einem Retrieval-Rechner. Für jeden Parallelitätsgrad ist nur der beste Wert (der meist für M = 40 msec vorlag) gezeigt.

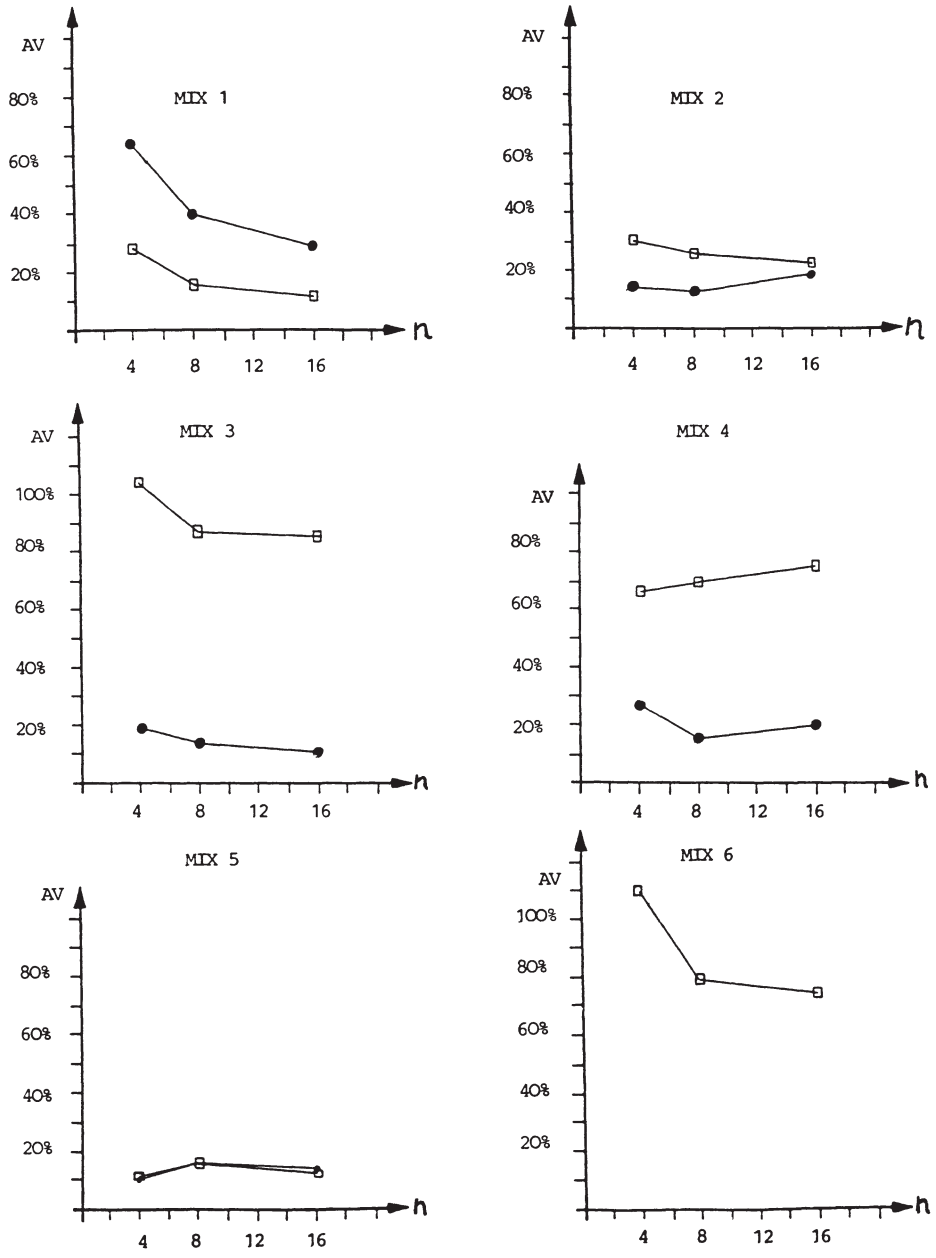
Für Mix 2, Mix 3 und Mix 4 waren Durchsatz- und Antwortzeitänderungen für einen U-Rechner zum Teil deutlich besser als für zwei U-Rechner. Die Durchsatzsteigerungen lagen - wie auch für Mix 5 - zwischen 80% und 90%, die Verschlechterungen der TA-Bearbeitungszeiten betrugen zwischen 10% und 20%. Für Mix 1 waren die Ergebnisse für den allgemeineren Fall mit zwei U-Rechnern erstaunlicherweise deutlich besser als für einen U-Rechner (bester Wert D = 1.85 bei einer Antwortzeitverschlechterung von 11% im Gegensatz zu D = 1.61 und 28% schlechteren Antwortzeiten). Die guten



