Erhard Rahm

Synchronisation in Mehrrechner-Datenbanksystemen

Konzepte, Realisierungsformen und quantitative Bewertung

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo

I Einführung

1. Einleitung

Der Einsatz von Datenbanksystemen (DBS) zur Verwaltung gemeinsamer Datenbestände in administrativ-betriebswirtschaftlichen Rechneranwendungen nimmt ständig zu. Innerhalb von **Transaktionssystemen** /HäMe86a,HäMe86b,Mey86/ werden sie vor allem dort eingesetzt, wo eine große Anzahl von Buchungs-, Auskunfts- oder Datenerfassungsvorgängen anfallen, also etwa bei Banken, Fluggesellschaften und Reisebüros oder zur 'aktenlosen' Sachbearbeitung bei Behörden und Versicherungen. Eine weitergehende Verbreitung von Transaktionssystemen ergibt sich auch mit Zunahme von 'Point-of-sale'- und BTX-Anschlüssen.

Den Kern eines Transaktionssystems bildet ein DB/DC-System. Während die DB-Komponente für alle Aufgaben der Datenhaltung und -sicherung zuständig ist, übernimmt das DC-System (Datenkommunikationssystem) die Verwaltung der Ein- und Ausgabenachrichten mit dem Terminal und der Transaktionsprogramme sowie Aufgaben der Speicherverwaltung /Mey86/. Im einfachsten Fall schickt der Benutzer eine Eingabenachricht (Bildschirmformular mit Transaktionscode und Daten) an das DC-System und erhält nach Bearbeitung des Auftrages durch das zugehörige Transaktionsprogramm, von dem aus die DB-Operationen aufgerufen werden, eine Ausgabenachricht am Terminal zurück. Im Gegensatz zu solchen *Einschritt-Transaktionen* umfassen *Mehrschritt-Transaktionen* mehrere Interaktionen (Dialogschritte) zwischen Benutzer und DB/DC-System. Die Verarbeitung der Benutzeraufträge durch das Transaktionssystem muß dabei unter bestimmten Bedingungen und Betriebscharakteristika erfolgen, die schon bei der Entwicklung des DB/DC-Systems zu berücksichtigen sind /HäMe86a/:

- Anschluß und Dialogbedienung sehr vieler Terminals (bis zu mehreren tausend)
- optimale Unterstützung von betrieblichen Abläufen durch vordefinierte Transaktionsprogramme (canned transactions)
- konkurrierende Ausführung sehr vieler kurzer Transaktionen (TA)
- Zugriff auf gemeinsame Datenbestände mit größtmöglicher Aktualität
- stochastisches Verkehrsaufkommen
- kurze Antwortzeiten als Optimierungsziel vor Auslastung.

Durch die ständig zunehmende Verbreitung und Benutzung von TA-Systemen ergaben sich vor allem für große Anwender - insbesondere Banken und Fluggesellschaften - in letzter Zeit weitergehende Anforderungen bezüglich der Leistungsfähigkeit, die von herkömmlichen DB/DC-Systemen auf zentralisierten Rechnersystemen nicht mehr erbracht werden können. Die Erfüllung dieser Anforderungen verlangt den Einsatz sogenannter **Mehrrechner-Datenbanksysteme**, bei denen die TA-Last auf mehreren gekoppelten Rechnern bearbeitet wird (eine präzisere Charakterisierung von Mehrrechner-DBS sowie eine Klassifikation folgen in Kap. 2). Es handelt sich dabei im wesentlichen um folgende vier **Anforderungen** /HäRa86/:

1. Abwicklung hoher TA-Raten

Eine Forderung aus /Gra85/ lautet hier bis 1990 pro Sekunde 1000 TA vom Typ Kontenbuchung (Debit-Credit /Anon85/) abzuwickeln, wobei für 95 % der TA eine Antwortzeit unter 1 Sekunde

einzuhalten ist; in großen Anwendungen werden schon bald noch höhere Durchsatzleistungen benötigt. Für die sehr einfachen TA bei der Telefonvermittlung sollen in absehbarer Zeit sogar Zehntausende von TA pro Sekunde DB-gestützt abgewickelt werden /HeGo87/. Da selbst bei kurzen TA wie der Kontenbuchung und 1000 TA/s eine Rechnerkapazität von über 100 MIPS erforderlich ist, wird somit bereits der Einsatz von Mehrrechner-DBS erzwungen. Allerdings dürfen die TA-Antwortzeiten in Mehrrechner-DBS, trotz den zur Bearbeitung einer TA nun i.a. notwendig werdenden Kommunikationsvorgänge, nicht nennenswert über denen heutiger Ein-Rechner-Systeme liegen, soll die Akzeptanz des Systems für Dialoganwendungen erhalten bleiben. Denn längere Antwortzeiten führen zu verringerter Produktivität und können eine Frustration des Benutzers oder Verärgerung wartender Kunden verursachen. Eine verminderte Produktivität hat jedoch auch zur Folge, daß zur Gewährleistung eines bestimmten Durchsatzes mehr Personal benötigt wird, was aus Kostengründen häufig nicht toleriert wird /ReSh84/.

Die Notwendigkeit des Einsatzes von Mehrrechner-DBS wird noch dadurch verschärft, da in vielen Anwendungen der Leistungsbedarf stärker zunimmt als die Steigerung der CPU-Leistung von Monoprozessoren oder eng gekoppelten Systemen. Dies ergibt sich nicht nur aus den steigenden TA-Raten, sondern auch durch hinzukommende TA-Typen, die zunehmende Verwendung von höheren Programmiersprachen und komfortableren Benutzerschnittstellen sowie die Bearbeitung komplexerer Vorgänge mit umfangreichen Integritätsbedingungen. Zusätzliche Probleme bereiten TA-Lasten, bei denen konkurrierend zu den kurzen TA auch Mehrschritt-TA, die vielen Arbeitsabläufen in natürlicherer Weise entsprechen, oder gar Stapelprogramme abgewickelt werden sollen /Inm86/.

Im Gegensatz zu den geforderten TA-Raten können heutige kommerziell verfügbare DBS meist nur weitaus bescheidenere Durchsatzanforderungen erfüllen, oft weniger als 100 TA/s. Das zentralisierte DBS IMS Fast Path, das speziell für die Abwicklung einfacher TA ausgelegt ist, erreicht mit heutiger Hardware (auf einem eng gekoppelten Multiprozessor) etwa bis 400 TA/s vom Typ Kontenbuchung. Für einfachere TA konnten in einem aktuellen Benchmark (August 1987) unter Nutzung des derzeit schnellsten IBM-Quadro-Prozessors (etwa 50 MIPS) sogar rund 1000 TA/s abgewickelt werden /Vig87/. Höhere TA-Raten werden von zentralisierten Systemen derzeit nur von Anwendungssystemen /Bur85,GiSp84/ erbracht, die auf dem Spezial-Betriebssystem TPF (Transaction Processing Facility) - vormals ACP (Airline Control Program /Siw77/) genannt - von IBM aufsetzen. Da TPF jedoch kein Datenbanksystem ist und nur eine sehr niedrige Programmierschnittstelle unterstützt, ist die Erstellung und Wartung der Anwendungssysteme äußerst aufwendig und kostenintensiv. Tandem glaubt schon heute /BGH86/ mit seinem Mehrrechner-DBS Encompass /Bor81/ und 100 seiner neuen VLX-Rechner /Ele86/ 1000 TA/s vom Debit-Credit-Typ schaffen zu können. Dies konnte jedoch noch nicht nachgewiesen werden; vielmehr beruht diese Aussage auf der Annahme eines mit der Rechneranzahl linear wachsenden Durchsatzes. Ein solch lineares Durchsatzwachstum konnte nämlich bei Messungen mit kleineren Konfigurationen beobachtet werden /HoCh85/. Mit dem neuen DBS NonStop SQL von Tandem wurden in einem Benchmark mit 32 VLX-Rechnern 'nur' 208 Debit-Credit-TA/s gemessen /Tan87/, so daß für 1 KTPS selbst bei linearer Durchsatzsteigerung bereits mehr als 150 VLX-Rechner vorzusehen wären.

2. Hohe Verfügbarkeit

Der Ausfall des TA-Systems bedeutet in den meisten Einsatzgebieten unmittelbare Umsatz- und Gewinneinbußen, weil während der Ausfallzeit z.B. keine Buchungen mehr vorgenommen werden können oder Sachbearbeiter tätigkeitslos herumsitzen. Hohe Verfügbarkeit ist daher eine zentrale

Forderung für alle TA-Systeme. Eine typische Anforderung für große Anwender erlaubt lediglich eine Ausfallzeit von fünf Minuten pro Jahr, bei der Telefonvermittlung liegen die Verfügbarkeitserfordernisse sogar noch höher /Gra85/. Die Erreichung einer solch 'permanenten' Verfügbarkeit (continuous operation) ist nur durch Fehlertoleranz (Redundanz) bei allen wichtigen Systemkomponenten (Prozessoren, Platten, Kommunikationswege, Kanäle, Kontroller, etc.) zu erreichen /Kim84/, damit der Ausfall einzelner Systemteile für den Benutzer transparent gehalten werden kann. Insbesondere können nur durch Mehrrechner-Architekturen CPU-Ausfälle verkraftet werden.

Zur Erlangung einer hohen Verfügbarkeit zählt auch, daß die Software- und Hardware-Konfiguration im laufenden Betrieb erweiterbar und änderbar ist (Einspielen neuer Betriebssystem- oder DBS-Versionen, Hinzufügen von Plattenlaufwerken u.ä.), daß Fehler automatisch erkannt und behoben werden und ausgefallene Systemkomponenten unterbrechungsfrei reintegriert werden können. Weiterhin ist eine einfache Handhabbarkeit und eine einfache Verwaltung (s.u.) besonders wichtig, weil jüngsten Untersuchungen /Gra86a/ zufolge - zumindest in fehlertoleranten Systemen - Fehler in der Administration eine hauptsächliche Ausfallursache darstellen.

Pionier und Marktführer auf dem Gebiet fehlertoleranter TA-Systeme ist Tandem /Bar78, Bar81/, das schon seit 1975 ausfallsichere Rechensysteme anbietet. Seit Beginn der 80er Jahre haben sich viele Konkurrenten auf dem Markt zu etablieren versucht /Kim84, Ser84/, jedoch konnte sich bisher im wesentlichen nur Stratus durchsetzen /Ser85/. Das Auragen-System /BBG83/ wird mittlerweile in überarbeiteter Form von Nixdorf unter der Bezeichnung Targon 32 vertrieben.

3. Modulare Erweiterungsfähigkeit

Die Forderung nach modularer Erweiterbarkeit verlangt ein mit den Anwendungen schritthaltendes Wachstum des TA-Systems. Vor allem sollte der Durchsatz mit der Rechneranzahl etwa linear anwachsen können, ohne daß nennenswerte Antwortzeitverschlechterungen in Kauf zu nehmen sind. Diese Forderung schließt monolithische Systeme von vorneherein aus; auch bei eng gekoppelten Mehrprozessor-Systemen ergibt sich oft eine frühzeitige Sättigung (z.B. bei 3-4 Prozessoren) durch überlastete Speicher oder Kommunikationsstrukturen.

4. Handhabbarkeit und einfache Verwaltung

Neben einer hohen Benutzer- und Programmierschnittstelle gilt es vor allem in Mehrrechner-DBS wegen der erhöhten Systemkomplexität, eine möglichst einfache Bedienung, Administration, Konfigurierbarkeit und Wartung zu ermöglichen. So sollte idealerweise nicht nur für den Benutzer und Anwendungsprogrammierer, sondern auch für Systemprogrammierer und Operateure ein 'single system image', das die Existenz mehrerer Rechner verbirgt, angestrebt werden. Die Verwaltung des gesamten Systems könnte dann über eine zentrale Master-Konsole erfolgen.

Neben den genannten Hauptanforderungen wird von Mehrrechner-DBS auch vielfach die Fähigkeit zur **Ortsverteilung** verlangt, um etwa in großen Unternehmen eine Anpassung des TA-Systems an die Organisationsstruktur zu ermöglichen (verteilte Datenbanksysteme) oder aber um etwa nach einer Explosion oder einem Erdbeben eine Katastrophen-Recovery durchführen zu können (siehe Kap. 2). Andererseits ist in ortsverteilten Systemen wegen der langsamen Kommunikationsverbindungen die Abwicklung hoher TA-Raten nur möglich, wenn der weitaus größte Teil (> 95 %) der TA lokal abgewickelt werden können, was jedoch nicht immer realisierbar sein dürfte /DYB87/.
Selbstverständliche Forderungen sind dagegen die Einhaltung des TA-Paradigmas /HäRe83b/

(Synchronisation, Recovery) sowie ein akzeptables Preis/Leistungsverhältnis.

Mehrrechner-DBS, welche die obigen Forderungen nach hohen TA-Raten, hoher Verfügbarkeit und modularer Erweiterungsfähigkeit erfüllen, werden auch als **Hochleistungs-Datenbanksysteme** bezeichnet /HäRa87/; sie sollen in dieser Arbeit weiter betrachtet werden. Daneben kommen Mehrrechner-DBS aber auch z.B. zur Realisierung sogenannter **Non-Standard-DBS** /HäRe85/ in Betracht, mit denen die Vorteile von Datenbanksystemen in neuen Anwendungsgebieten wie etwa VLSI, CAD/CAM oder Expertensystemen genutzt werden sollen. In einigen Prototyp-Implementierungen /HMMS87, Reu87, BLDN87/ wird dabei versucht, Parallelität innerhalb einer komplexen TA (Operation) auszunutzen und durch parallele Bearbeitung reihenfolgeunabhängiger Teiloperationen auf einem Mehrrechner-System möglichst kurze Antwortzeiten zu erreichen. Obwohl die allgemeinen Mehrrechner-Architekturen, die im nächsten Kapitel genauer vorgestellt werden, auch für Non-Standard-DBS anwendbar sind, ergeben sich bei der Realisierung signifikante Abweichungen und Erweiterungen, auf die in dieser Arbeit nicht näher eingegangen werden kann. So muß in Entwurfsumgebungen z.B. ein vollkommen neues TA-Konzept entwickelt werden, das die adäquate Verarbeitung sehr langer Entwurfs-TA sowie die Kooperation zwischen mehreren Designern unterstützt /KLMP84, BKK85, HHMM87/.

Ein zentrales Entwurfsproblem bei Mehrrechner-DBS ist, daß allen Anwendungen das Bild einer zentralen Datenbank zu zeigen ist - unabhängig davon, ob alle Rechner auf die ganze Datenbank oder nur auf eine Partition von ihr zugreifen können, ob Teile der Daten repliziert verwaltet werden oder nicht, oder wo die TA ausgeführt wird (Ortstransparenz). Desweiteren ist einem Benutzer - wie schon bei zentralisierten DBS - durch geeignete Synchronisationsmaßnahmen die konkurrierende Verarbeitung anderer TA zu verbergen (logischer Einbenutzerbetrieb) und die Konsistenz der Daten darf trotz Mehrbenutzerbetrieb nicht gefährdet werden. Die geforderte Fehlertransparenz schließlich verlangt die Bereitstellung von geeigneten Recovery-Strategien, mit denen z.B. die Ausfallbehandlung für einen Rechner durch die überlebenden Prozessoren vorgenommen werden kann.

Die **Realisierung der Synchronisationskomponente**, die in einem Mehrrechner-DBS entscheidenden Einfluß auf die Leistungsfähigkeit hat, steht im Mittelpunkt dieser Arbeit. Dabei werden eine Reihe neuer Synchronisationsprotokolle für Hochleistungs-DBS entwickelt, die mit einem möglichst geringen Ausmaß an Interprozessor-Kommunikationen auskommen, da nur so die geforderten hohen TA-Raten und kurzen Antwortzeiten erreichbar sind. Neben einem qualitativen Verfahrensvergleich werden die vielversprechendsten Konzepte durch detaillierte und realitätsnahe Simulationen quantitativ bewertet und ihr Verhalten analysiert.

Bevor auf die Realisierung der Synchronisationskomponente in Hochleistungs-DBS genauer eingegangen werden kann, ist es zunächst erforderlich, die betrachteten Typen von Mehrrechner-DBS genauer festzulegen, kennzeichnende Eigenschaften herauszuarbeiten und eine Beurteilung mit Hinblick auf die Realisierung von Hochleistungs-DBS vorzunehmen. Diesen Aufgaben widmet sich das nächste Kapitel, in dem zur besseren Orientierung als erstes eine Klassifikation von Mehrrechner-DBS vorgestellt wird. Wie sich zeigt, eignen sich primär zwei allgemeine Mehrrechner-Architekturen - DB-Sharing /Reu85a, Rah86c/ und DB-Distribution genannt - zur Realisierung von Hochleistungs-DBS. Die kennzeichnende Eigenschaft von DB-Sharing ist, daß jeder Rechner direkt auf die gesamte Datenbank (alle Platten) zugreifen kann, während bei DB-Distribution eine Partitionierung der Externspeicher vorliegt. Ein Vergleich der beiden Architekturklassen läßt erkennen, daß zur Realisierung von Hochleistungs-DBS wichtige Vorteile für DB-Sharing sprechen, obwohl sich bisherige Forschungs-

aktivitäten und Systementwicklungen vorwiegend auf DB-Distribution-Systeme, zu denen auch die verteilten DBS zählen, konzentrierten. DB-Sharing-Systeme erlangten dagegen erst in jüngster Zeit zunehmendes Interesse; die Konzeption und quantitative Bewertung der Synchronisationsverfahren in dieser Arbeit wird daher auch schwerpunktmäßig für diese noch relativ wenig erforschte Systemklasse vorgenommen.
Den Abschluß von Kapitel 2 bildet eine Zusammenstellung von Schlüsselkonzepten, die zur Erreichung hoher TA-Raten und hoher Verfügbarkeit von großer Bedeutung sind.

Im zweiten Teil der Arbeit werden zunächst die für die Synchronisation grundlegenden Annahmen und Definitionen vorgestellt, so der zentrale Begriff der Serialisierbarkeit als weithin akzeptiertes Korrektheitskriterium sowie die Unterscheidung verschiedener Konsistenzebenen. Danach werden in knapper Form die wichtigsten Synchronisationstechniken, die für zentralisierte und verteilte DBS bekannt sind, beschrieben und systematisiert. Eigene Vorschläge in Teil II betreffen vor allem sogenannte optimistische Synchronisationsverfahren.

Der dritte Teil dieser Arbeit widmet sich der Konzeption und qualitativen Beurteilung neuer Synchronisationstechniken für DB-Sharing-Systeme. Hierzu wird zunächst das Systemmodell der betrachteten Architektur präzisiert, und es werden die vielfältigen Beziehungen zwischen den DB-Sharing-Systemkomponenten herausgearbeitet. Eine Klassifizierung der vorgestellten Synchronisationskonzepte unterteilt diese in Sperrverfahren und optimistische Protokolle, die entweder unter zentraler oder verteilter Kontrolle ablaufen können. Bei dem Entwurf der einzelnen Verfahren wird stets das Zusammenspiel zwischen Synchronisation und den anderen Systemkomponenten, vor allem der Systempufferverwaltung, der Recovery-Komponente und der Lastkontrolle, in die Überlegungen einbezogen, um so eine abgestimmte Gesamtkonzeption zu ermöglichen. Dies gilt in besonderem Maße für das vielversprechende Primary-Copy-Sperrverfahren, dessen Realisierung und Zusammenwirken mit anderen Systemfunktionen ausführlich dargestellt werden. Neben den grundlegenden Techniken werden für alle Protokolle auch vielfältige Optimierungsmöglichkeiten und Entwurfsalternativen berücksichtigt. Den Abschluß von Teil III bildet ein qualitativer Leistungsvergleich der vorgestellten Konzepte und Verfahren.