D = Durchsatzänderung
n = Parallelität

□ 2 Update-Rechner
● 1 Update- und 1 Retrieval-Rechner

Bild 4: Durchsatzänderungen im Vergleich zum 1-Rechner-Fall



AV = Antwortzeit-Verschlechterung
n = Parallelität

□ 2 Update-Rechner
● 1 Update- und 1 Retrieval-Rechner

Bild 5: Antwortzeitänderungen im Vergleich zum 1-Rechner-Fall

Werte für zwei U-Rechner wurden durch die vorgenommene Lastkontrolle erreicht, mit der alle globalen Sperrkonflikte auf einer besonders kritischen Area beseitigt werden konnten. Für die (fast reine) Leselast Mix 5 wurden die besten Ergebnisse erreicht, unabhängig davon ob einer oder zwei Rechner Änderungen vornehmen konnten. Der Durchsatz stieg für diese TA-Last im Vergleich zum 1-Rechner-Fall bis zu 92% bei nur 10% schlechteren Antwortzeiten.

Für Mix 3, Mix 4 und Mix 6 kam es für zwei U-Rechner zu sehr schlechten Ergebnissen. Die besten Werte lagen bei den Durchsatzsteigerungen zwischen 13% (Mix 3) und 31% (Mix 4) und bei den Antwortzeitverschlechterungen zwischen 65% (Mix 4) und 84 % (Mix 3). Bei den genannten Referenzstrings führten vor allem Sperrkonflikte auf Hot-Spot-Objekten, für die der Zugriff auch auf Seitenebene synchronisiert wurde, zu diesen Resultaten. Darüberhinaus entstanden vor allem für Parallelität 8 und 16 eine Fülle von globalen Deadlocks. Die Ergebnisse wurden dann einerseits über den Wiederholungsfaktor Q verschlechtert und außerdem verursachte die globale Deadlockerkennung über einen Timeout-Mechanismus zum Teil sehr lange Behinderungen für die beteiligten TA.
Der Aufwand für das Senden, Empfangen und Verarbeiten der Bucks war im wesentlichen nur abhängig vom Parameter M. Für M = 20 msec wurde dafür rund 8% und für M = 60 msec etwa 3.5% der CPU-Leistung in jedem der Rechner benötigt. Daraus ergibt sich für das untersuchte Parameterspektrum eine maximal mögliche Durchsatzsteigerung von 1.93, die von Mix 5 nahezu erreicht wurde. Die mit **länger werdenden Masterphasen** verbundene Verringerung des Kommunikationsaufwandes fiel wesentlich weniger ins Gewicht als die **damit entstehenden Wartezeitverlängerungen** auf eine globale Sperre bzw. bis zur nächsten Masterphase. So wurden für M = 60 msec fast immer die schlechtesten Resultate gemessen. **Kleinere M-Werte** als 20 msec führen dagegen zu sehr **stark steigenden Kommunikationskosten** und dazu, daß nur wenige TA in einer Masterphase ´bedient´ werden können. Dies erlaubt jedoch nur geringe Parallelitätsgrade und damit verbunden eine geringe Nutzung der Rechnerkapazitäten (schlechter Durchsatz).
Der Zusammenhang zwischen Parallelität und Durchsatz wird auch aus Bild 4 deutlich. Außer für die TA-Lasten, bei denen verstärkt globale Deadlocks auftraten, kam es **mit zunehmender Parallelität zu wachsenden Durchsatzsteigerungen.** Der Grund dafür war, daß bei höherer Parallelität es immer seltener vorkam, daß alle TA nicht mehr weiterarbeiten konnten. Dies bewirkte eine bessere Nutzung der Slavephasen und damit einen erhöhten Durchsatz.
Sehr überraschend ist jedoch, daß in Bild 5 die Antwortzeitverschlechterungen in den meisten Fällen mit steigender Parallelität n geringer werden. Dieses Phänomen trat auf, obwohl die Behinderungen bei DB-Sharing wegen der doppelten Anzahl von TA immer größer waren als bei einem Rechner und obwohl bei DB-Sharing Antwortzeitverzögerungen aufgrund der Kommunikationsnotwendigkeit auftraten. Zur Erklärung soll beispielhaft Mix 1 im 1-Rechner-Fall sowie für ein DB-Sharing mit einem U-Rechner betrachtet werden. Für n = 4 war die mittlere Antwortzeit bei DB-Sharing etwa 63% und für n = 16 nur noch 28% (für M = 40 msec) schlechter als im zentralisierten Fall. Tabelle 2 zeigt, wie sich die mittleren Antwortzeiten zusammensetzen.