Im vierten Teil, der sich der quantitativen Bewertung der Synchronisationsverfahren widmet, werden zunächst existierende Leistungsanalysen für zentralisierte und verteilte DBS sowie für DB-Sharing-Systeme zusammengestellt und analysiert. Es folgt dann die Darstellung eines umfangreichen und detaillierten Simulationssystems, das die wichtigsten funktionellen Komponenten eines DB-Sharing-Systems nachbildet und die quantitative Bewertung der neu entworfenen Synchronisationsverfahren zuläßt. Die Beschreibung und Analyse der Simulationsergebnisse konzentriert sich dabei v.a. auf das Primary-Copy-Sperrverfahren sowie auf optimistische Protokolle, die auf einer Token-Ring-Topologie basieren.

Am Ende werden die wesentlichen Erkenntnisse und Beobachtungen der vorliegenden Untersuchung zusammengefaßt und eine Reihe von Schlußfolgerungen und Empfehlungen gegeben.

2. Mehrrechner-Datenbanksysteme

Nachdem die wichtigsten Anforderungen an Mehrrechner-DBS in der Einleitung spezifiziert wurden, soll hier zunächst eine Klassifikation grundlegender Mehrrechner-Architekturen vorgenommen werden. In Abschnitt 2.2 folgt dann ein qualitativer Vergleich zwischen dem DB-Sharing- und dem DB-Distribution-Ansatz mit Hinblick auf die Realisierung von Hochleistungs-DBS. Danach werden in 2.3 noch eine Reihe von Schlüsselkonzepten zur Realisierung von TA-Systemen hoher Leistungsfähigkeit angegeben.

2.1 Klassifikation von Mehrrechner-DBS

Zentrale Merkmale unseres in Abb. 2.1 gezeigten Klassifikationsschemas /HäRa86/ sind die Kriterien 'Zugriffsarten auf Externspeicher' sowie 'Rechnerkopplung', die zu einer Separierung von drei allgemeinen Architekturklassen für Mehrrechner-DBS /Sto86, Reu86b/ führen:

1. Eng gekoppelte Mehrrechner-DBS (shared memory)
2. DB-Sharing (shared disk)
3. DB-Distribution (shared nothing).

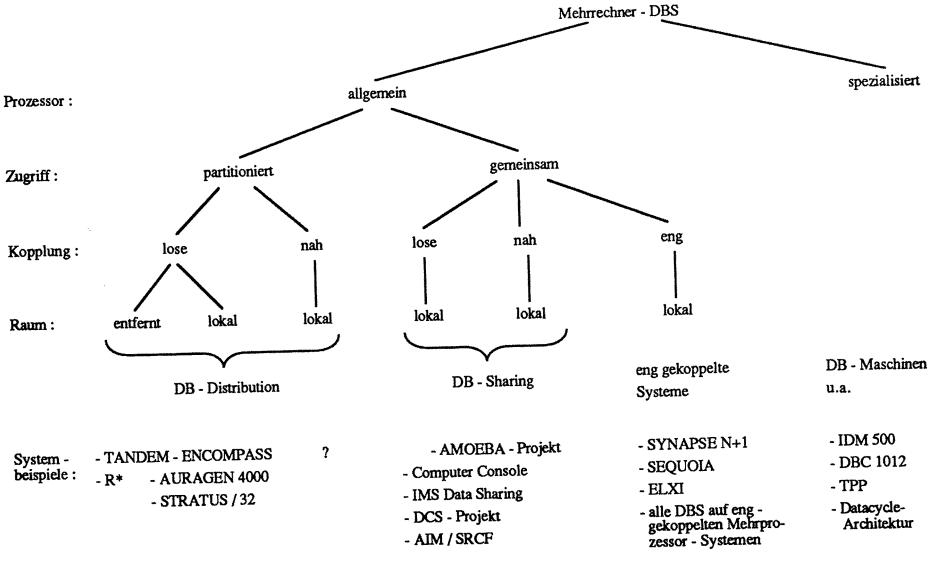


Abb. 2.1: Klassifikationsschema für Mehrrechner-Datenbanksysteme

Diese Klassenbildung wird nun durch die Beschreibung der einzelnen Kriterien näher charakterisiert, um so die wesentlichen Unterschiede sowie eine verfeinerte Aufteilung innerhalb der Klassen herauszuarbeiten. Bei der **Wahl der Prozessoren** konzentrieren wir uns dabei auf allgemeine Verarbeitungsrechner im Gegensatz zu Spezialprozessoren, wie sie typischerweise in **Datenbankmaschinen** /Amb85, BoRe85, Hsi83, Mal84/ eingesetzt werden. Denn mit Spezialrechnern, die eine funktionsorientierte Verteilung der Last verlangen, wird generell die Last-Balancierung, die Verfügbarkeit im Fehlerfall sowie modulares Wachstum wesentlich schwieriger als in homogen aufgebauten Architekturen. Außerdem sind DB-Maschinen meist auf die parallele Bearbeitung mächtiger

Operationen (Suchfrage, Verbund) ausgelegt, während in TA-Systemen typischerweise sehr kurze DB-Operationen mit einfachem Satzzugriff dominieren. Neuere Vorschläge zur Realisierung von Hochleistungs-TA-Systemen unter Nutzung von Spezialprozessoren verkörpern der 'Transaction Pipeline Processor' (TPP /Reu85b/) sowie der sogenannte Datacycle-Ansatz /HeGo87/.

Bei der **Externspeicheranbindung** unterscheiden wir zwischen partitioniertem und gemeinsamem Zugriff:

Partitionierter Zugriff

Beim partitionierten Zugriff, der die Klasse der DB-Distribution-Systeme kennzeichnet, ist jedes Plattenlaufwerk genau einem Rechner zugeordnet; es findet also eine Partitionierung der Plattenperipherie und der darauf gespeicherten Daten statt. Da ein Rechner nur zu seiner Partition direkten Zugriff hat, ist die TA-Verarbeitung in der Regel verteilt (Austauschen externer Daten, Verschicken von DB-Operationen). Die Plattenzuordnung ist prinzipiell statisch; sie kann zwar geändert werden - z.B. beim Rechnerausfall oder bei Konfigurationsänderung -, jedoch ist damit ein erheblicher Aufwand verbunden (Multi-Ports, Verlegen von Kabeln u.ä.).

Die Partitioniering der Plattenlaufwerke impliziert nicht notwendigerweise die strikte Partitionierung der Datenbank; vielmehr können die Daten teilweise oder vollkommen repliziert an mehreren bzw. allen Knoten gespeichert werden. Eine solche Replikation ist jedoch allenfalls in verteilten DBS /BEKK84/ - also ortsverteilten DB-Distribution-Systemen - von Bedeutung (höhere Verfügbarkeit, insbesondere bei 'Katastrophen'; schnellerer Lesezugriff), wenngleich sie sich bei einer hohen Änderungsintensität leistungsmindernd auswirken kann (Aufwand zur Aktualisierung der Replikate). Im rein lokalen Fall dagegen machen schnelle Kommunikationsverbindungen die Führung von Replikaten für effiziente Abfrageoperationen i.a. überflüssig; Fehlertoleranz gegenüber Externspeicherausfällen läßt sich einfacher durch Spiegelplatten erzielen. In lokal angeordneten DB-Distribution-Systemen gehen wir daher von einer strikten DB-Partitionierung (Datenverteilung) aus.

Gemeinsamer Zugriff

Hierbei hat jeder Prozessor direkten Zugriff auf alle Platten (Daten); die erforderliche Kanalkopplung impliziert eine lokale Aufstellung aller Rechner. Wegen der Erreichbarkeit aller Daten kann eine TA beim gemeinsamen Zugriff (DB-Sharing bzw. enge Kopplung) prinzipiell vollständig in einem Rechner bearbeitet werden. Während die Koordinierung der Plattenzugriffe bei der engen Kopplung über die gemeinsam benutzte Betriebssystemkopie erfolgen kann, sollte bei DB-Sharing die Synchronisierung und Optimierung der Zugriffsbewegungen in den (intelligenten) Plattenkontrollern erfolgen. In den heutigen Computer-Systemen können jedoch meist nur bis zu acht Rechner (über Multi-Ports) an eine Platte angeschlossen werden, neben den hohen Kosten des Mehrfachanschlusses v.a. durch die meist relativ langsamen Kanalkopplungen (3-5 MB/s) verursacht. Durch Verwendung sehr schneller Lichtwellenleiter lassen sich jedoch solche Beschränkungen weitgehend aufheben, wobei natürlich zur Vermeidung von Engpässen die Daten auf eine ausreichende Menge von Plattenlaufwerken zu verteilen sind. Eine flexible Plattenanbindung für mehr als acht Rechner wird bereits bei den DEC VAX-Clustern /KLS86/ angeboten, wobei anstelle der üblichen Kanalbefehle eine nachrichtenorientierte E/A-Schnittstelle verfolgt wird. Dabei werden alle Plattenzugriffe über spezielle Platten-Server abgewickelt, von denen allerdings zur Umgehung von Engpässen eine ausreichende Anzahl vorliegen muß. Über die damit erreichbaren Zugriffszeiten, die natürlich in etwa denen bei direkter Plattenanbindung entsprechen müssen, werden jedoch keine Angaben gemacht.

Die **Rechnerkopplung** hat weitreichende Auswirkungen auf die Kooperation und Kommunikation zwischen Rechnern und Prozessen. Dabei unterscheiden wir zwischen enger (tightly coupled), loser (loosely coupled) und naher (closely coupled) Kopplung (*):

Enge Kopplung

Bei der engen Kopplung teilen sich alle Rechner einen gemeinsamen Hauptspeicher, und es existiert nur eine Kopie von Betriebssystem (BS) und Datenbankverwaltungssystem (DBVS). Der gemeinsame Hauptspeicher erlaubt eine sehr effiziente Kooperation/Kommunikation zwischen den Prozessoren, und ein 'single system image' ist von vorneherein gegeben (nur eine BS-Kopie).

Allerdings verursachen gemeinsamer Hauptspeicher und Kommunikationspfade oft schon bei wenigen Rechnern Engpässe, so daß meist nur 2 bis 4 Rechner eng gekoppelt werden. Zwar werden zur Reduzierung der Hauptspeicherzugriffe üblicherweise schnelle Cache-Speicher in jedem Prozessor vorgesehen, doch wird damit das Problem der Cache-Invalidierungen eingeführt: Änderungen in einem Cache hinterlassen veraltete Objektkopien in den anderen Caches bzw. im Hauptspeicher. Die Lösung dieses Cache-Invalidierungsproblems, zu dem eine Vielzahl von Vorschlägen existieren /ArBa86, BiDe86, YYF85/, erfordert aber meist zusätzliche Kommunikationen und verkompliziert die Behandlung von Prozessorausfällen. Ein großer Nachteil der engen Kopplung liegt auch in der Gefahr der Fehlerfortpflanzung (gemeinsamer Hauptspeicher, nur eine Kopie von BS und DBVS), welche die Erfüllung der skizzierten Verfügbarkeitsanforderungen i.a. unmöglich macht.

In letzter Zeit sind eine Reihe von eng gekoppelen Systemen entwickelt worden, mit denen die Fehlertoleranz sowie Erweiterbarkeit socher Systeme verbessert werden soll (z.B. Synapse /Jon83, Ong84, NeIn85/, Seqouia /Ber86/ oder Elxi /OKS83, Ols85, San86/). Jedoch verfügen auch diese Systeme im besten Fall über etwa 50 MIPS Gesamtkapazität, so daß sie die hohen Durchsatzanforderungen nicht erfüllen können. Aus den genannten Gründen kommen die eng gekoppelten Systeme zur Realisierung von Hochleistungssystemen nicht wirklich in Betracht. Sie können jedoch als Knoten innerhalb von DB-Sharing- oder DB-Distribution-Systemen verwendet werden.

Lose Kopplung

Hierbei besitzt jeder Rechner einen eigenen Hauptspeicher und eine separate Kopie von BS und DBVS (autonome Rechner). Die Kommunikation zwischen den Rechnern geschieht ausschließlich über Nachrichten, die über Leitungen ausgetauscht werden. Die Platten können von den Rechnern privat (DB-Distribution) oder gemeinsam (DB-Sharing) benutzt werden.

Da kein gemeinsamer Hauptspeicher mehr als Engpaß vorhanden ist, verspricht die lose Kopplung den Vorteil einer besseren Erweiterbarkeit (größere Anzahl von Rechnern). Die höhere Entkopplung durch die eigenen BS-Kopien bzw. lokalen Hauptspeicher läßt auch eine erhöhte Verfügbarkeit gegenüber der engen Kopplung zu. Fehler in der BS-Software haben nur noch lokale Auswirkungen.

Hauptnachteil der losen Kopplung ist die teuere Kommunikation zwischen den Rechnern, die es selbst bei hohen Bandbreiten nicht erlaubt, synchron (d.h. ohne die CPU freizugeben) auf eine

---

* Eng gekoppelte Systeme werden auch häufig als Mehr- oder Multi*prozessor*-Systeme bezeichnet, zur Abgrenzung von Mehr*rechner*-Systemen (bzw. Multicomputer-Systemen), welche aus einer Menge autonomer Rechnerknoten bestehen /Sch83/. Bei den Mehrrechner-Systemen wird in /Neh87/ weiter unterschieden zwischen nachrichtengekoppelten, speichergekoppelten und hybrid gekoppelten Systemen. Während lose gekoppelte Systeme den nachrichtengekoppelten Mehrrechner-Systemen entsprechen, sind hybrid gekoppelte Systeme in unserer Terminologie der 'nahen Kopplung' zuzurechnen.

Antwort zu warten (-> Prozeßwechsel). Zur Eingrenzung des Kommunikationsaufwandes sollten daher neben einem schnellen Kommunikationssystem auch effiziente Kommunikationsprimitive und billige Prozeßwechsel zur Verfügung stehen. Zusätzlich ist die Nachrichtenanzahl durch den Entwurf geeigneter Algorithmen zu reduzieren.

Nahe Kopplung
Ziel einer nahen Rechnerkopplung ist es, die Vorteile der engen und der losen Kopplung zu vereinen, ohne die jeweiligen Nachteile in Kauf zu nehmen. So soll unter Beibehaltung einer ausreichenden Verfügbarkeit und Erweiterbarkeit - zumindest für Teilaufgaben - eine ähnlich effiziente Interprozessor-Kommunikation wie bei der engen Kopplung erreicht werden. Dazu verfügt, wie bei der losen Kopplung, jeder Rechner über einen eigenen Hauptspeicher sowie eine eigene BS- und DBVS-Kopie. Eine effiziente Kommunikation kann z.B. über gemeinsam benutzte (möglicherweise nichtflüchtige) Halbleiter-Speicherbereiche oder spezielle Hardware-Einheiten (z.B. zur Synchronisation) geschehen. Dabei sollte eine Antwort auf eine Anfrage in wenigen Mikrosekunden möglich sein, um ein synchrones Warten ohne Prozeßwechsel zu erlauben.

Die nahe Kopplung setzt voraus, daß alle Rechner in räumlicher Nachbarschaft angeordnet sind. Für DB-Distribution erscheint eine solche Kopplung weniger erstrebenswert, weil damit die 'shared nothing'-Eigenschaft, die Voraussetzung für eine mögliche Ortsverteilung ist, verletzt würde. Bei DB-Sharing dagegen eröffnen sich durch die nahe Kopplung eine Reihe vielversprechender Optimierungsmöglichkeiten (z.B. Erweiterung der Speicherhierarchie um einen globalen Systempuffer oder für Synchronisationszwecke /Rah86b, HäRa87/). In Kapitel 10.4 wird die nahe Kopplung bei DB-Sharing noch genauer diskutiert werden. Zumeist gehen wir jedoch in dieser Arbeit von einer losen Rechnerkopplung aus, die prinzipiell die beste Verfügbarkeit und Erweiterbarkeit gestattet.

Unser letzter Klassifikationspunkt, der schon mehrfach angesprochen wurde, betrifft die **räumliche Aufstellung der Rechner**. So ist eine ortsverteilte Anordnung der Rechner, die nur relativ geringe Bandbreiten (z.B. 1-100 KB/s) zuläßt, nur bei DB-Distribution möglich (verteilte DBS). Die Realisierung von Hochleistungs-DBS verlangt jedoch ein sehr schnelles Kommunikationssystem, so daß hierzu praktisch nur lokale Mehrrechner-DBS (z.B. mit 1-100 MB/s) in Frage kommen.

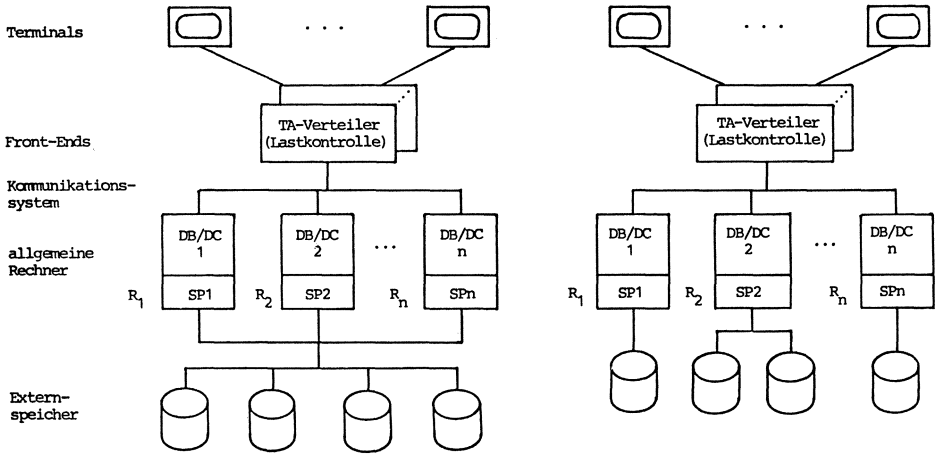
Nachdem wir die eng gekoppelten Systeme schon weiter oben ausgeklammert haben, stellt sich nun vor allem die Frage nach der Tauglichkeit des DB-Sharing- bzw. des DB-Distribution-Ansatzes. Dieser Frage soll im nächsten Abschnitt nachgegangen werden.

Abschließend sei noch darauf hingewiesen, daß mit der vorgestellten Klassifikation von Mehrrechner-DBS keineswegs alle Realisierungsformen verteilter TA-Systeme erfaßt sind, weil diese auch durch eine Verteilung innerhalb des DC-Systems realisierbar sind /Häu85, Mey87/. Besonderheiten bei heterogenen Mehrrechner-DBS /GlLu84,GlPo86/, bei denen auf den einzelnen Rechnern DBVS und BS unterschiedlichen Typs laufen können, werden in dieser Arbeit ebenfalls nicht betrachtet.