	1-Rechner-Fall		DB-Sharing	
	n = 4	n = 16	n = 4	n = 16
eigentlicher CPU-Bedarf	25.0%	6.3%	15.3%	4.9%
Wartezeit auf Sperren	3.3%	17.7%	19.2%	32.9%
Wartezeit auf Masterphase	-	-	31.3%	25.6%
Verzögerung wegen Kommunikationsabwicklung	-	-	1.6%	1.7%
CPU-Belegung durch andere TA	71.7%	76.0%	32.6%	34.7%

Tabelle 2: Zusammensetzung der mittleren Antwortzeiten am Beispiel von Mix 1

Wie man sieht, sind für DB-Sharing die Behinderungen wegen der Sperrkonflikte bzw. wegen des Protokolls wesentlich höher als im zentralisierten Fall, jedoch konkurrieren im 1-Rechner-Fall durchschnittlich mehr als doppelt soviele rechenwillige TA um die CPU. Außerdem ist der Anteil der Antwortzeit, der mit Warten auf Sperren verbraucht wird, im 1-Rechner-Fall bei n = 16 um gut 14% auf

über das Fünffache gegenüber n = 4 angestiegen. Bei DB-Sharing stieg der Zeitanteil, in dem eine TA nicht rechenbereit war, dagegen nur um 8% (von 52% auf 60%).
Die globalen Sperrinformationen (in den Blockeinträgen, in der Areatabelle und in der GHT) sowie die Haltesperren führten - vor allem bei einem U-Rechner - dazu, daß viele Sperren sofort gewährt werden konnten bzw. keine Kommunikation erforderten. Bei zwei U-Rechnern konnten so 48% bis 94% der Sperren eines Mixes sofort und 5% bis 45% nach Eintritt der Masterphase gewährt werden. Der Anteil der noch länger verzögerten Sperranforderungen lag zumeist unter 5%. Bei nur einem U-Rechner konnten (außer für Mix 1) sogar 87% - 98% der Sperren sofort gewährt werden. So war es keine Seltenheit, daß für 70% und mehr der TA kein Sperrkonflikt vorkam bzw. kein LOCK-REQUEST an den anderen Rechner notwendig war.
Der Grund für diese Zahlen liegt darin, daß selbst bei zwei U-Rechnern wegen der Haltesperren meist für mehr als 80% aller Sperranforderungen ein Blockeintrag vorhanden war. Bei vorhandenem Blockeintrag konnten die Sperren nahezu immer gewährt werden, wenn auch für SHARED=true der Rechner i.a. Master sein mußte. Selbst wenn bei einer Sperranforderung noch kein Sperrkontrollblock vorlag, konnte die Sperre (als Master) zumeist wegen eines OO-Eintrages in der GHT gewährt werden. Dies wurde durch eine relativ große GHT (16K Einträge) erreicht, so daß selten ein 'ungünstiger' GHT-Eintrag vorlag, wenn nicht tatsächlich eine Kommunikation mit dem anderen Rechner (z.B. wegen eines Veralterungskonflikts) notwendig war.
Bei nur einem U-Rechner wurden die Slavephasen wesentlich besser genutzt als für zwei, da im U-Rechner auch als Slave Lesesperren gewährt werden können. Außerdem galt im Retrieval-Rechner bei fast allen Blockeinträgen SHARED = false, da der U-Rechner keine HS-Sperren hält. Für Mix 1 jedoch führte der hohe Anteil von X-Sperren (über 40%) dazu, daß im U-Rechner nur noch relativ wenige Lesesperren vorkamen und auch im Retrieval-Rechner verstärkt Konflikte mit Update-TA auftraten. Das bewirkte für diesen Mix die schlechtesten Ergebnisse, die mit einem U-Rechner vorkamen.