2.2 DB-Sharing vs. DB-Distribution

Die Grobstruktur von DB-Sharing- und DB-Distribution-Systemen ist in Abb. 2.2 dargestellt. Wir gehen hierbei, wenn nichts anderes gesagt wird, von einer lokalen Anordnung sowie einer losen Kopplung der Rechner aus. Die räumliche Nachbarschaft der Rechner erlaubt nicht nur schnelle Kommunikationsverbindungen, sondern auch eine Terminal-Anbindung über mehrere Front-Ends, die

gewisse Aufgaben des DC-Systems wie etwa Verteilung der TA und der Lastkontrolle übernehmen. Für die Bearbeitung einer TA kann dabei prinzipiell jeder Rechner ausgewählt werden; die Auswahl des Rechners kann so (im Gegensatz zu ortsverteilten Systemen) z.B. auch unter Berücksichtigung der aktuellen Rechnerauslastung erfolgen.



DB-Sharing **DB-Distribution**

Abb. 2.2: Grobarchitektur von DB-Sharing- und DB-Distribution-Systemen

Wie angedeutet verfügt jeder Rechner über einen eigenen Systempuffer (SP) sowie eine eigene DC-Komponente. Der entscheidende Unterschied zwischen beiden Architekturklassen liegt in der Zuordnung der Externspeicher zu den Rechnern:

- Bei **DB-Sharing** können alle Rechner (DBVS) direkt auf alle Externspeicher und damit auf alle Daten der DB zugreifen.
- Bei **DB-Distribution** verwaltet jeder Rechner bzw. jedes DBVS eine Partition der DB und kann jeweils nur auf seine Partition direkt zugreifen. Daten aus anderen Partitionen müssen explizit angefordert und ausgetauscht werden.

Aus der Art der Externspeicheranbindung ergeben sich weitreichende Konsequenzen bezüglich der Tauglichkeit, die in der Einleitung genannten Anforderungen an Hochleistungs-DBS erfüllen zu können als auch für die Realisierung einzelner Systemkomponenten. Bisherige Vergleiche der beiden Architekturklassen /Tra83,Sho86,Sto86,Reu86b,CDY86/ führten teilweise zu gegensätzlichen Einschätzungen, vor allem dadurch bedingt, daß oft wichtige Aspekte unberücksichtigt blieben. Die folgende Gegenüberstellung beider Systemarchitekturen stützt sich auf der Untersuchung /HäRa87/ (die wiederum auf /Här86, HäRa86/ basiert) ab, in der eine ausführliche Bewertung hinsichtlich einer Vielzahl von Kriterien vorgenommen wurde.

Für den DB-Distribution-Ansatz sprechen im wesentlichen zwei Gesichtspunkte: die Rechner können sowohl lokal als geographisch verteilt angeordnet sein, und es kann bei der Realisierung auf vielzählige Erkenntnisse und Lösungsvorschläge aus dem Gebiet der verteilten DBS zurückgegriffen werden.

Die **Ortsverteilung** der Rechner erlaubt es, in großen Unternehmen eine Anpassung an lokale Bedürfnisse vorzunehmen (Systemleistung vor Ort), und sie ist Voraussetzung für eine schnelle *Katastrophen-Recovery*. Hierzu kann man nämlich alle Daten vollständig repliziert in zwei (oder

mehr) geographisch verteilten Rechenzentren führen, so daß im Katastrophenfall die Verarbeitung am überlebenden Zentrum fortgesetzt werden kann. Im Normalbetrieb wird die TA-Last unter beiden Zentren aufgeteilt, und die Änderungen werden untereinander ausgetauscht /GrAn85/.

Die **Anzahl verfügbarer Lösungsvorschläge und Implementierungen** spricht auch klar für DB-Distribution, da das Gebiet der verteilten DBS seit über 10 Jahren intensiv bearbeitet wird. Zumindest einfache verteilte DBS werden bereits von vielen DBS-Hersteller angeboten, so daß die Erstellung leistungsfähigerer Systeme ggf. auf vorhandener Software aufbauen kann. Als Paradebeispiel steht hier Tandem, das sein Encompass-System /Bor84, Hel85a/ zunehmend in Richtung 'Hochleistungseigenschaft' entwickelt /Hel85b/.

DB-Sharing dagegen ist ein relativ neuer Ansatz. Eine erste Realisierung stellt das auf zwei Rechner beschränkte IMS Data Sharing /IMS81, Kee82, SUW82/ dar, das aber keine Hochleistungsfähigkeiten aufweist. Neuere System- und Prototyp-Entwicklungen streben jedoch verbesserte Fehlertoleranz- und Leistungsmerkmale an; zu nennen sind hier Computer Console's Power System 5/55 /WIH83/, das AIM/SRCF-System (Shared Resource Control Facility /AIM86/), DCS (Data Sharing Control System /Sek84/) sowie das Amoeba-Projekt /Tra83, Sho85/. Die DEC VAX-Cluster /KLS86/, von denen 1985 bereits rund 2000 Installationen existierten, ähneln zwar auch der DB-Sharing-Architektur, jedoch handelt es sich dabei um kein Datenbanksystem.

Wie sieht es aber mit der Eignung hinsichtlich der Erfüllung der zentralen Anforderungen an Hochleistungs-DBS - also hohe TA-Raten, hohe Verfügbarkeit, Erweiterbarkeit und Handhabbarkeit - aus? Wie die Diskussion zeigen wird, ergeben sich hier zumindest für die drei zuletzt genannten Punkte deutliche Nachteile für DB-Distribution, die alle auf die Notwendigkeit der Datenpartitionierung zurückzuführen sind. Denn da diese nur selten und mit hohem Aufwand geändert werden kann, wird die Flexibilität des Systems, auf Änderungen zu reagieren, stark eingeschränkt.

Verfügbarkeit: Bei DB-Sharing können nach Ausfall eines Rechners die überlebenden Prozessoren weiterhin alle Daten erreichen, so daß auf ihnen nach bestimmten Recovery-Maßnahmen die gescheiterten TA wiederholt und die laufende TA-Last gleichmäßig aufgeteilt werden können. DB-Distribution dagegen erfordert nach erfolgreicher Recovery die Übernahme der ausgefallenen Partition durch die überlebenden Rechner, um die Erreichbarkeit der Daten zu gewährleisten (Anbindung jeder Platte an mindestens zwei Rechner). Da i.a. ein Rechner die gesamte Partition des ausgefallenen Rechners übernehmen muß, wird dieser während der Ausfallzeit leicht überlastet. Eine ungünstige Lastverteilung führt zwangsläufig zu Leistungseinbußen.

Erweiterbarkeit: Bei DB-Distribution erzwingt die Hinzunahme eines Rechners die Neuaufteilung der Datenbank (N -> N+1), was nur sehr umständlich (Änderung der Externspeicherzuordnung) und oft nicht im laufenden Betrieb möglich ist. Eine ausreichende Flexibilität zur Änderung der Datenverteilung bieten zudem nur relationale DBS (horizontale oder vertikale Partitionierung von Relationen), da in hierarchischen und netzwerkartigen Datenmodellen i.a. nur sehr grobe Verteileinheiten möglich sind (z.B. Segmente, Areas, Satztypen). Hier können Umverteilungen sogar Programmänderungen zur Folge haben (*).

---

* Ein weiterer Punkt, der gegen die Verwendung nicht-relationaler DBS bei DB-Distribution spricht, ist der zu erwartende hohe Kommunikationsaufwand. Denn mit den satzorientierten DML-Befehlen, die möglicherweise noch von mehreren Rechnern zu bearbeiten sind, ist der Kommunikations-Overhead für eine entfernte Bearbeitung oft höher als die Verarbeitungskosten der Operation selbst /Sto80/.

Bei DB-Sharing dagegen wird ein neuer Rechner mit allen Platten verbunden; es ist keine Datenverteilung vorzunehmen. Insbesondere ist der Übergang von einem zentralisierten System zu DB-Sharing ohne Änderung existierender Datenbanken oder Anwendungsprogramme möglich, unabhängig davon, welches Datenmodell eingesetzt wird. Allerdings erschwert die gemeinsame Plattenanbindung bei DB-Sharing den Einsatz sehr vieler Rechner, was jedoch bei Verwendung leistungsfähiger Knoten (z.B. 20-50 MIPS) keine Einschränkung für die nächste Zukunft bedeutet. Außerdem ist auch bei DB-Distribution wegen der vorzunehmenden Datenverteilung meist nur eine begrenzte Rechneranzahl sinnvoll einsetzbar. In einer analytischen Untersuchung /DIY86/ wurde ermittelt, daß es unter Leistungsgesichtspunkten ohnehin ratsamer ist, eine kleinere Anzahl (<= 10) leistungsfähiger Rechner zu koppeln als eine größere Anzahl kleinerer Rechner. Denn bei langsameren Rechnern verlängert sich die Antwortzeit der TA und damit werden mehr Synchronisationskonflikte verursacht (Sperren werden länger gehalten u.ä.), was wiederum zu Durchsatzeinbußen und weiteren Antwortzeitverschlechterungen führt. Unsere Überlegungen für DB-Sharing in Teil III und IV der Arbeit orientieren sich daher an Systemen mit relativ wenigen, aber leistungsstarken Rechnern.

Das Hinzufügen neuer Platten/Daten ist bei DB-Sharing auch wesentlich einfacher als bei DB-Distribution. Vor allem braucht keine Entscheidung darüber getroffen zu werden, welchem Rechner die neuen Platten/Daten zuzuordnen sind.

Handhabbarkeit: Hier ergeben sich ebenfalls Nachteile für DB-Distribution, da die Erstellung und Änderung der Datenpartitionierung (physischer DB-Entwurf) sehr schwierig und nur teilweise automatisierbar ist. Daher läßt sich auch die Verteilung der Daten gegenüber Systemverwalter oder Datenbankadministrator i.a. nicht verbergen (kein 'single system image'). Die ortsverteilte Anordnung der Rechner kann zwar organisatorische Vorteile mit sich bringen, andererseits verkompliziert sich damit die Verwaltung des Gesamtsystems, und in der Regel ist in jedem Knoten qualifiziertes Bedienpersonal vorzusehen.

Eine Beurteilung hinsichtlich der Erreichbarkeit hoher TA-Raten und kurzer Antwortzeiten ist auf der gewählten Betrachtungsebene natürlich nur schwer möglich, da hier die Realisierung der einzelnen Systemfunktionen sowie die Charakteristika der jeweiligen TA-Last entscheidenden Einfluß haben. Die Realisierung des Systems ist dabei auch entscheidend von der Art der **TA-Bearbeitung** abhängig:

Bei *DB-Distribution* können nur Daten der lokalen Partition direkt referenziert werden, externe Daten werden durch Daten- oder Funktionsaustausch besorgt (I/O-request shipping vs. function-request/ DB-call shipping /YCDT86/). Die übliche Vorgehensweise ist hierbei das Verschicken von DB-Operationen, die dann durch eigens erzeugte Sub-TA abgearbeitet werden. Bei TA-Ende sind alle Sub-TA in ein rechnerübergreifendes Mehr-Phasen-Commit-Protokoll (z.B. 2PC) einzubeziehen, um die Atomizität der Gesamt-TA zu gewährleisten. Verteilt ist auch das Zurücksetzen einer TA, die auf externe Daten zugegriffen hat.

Im Gegensatz dazu kann bei *DB-Sharing* jede DB-Operation einer TA vollständig lokal bearbeitet werden, da alle Daten direkt erreichbar sind. Kommunikation wird hier im wesentlichen nur zur Synchronisation benötigt sowie zum Holen von Daten aus fremden Systempuffern. Die rechnerübergreifende Synchronisation der Zugriffe auf die gemeinsame Datenbank ist bei DB-Sharing offensichtlich notwendig, um die Konsistenz der Daten und die Ablaufintegrität zu bewahren. Das Holen von Seiten aus fremden Puffern ist eine Folge des bei DB-Sharing zu lösenden **Veralterungs-**

problems (buffer invalidation), ein analoges Problem zur Cache-Invalidierung bei eng gekoppelten Multiprozessoren oder der Aktualisierungsproblematik bei replizierten Datenbanken /Rah86a/. Denn die Änderung einer Seite in einem Systempuffer invalidiert die Kopien dieser Seite in anderen Puffern und auf der physischen Datenbank. Da der Zugriff auf veraltete Seiten natürlich zu verhindern ist, müssen ggf. Änderungen aus fremden Puffern angefordert werden.

Die unterschiedliche TA-Bearbeitung ist bei der Realisierung der auf den Front-Ends angesiedelten **Lastkontrolle** zu berücksichtigen. Die zentralen Aufgaben der (globalen) Lastkontrolle /Reu86a, Rah86d, YBL86, CaLu86, WaMo85/ sind Verteilung und Balancierung der Last, Verhinderung von Überlastsituationen sowie größtmögliche Ausnutzung und Bewahrung von Lokalität. Eine neue TA ist demjenigen Rechner zuzuordnen, bei dem sie einen Großteil der benötigten Daten, Sperren u.ä. vorfindet und daher mit möglichst wenigen Interprozessor-Kommunikationen bearbeitbar ist. Für DB-Sharing bedeutet das, den Rechner zu bestimmen, an dem v. a. eine weitgehend lokale Synchronisation möglich ist. Für DB-Distribution ist der Rechner zu suchen, an dem die meisten der benötigten Daten direkt zugreifbar sind.

Die Zuordnung einer TA zu einem Rechner (TA-Routing), die i.a. anhand des TA-Typs und ggf. der aktuellen Parameter erfolgt, ist für *DB-Distribution* vergleichsweise einfach, da hier die aktuelle Datenverteilung stets bekannt ist. Die statische Verteilung der Daten bedingt allerdings auch eine geringe Flexibilität für die Lastkontrolle, so daß größere Schwankungen im Lastprofil - wie sie oft mehrmals täglich vorkommen - oder ein Rechnerausfall weit weniger gut verkraftet werden als bei DB-Sharing. Desweiteren ist die Rechnerauslastung bereits durch die Datenverteilung weitgehend bestimmt, weil ein Rechner alle Operationen (lokaler und externer) TA auf die ihm zugeordnete Datenpartition durchführen muß. Aufgrund der Lastschwankungen sind somit ungleichmäßige Rechnerauslastungen und die damit verbundenen Leistungseinbußen vorprogrammiert. Auch das Ausmaß der Interprozessor-Kommunikationen ist durch die Datenverteilung schon weitgehend festgelegt, da mit der sogenannte Query-Optimierung /BEKK84,YuCh84/ nur für mengenorientierte Operationen Einsparmöglichkeiten erreichbar sind, in kommerziellen TA-Systemen jedoch Einzelsatzzugriffe dominieren. Außerdem kann die Query-Optimierung die aktuelle Rechnerauslastung i.a. ohnehin nicht berücksichtigen, da sie aus Effizienzgründen meist schon zur Übersetzungszeit der TA-Programme vorgenommen wird (*).

Bei *DB-Sharing* dagegen bestehen wesentlich mehr Freiheitsgrade bezüglich der Lastverteilung und damit hinsichtlich der Minimierung von Interprozessor-Kommunikationen und der Last-Balancierung. Denn jede DB-Operation und somit jede TA kann von jedem der Rechner ausgeführt werden (es wird unterstellt, daß die TA-Programme in jedem Rechner verfügbar sind); ein Objekt kann parallel in verschiedenen Rechnern gelesen werden (Replikation im Puffer). Die DB-Sharing-Architektur erlaubt es sogar bei Unterlast ganze Rechner für andere Aufgaben 'abzustellen'. Dies scheint besonders wichtig, da das System für die Spitzenanforderungen ausgelegt sein muß, die meiste Zeit jedoch weitaus geringere TA-Raten zu bewältigen sind.

Um die Flexibilität und Optimierungsmöglichkeiten der DB-Sharing-Architektur aber nutzen zu

---

* Noch weniger Möglichkeiten einer globalen Lastkontrolle bestehen in verteilten DBS (ortsverteilten DB-Distribution-Systeme), da dort Terminals typischerweise einem Rechner fest zugeordnet sind (kein dynamisches TA-Routing). Verteilentscheidungen unter Berücksichtigung der aktuellen Rechnerauslastung sind in solchen Systemen schon wegen den großen Entfernungen kaum möglich.

können, ist nicht nur eine geeignete Realisierung für die Lastkontrolle, sondern auch für Logging/ Recovery und vor allem für die Synchronisationskomponente erforderlich. Denn das verwendete Synchronisationsprotokoll bestimmt bei DB-Sharing entscheidend den zur TA-Bearbeitung erforderlichen Kommunikationsaufwand. Wie sich zeigen wird, sollte zur Minimierung des Nachrichtenbedarfs auch das Veralterungsproblem zusammen mit der Synchronisation behandelt werden und eine effektive Zusammenarbeit mit der Lastkontrolle möglich sein. Mit dem Entwurf und der quantitativen Bewertung von Synchronisationsprotokollen für DB-Sharing, die diese und weitere Anforderungen (siehe Kap. 3.2) erfüllen, beschäftigen wir uns ausführlich in Teil III und IV dieser Arbeit.

Auch bei der Realisierung von DB-Distribution-Systemen sind einige besondere Schwierigkeiten zu bewältigen. Die Hauptprobleme liegen hier im physischen DB-Entwurf (Datenpartitionierung /SaWi85/), in der im Vergleich zu zentralisierten Systemen und auch DB-Sharing komplexeren Anfragebearbeitung /Moh84/, welche die Datenverteilung berücksichtigen muß, und - damit verbunden - in der Verwaltung verteilter Kataloge bzw. Dictionaries /Gra86b/. In verteilten DBS ist auch die Fehlerbehandlung i.a. weit schwieriger als in lokalen Systemen, in denen vor allem das Kommunikationssystem zuverlässiger ausgelegt werden kann. So ist in verteilten DBS z.B. mit Netzwerk-Partitionierungen zu rechnen, wobei Teile des Systems nicht mehr miteinander kommunizieren können; andere Probleme werden in /Str85/ diskutiert.

Die Ausführungen in diesem Kapitel haben gezeigt, daß zur Realisierung von Hochleistungs-DBS viele Punkte für DB-Sharing sprechen. Der DB-Sharing-Ansatz kann als natürliche Weiterentwicklung von zentralisierten DBS auf Mehrrechner-DBS angesehen werden, da beim Übergang auf ein DB-Sharing-System keine Änderung bestehender Anwendungsprogramme und Datenbanken notwendig ist. Mit DB-Sharing wird nicht nur eine einfachere Adaption an eine verminderte oder erhöhte Anzahl von Rechnern als bei DB-Distribution möglich, sondern auch eine flexiblere Anpassung an Lastschwankungen. Die größeren Möglichkeiten zur Last-Balancierung und zur Reduzierung des Kommunikationsaufwandes bieten ein höheres Optimierungspotential zur Erreichung eines hohen Durchsatzes und kurzer Antwortzeiten, während die Leistungsfähigkeit eines DB-Distribution-Systems bereits mit der statischen Datenpartitionierung weitgehend festgelegt ist. Zusätzliche Leistungsvorteile dürften sich bei DB-Sharing durch den Einsatz einer nahen Rechnerkopplung ergeben. Schließlich ist die Realisierung eines Hochleistungs-DBS mit DB-Sharing nicht nur auf relationale Systeme beschränkt, sondern auch mit netzwerkartigen und hierarchischen Datenmodellen möglich.