Der Anteil von Sperranforderungen, für die wegen eines Veralterungskonflikts ein Austausch der geänderten Seite über Platte notwendig war, lag meist unter 1%. Diese Verzögerungen fielen kaum ins Gewicht, zumal bei Masterphasen von 40 oder 60 msec die Seite bis zum Wegschicken der LOCK-RESPONSE bereits ausgeschrieben war.

7. Zusammenfassung

Für ein DB-Sharing mit zwei Rechnern wurde ein Sperrverfahren vorgestellt, mit dem eine deutliche Durchsatzerhöhung bei vertretbaren Antwortzeiten erreicht werden sollte. Zur Quantifizierung der Leistungsfähigkeit wurden Simulationen durchgeführt, bei denen als Lastmodell und als DB-Referenzmodell Seitenreferenzstrings herangezogen wurden. Die Verwendung dieser Referenzstrings garantiert eine genaue und realitätsnahe Simulation der Anwendung und insbesondere von Wartesituationen. Durch die Benutzung von sechs Mixes konnten verschiedene Anwendungsklassen betrachtet werden, was den Praxisbezug der Ergebnisse zusätzlich erhöht. Die im realen Betrieb (1-Rechner-System) aufgezeichneten Referenzstrings verkörpern implizit das Zugriffsverhalten, das Auftreten von Prozeßwechseln und die auf Grund gesetzter Sperren enstehenden Wartesituationen der Transaktionen der betreffenden Anwendung bei der aktuell erreichten Parallelität (< 8). Eine Transformation dieser Referenzstrings auf beliebige höhere Parallelitätsgrade ist sicherlich unzulässig, wenn auf die Realitätstreue des Lastmodells Wert gelegt wird. Die simulierten Parallelitätsgrade (4 - 16 pro Rechner) dürften jedoch noch eine hohe Übereinstimmung von realem Ablauf und modellhafter Nachbildung gewährleisten.
Das Sperrverfahren kann als Erweiterung des beim Data Sharing in IMS verwendeten Protokolls aufgefaßt werden, wobei neben einer Synchronisation auf Blockebene ein weiteres Granulat (Area) berücksichtigt wurde. Zudem wurde durch die Einführung der Haltesperren eine Unterstützung der Lokalität der Verarbeitung und eine Lösung des Veralterungsproblems bei einer ¬FORCE-Ausschreibstrategie erreicht. Bei den Untersuchungen wurden keine absoluten Leistungsaussagen vorgenommen, da hierzu kein Referenzsystem für die zur Validierung erforderlichen Messungen zur Verfügung stand. Zur Bestimmung des Durchsatz- und Antwortzeitverhaltens wurde daher der gleiche Algorithmus für den 1-Rechner-Fall angewendet, um die gewonnenen Ergebnisse als Bezugsgröße für die DB-Sharing-Resultate zu benutzen. Die Simulationen ergaben durchaus zufriedenstellende Durchsatzsteigerungen, während sich die TA-Bear-