2.3 Schlüsselkonzepte zur Realisierung von Hochleistungs-DBS

Zur Abrundung der allgemeinen Diskussion über Mehrrechner-DBS soll in diesem Abschnitt noch eine Zusammenstellung wesentlicher Konzepte und Techniken zur Erreichung hoher Leistungsfähigkeit und hoher Verfügbarkeit vorgenommen werden.

2.3.1 Techniken zur Erlangung hoher TA-Raten und kurzer Antwortzeiten

Hierzu wurden für Mehrrechner-DBS bereits einige zentrale Punkte angesprochen, nämlich die **Begrenzung des Kommunikations-Overheads**, die Bedeutung der **Lastkontrolle** (Lastverteilung und -Balancierung) sowie mögliche Optimierungen über eine **nahe Rechnerkopplung**. Wie wir gesehen haben, bietet vor allem die DB-Sharing-Architektur bezüglich dieser Punkte ein großes Optimierungs-

potential, sofern sich geeignete Algorithmen etwa bei der Synchronisation oder der Lastkontrolle finden lassen. Bei der Entwicklung solcher Algorithmen ist insbesondere auf die Minimierung von 'synchronen' Nachrichten zu achten, für die eine TA bis zum Eintreffen einer Antwortnachricht unterbrochen werden muß (Antwortzeitverschlechterung). Generell sind zur Begrenzung des Kommunikationsaufwandes ein schnelles Kommunikationssystem wichtig sowie effiziente Kommunikationsprimitive und geringe Prozeßwechselkosten. Einsparungen sind auch möglich durch Bündelung und gemeinsame Übertragung von Nachrichten (group request), jedoch i.a. zu Lasten der Antwortzeiten (Bündelungswartezeiten).

Eine weitere leistungsbestimmende Funktion ist die **Synchronisationskomponente**, nicht nur wegen ihres Einflußes auf den Kommunikationsbedarf. Bei ihrer Realisierung ist auch darauf zu achten, daß eine möglichst hohe Parallelität erhalten bleibt, d.h., das Ausmaß an TA-Blockierungen und -Rücksetzungen sollte so gering wie möglich gehalten werden. Die hierfür in Frage kommenden Synchronisationstechniken werden in den folgenden Kapiteln ausführlich besprochen.

Wie schon in zentralisierten DBS, so ist auch in Mehrrechner-DBS die **Reduzierung des E/A-Aufwandes** von zentraler Bedeutung für die Leistungsfähigkeit des Systems. Eine geringe E/A-Häufigkeit setzt eine auf die Anwendung zugeschnittene Wahl von Speicherungsstrukturen und Zugriffspfaden voraus. So erlauben Hash-Verfahren und B*-Bäume schnellen Schlüsselzugriff auf Datensätze; Clusterbildung gestattet die effiziente Verarbeitung zusammengehöriger Daten. Desweiteren bestimmen vor allem die Realisierung der Systempufferverwaltung und der Log-Komponente den zur TA-Verarbeitung anfallenden E/A-Bedarf:

Bei der **Systempufferverwaltung** kann die Ersetzungsstrategie zur Begrenzung physischer E/A-Vorgänge Lokalität im Referenzverhalten nutzen /EfHä84, HäRe83a/, wobei immer größer werdende Systempuffer (z.B. > 50 MB) die Trefferraten positiv beeinflußen. Einen wichtigen Einfluß auf das E/A-Verhalten übt auch die Vorgehensweise beim Ausschreiben geänderter Datenbankseiten aus. Die sogenannte **NOFORCE-Strategie** /HäRe83b/, bei der Änderungen bei EOT nur gesichert, die geänderten Seiten jedoch nicht in die physische Datenbank ausgeschrieben werden müssen, hilft i.a. E/A einzusparen. Denn damit kann eine Seite mehrfach geändert werden, ohne daß ein Ausschreiben notwendig wird; diese Akkumulierung von Änderungen zahlt sich vor allem bei großen Puffern aus. Desweiteren werden die Antwortzeiten von Änderungs-TA nicht unnötig durch Ausschreibvorgänge erhöht.

Weitere E/A-Einsparungen verspricht man sich von einer erweiterten Speicherhierarchie, insbesondere durch seitenadressierbare Halbleiterspeicher ('expanded storage'), die derzeit Zugriffszeiten um 1 ms erlauben. Solche Halbleiterspeicher werden auch in einigen Plattenkontrollern geführt ('disk cache'), um etwa sequentielle Lesevorgänge durch das Laden einer ganzen Spur von der Platte in den Halbleiterspeicher (prefetching) zu beschleunigen.

Das **Logging** sollte vor allem mittels sequentieller E/A (bzw. chained I/O) anstelle von wahlfreier E/A erfolgen, weil jede wahlfreie E/A die Antwortzeit einer TA um etwa 25-60 ms verlängert. Zur Erreichung hoher TA-Raten (und zur Begrenzung des Log-Umfangs) sollte v.a. ein **eintragsweises Logging** /Reu83/ anstelle von Seiten-Logging durchgeführt werden, weil somit ein effektives **Gruppen-Commit** /Gaw85a,DeW84/ ermöglicht wird. Dabei wird das EOT einer Gruppe von TA so lange verzögert, bis die zugehörigen Log-Daten zusammen ausgeschrieben sind. Diese Technik erlaubt es bei Eintrags-Logging, den Log-Aufwand auf etwa 0.1 - 0.2 Log-E/A pro TA zu begrenzen /Gra85/.

Nichtflüchtige Halbleiterspeicher (solid state disks) lassen sich auch beim Logging gewinnbringend einsetzen /CKS86/, da hiermit das Auschreiben der Log-Daten etwa 10-mal so schnell wie mit sequentieller E/A auf herkömmlichen Platten geschehen kann. Allerdings sind solche 'elektronischen Platten' erheblich teuerer als konventionelle Platten, so daß ihr Einsatz auf besonders zeitkritische Funktionen beschränkt sein dürfte.

Die für Hochleistungssysteme obligatorische NOFORCE-Strategie erzwingt zur Begrenzung des REDO-Aufwandes nach einem Rechnerausfall ein Sicherungspunktverfahren als unterstützende Maßnahme. Da das Ausschreiben aller Änderungen im Systempuffer zur Erstellung des Sicherungspunktes bei den unterstellten großen Puffern zu untolerierbar langen Totzeiten führen würde, sind hierzu sogenannte 'fuzzy checkpoints' /HäRe83b/ die gegebene Alternative.

Hauptspeicher-Datenbanken (main memory databases), die in jüngster Zeit im Mittelpunkt zahlreicher Untersuchungen und Prototyp-Entwicklungen stehen (z.B. /Amm85, DeW84, EiJa86, GLV84, LeRo85, LeCa86/), versprechen die größtmöglichen E/A-Einsparungen. Dabei können nach dem zu Beginn erforderlichen Laden der Datenbank von einer Archivkopie sämtliche DB-Zugriffe im Hauptspeicher befriedigt werden. E/A-Vorgänge sind damit nur noch für Logging-Aktivitäten erforderlich, um nach einem Rechner- oder Hauptspeicher-Ausfall den aktuellen DB-Zustand wiederherstellen zu können. Für Hauptspeicher-Datenbanken sind jedoch eine Reihe von Problemen neu zu lösen, so bei der Recovery /Hag86, Eic87, LeCa87/, der Query-Bearbeitung oder dem Entwurf von geeigneten Speicherungsstrukturen /LeCa86/.

Für unsere weiteren Überlegungen spielen Hauptspeicher-Datenbanken vor allem aus zwei Gründen keine Rolle. Zum einen sind sie primär auf Monoprozessoren oder eng gekoppelte Multiprozessoren begrenzt, mit deren CPU-Kapazität sehr hohe TA-Raten (z.B. > 1000 TA/s) nicht zu erreichen sind. Zum anderen umfassen die Datenbanken großer Anwender oft Hunderte von Giga-Bytes und können daher in absehbarer Zukunft nicht vollkommen im Hauptspeicher gehalten werden. Zudem wachsen der CPU-Bedarf und das Datenvolumen in den hier betrachteten Großanwendungen permanent und oft schneller als der technologische Zuwachs bei der Kapazität einzelner CPUs oder bei der Hauptspeicher-Größe.

2.3.2 Fehlertoleranz-Konzepte

Das TA-Konzept /HäRe83b,Gra80/ als fundamentales Konzept in DB/DC-Systemen enthält bereits wesentliche Zusicherungen hinsichtlich der Maskierung und Behandlung von Fehlern. Die vier kennzeichnenden Eigenschaften (ACID) definieren eine TA als atomare Einheit der Konsistenz, der Isolation und der Recovery. Die 'Alles-oder-Nichts'-Eigenschaft (Atomizität) bietet dem TA-Programm eine Maskierung gegenüber allen vom System erwarteten Fehlern, die Dauerhaftigkeits-Eigenschaft garantiert, daß Änderungen erfolgreicher TA alle erwarteten Fehler überleben. So wird nach Ausfall eines Rechners der jüngste TA-konsistente Datenbankzustand wiederhergestellt, in dem die Änderungen aller erfolgreich beendeten TA reflektiert sind, jedoch keinerlei Auswirkungen von TA, die zum Ausfallzeitpunkt noch in Bearbeitung waren.