beitungszeiten teilweise stark erhöhten. Die (simulierte) relative Antwortzeitverschlechterung bezieht sich jedoch nur auf die DB-Verarbeitung. Die Antwortzeit in einem Transaktionssystem dagegen besitzt einen DB- und einen (nicht zu unterschätzenden) DC-Anteil, der auch bei DB-Sharing konstant bleiben dürfte.
Die unzufriedenstellenden Resultate, die durch bestimmte Seitentypen verursacht wurden, die sich vielfach als 'hot spots' herausstellten, dürften durch eine eintragsweise Synchronisation zwischen den Rechnern verbessert werden. Wenn es die TA-Last erlaubt, garantiert die Verwendung von nur einem ändernden Rechner, ebenso wie der Einsatz einer optimierten Lastkontrolle, zum Teil deutlich bessere Leistungsmerkmale.
Obwohl prinzipiell möglich, erscheint es aus verschiedenen Gründen nicht ratsam, das vorgestellte Verfahren auf mehr als zwei Rechner auszuweiten. Die Untersuchung allgemeinerer Synchronisationsalgorithmen ist Gegenstand weiterer Forschungsarbeit.

Danksagung
Viele Anregungen und Hinweise zum Thema "Synchronisation bei DB-Sharing" bekamen wir von A. Reuter, bei dem wir uns auch für seine ständige Diskussionsbereitschaft bedanken möchten. M. Köstler hat insbesondere zum Gelingen der Implementierung des Simulationsprogrammes beigetragen, wofür wir ihm ebenfalls danken wollen.

Literaturverzeichnis

CODA73 Codasyl DDL Journal of Development, June 73 Report, erhältlich bei IFIP Administrative Data Processing Group, 40 Paulus Potterstraat, Amsterdam.

Gr78 Gray, J.: Notes on Database Operating Systems, in: Operating Systems: an Advanced Course, Bayer, R., Grahan, R.M., Seegmüller, G. (eds.), Lecture Notes on Computer Science 60, Springer Verlag, 1978, pp. 393-481.

Gr85 Gray, J., Good, B., Gawlick, D., Homan, P., Sammer, H.: One Thousand Transactions per Second, in: Proc. IEEE-Spring CompCon, San Francisco, Feb. 1985.

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

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

HR85 Härder, T., Rahm, E.: Klassifikation und Eigenschaften von Mehrrechner-Datenbanksystemen, Interner Bericht (in Vorbereitung), FB Informatik, Uni Kaiserslautern, 1985.

IMS81 Administering IMS/VS Systems that Share Data, in: IMS/VS V1, System Administration Guide, Release 2, Program Number 5740-XXL, SH20-9178-1, März 1981, S. 357-390.

Ke82 Keene, W.N.: Data Sharing Overview, in: IMS/VS V1, DBRC and Data Sharing User's Guide, Release 2, G320-5911-0, August 1982.

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

PR83 Peinl, P., Reuter, A.: Empirical Comparison of Database Concurrency Control Schemes, in: Proc. 9th Int. Conf. on VLDB, Florenz, 1983, S. 97-108.

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

Re81 Reuter, A.: Fehlerbehandlung in Datenbanksystemen, Hanser Verlag, München Wien, 1981.

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

SUW82 Strickland, J.P., Uhrowczik, P.P., Watts, V.L.: IMS/VS: An Evolving System, in: IBM Systems Journal 21, No. 4, 1982, S. 490-510.

UDS84 UDS4.0, Anwendungen programmieren, Benutzerhandbuch, Siemens AG, München, Feb. 1984, Bestell-Nr.: U930-J-Z55-2.

Tr83 Traiger, I.: Trends in Systems Aspects of Database Management, IBM Research Report RJ 3845, San Jose, USA, April 1983.

Diese Arbeit wurde von der SIEMENS AG finanziell unterstützt.