Die TA-Eigenschaften spiegeln den TA-Programmen zwar eine fehlerfreie Systemumgebung vor, garantieren jedoch keineswegs eine hohe Verfügbarkeit für den Benutzer /Här87b/, da sie weder das Auftreten von Fehlern verhindern, noch eine unterbrechungsfreie Recovery zusichern. Hohe Verfügbarkeit kann auch bei Ausfall einzelner Komponenten erreicht werden, falls - unter Verwendung redundanter Komponenten - die Ausfallbehandlung so schnell erfolgen kann, daß keine nicht

tolerierbaren Unterbrechungen entstehen. Jedoch ist der Totalausfall des Terminalnetzes unter allen Umständen zu verhindern, da dessen Reaktivierung bei mehreren tausend Terminals erfahrungsgemäß eine Stunde oder länger dauern kann /Gra86a/.

Voraussetzung zur Erlangung einer hohen Fehlertoleranz ist eine ausreichende Redundanz bei allen wichtigen Hardware- und Software-Komponenten /Kim84/. Dabei ist zur Tolerierung von Einfach-Fehlern mindestens eine zweifache Auslegung bei all diesen Komponenten vorzusehen. Neben den bereits in der Einleitung angesprochenen Punkten wie einfache Handhabbarkeit oder dynamische Konfigurierbarkeit, sind für eine hohe Verfügbarkeit auch geeignete Konzepte zur fehlertoleranten Ausführung, zur fehlertoleranten Speicherung und fehlertoleranten Kommunikation anzubieten. Die wichtigsten Aspekte dazu sollen im folgenden kurz aufgeführt werden; eine ausführlichere Diskussion findet sich in /Här87b/ und /Gra86a/. In /Här87b/ werden auch Meßergebnisse an einem fehlertoleranten TA-System (Tandem) vorgestellt, mit denen die durch die Fehlertoleranz-Mechanismen eingeführten Zusatzkosten zum Teil quantifiziert werden konnten.

Fehlertolerante Ausführung

Bei der fehlertoleranten Ausführung von Software unterscheidet man i.a. zwischen drei Konzepten:

- redundante Ausführung
- Prozeß-Paare und
- Schattenprozesse.

Bei der *redundanten Ausführung*, die z.B. von Stratus /Hen83,Kas83/ angewendet wird, wird jede Instruktion parallel in wenigstens zwei CPUs ausgeführt und durch Ergebnisvergleich ein ständiger Selbsttest vorgenommen. Diese Hardware-Maßnahmen zur Fehlertoleranz sind für die gesamte Software transparent, bieten aber auch keine Hilfe gegenüber Software-Fehlern und zunächst auch nicht gegenüber transienten Fehlern (da alle beteiligten Prozessoren dieselbe Systemumgebung sehen). Ein weiterer Nachteil ist, daß die parallel mitlaufenden Prozessoren keine eigene Arbeit verrichten.

Beim Konzept der *Prozeß-Paare*, das bei Tandem und Auragen /Kim84,Ser84/ sowie in verallgemeinerter Form im HAS-Projekt /Agh83,Agh86/ Anwendung findet, soll nach Ausfall des sogenannten Primary-Prozesses ein bis dahin passiver Backup-Prozeß dessen Aufgaben übernehmen (Forward Recovery). Voraussetzung dazu ist, daß der Backup-Prozeß in gewissen Abständen mit ausreichenden Statusinformationen über den Verarbeitungszustand im Primary-Prozeß informiert wird (Checkpointing), was jedoch u.U. sehr teuer werden kann. In Tandem werden Prozeß-Paare daher mittlerweile nur noch für Systemprozesse (etwa im BS-Kern) benutzt /Gra86a/.

Der hohe Checkpointing-Aufwand wird mit *Schattenprozessen* /Här87b/ vermieden, bei denen im Fehlerfall eine gescheiterte TA vollständig zurückgesetzt (Backward Recovery) und danach im Schattenprozeß erneut gestartet wird (*). Bei Protokollierung der Eingabe-Nachrichten durch das DC-System können so zumindest zurückgesetzte Einschritt-TA für den Benutzer transparent wiederholt werden. Bei Mehrschritt-TA ist dies nur möglich, solange die bei der Wiederausführung neu abgeleiteten Nachrichten den ursprünglichen entsprechen /HäMe86b/. Schattenprozesse stellen für die fehlertolerante TA-Verarbeitung das gegebene Konzept, da bei den typischerweise kurzen TA der

---

* Schattenprozesse können im Prinzip als spezielle Form von Prozeß-Paaren aufgefaßt werden, deren kennzeichnende Eigenschaft - die rückwärtsorientierte Recovery-Strategie - einen geringen Checkpointing-Aufwand ermöglicht.

Aufwand einer redundanten Ausführung oder einer Forward Recovery mit Prozeß-Paaren nicht gerechtfertigt erscheint. Das Konzept der Schattenprozesse wird u.a. bei Tandem sowie dem angekündigten XRF (Extended Restart Facility) für IMS verwendet. Bei XRF residieren alle Schattenprozesse in einem passiven Backup-Rechner ('Hot Stand-By'), der nach Ausfall des primären Verarbeitungsrechners dessen Funktion übernimmt /Gaw85b/.
Nach einem Rechnerausfall können die gescheiterten TA erst dann in einem anderen Rechner neu gestartet werden, nachdem die überlebenden Rechner die Crash-Recovery für den ausgefallenen Prozessor durchgeführt haben. Die sofortige Recovery ist notwendig, um die vom gescheiterten Rechner gehaltenen Betriebsmittel (z.B. Sperren) schnellstmöglich wieder verfügbar zu machen und um für die gescheiterten TA noch akzeptable Antwortzeiten zu erzielen. Die Recovery wird dabei mit der (lokalen) Log-Datei des ausgefallenen Rechners vorgenommen.

Fehlertolerante Speicherung

Schlüsseleigenschaft zur fehlertoleranten Speicherung ist die Replikation der Daten auf Geräten mit unabhängigen Fehlermodi /Gra85/. So sollten die für TA- und Systemfehler benutzten temporären Log-Dateien, die i.a. sowohl UNDO- als auch REDO-Log-Daten halten, zweifach geführt werden (duplex logging). Zur Behandlung von Plattenfehlern werden üblicherweise eine oder mehrere Archiv-Versionen der DB auf Band gehalten. Im Fehlerfall läßt sich mit ihnen der aktuelle DB-Zustand durch die Nachführung der auf Archiv-Protokolldateien gesicherten Änderungen herstellen /Reu81,Reu83/.

Eine andere Form der Datenreplikation zur Behandlung von Plattenfehlern bieten **Spiegelplatten**. Weil bei ihnen alle Daten doppelt auf unabhängige Platten geschrieben werden, führt ein einfacher Plattenfehler zu keiner Unterbrechung und wird automatisch maskiert. Der Nachteil des doppelten Schreibaufwandes, der bei einer NOFORCE-Ausschreibstrategie (s.o.) nicht zu Lasten der Antwortzeit zu gehen braucht, dürfte i.a. durch das größere Optimierungspotential bei Lesevorgängen kompensiert werden. Zum Lesen eines Blocks kann nämlich immer diejenige Platte mit dem geringeren Positionierungsaufwand gewählt werden. Spiegelplatten bieten zwar eine äußerst hohe Ausfallsicherheit pro Plattenpaar (> 1000 Jahre MTBF nach /Gra86a/), sicherheitshalber werden jedoch oft weiterhin Archiv-Kopien und -Protokolldateien geführt. Denn bei 500 Plattenpaaren ist bereits etwa alle zwei Jahre wieder mit einem Ausfall zu rechnen.

Eine weitere Form der fehlertoleranten Speicherung stellt die Replizierung der Daten in verteilten DBS dar. Mit ihr werden zwar hohe Änderungskosten eingeführt, jedoch bietet die räumliche Separierung zusätzliche Verfügbarkeitsvorteile verglichen etwa mit Spiegelplatten (andere Umgebung, anderes Bedienpersonal). Die ortsverteilte Datenreplikation ist auch Voraussetzung für eine schnelle Katastrophen-Recovery (s.o.).

Fehlertolerante Kommunikation

Gemeinsame (Haupt-) Speicher erlauben eine sehr schnelle Kommunikation, Kooperation und Synchronisation zwischen Prozessen bzw. Prozessoren. Andererseits bergen gemeinsame Speicher die Gefahr der Fehlerfortpflanzung und begrenzen Erweiterbarkeit und räumliche Verteilbarkeit des Systems.

Gesichtspunkte der Verfügbarkeit, des modularen Wachstums und der verteilten Ausführung sprechen für eine nachrichtenbasierte Schnittstelle zwischen den Software-Komponenten (Prozessen) /Gra85/. Prozesse und nachrichtenorientierte Kommunikation erlauben eine hierarchische Zerlegung des Systems in modulare Einheiten sowie die Eingrenzung von Fehlern. Die höheren Kosten zur Kommu-

nikation und Synchronisation können jedoch nur mit nachrichtenorientierten BS mit optimierten Kommunikationsprimitiven und sehr schnellen Prozeßwechseln (< 500 Instr.) in Grenzen gehalten werden.

Fehlertolerante Kommunikation erfordert auf der Hardware-Seite mehrfach ausgelegte Kommunikationspfade mit unabhängigen Fehlermodi. Verbindungsorientierte Protokolle (sessions) erlauben die Behandlung und Maskierung von Kommunikationsfehlern gegenüber den Anwendungen. So können z.B. verlorengegangene oder doppelte Nachrichten über Timeout-Verfahren oder Sequenznummern erkannt werden. Bei den oben erwähnten Prozeß-Paaren erlaubt es das Session-Konzept auch bei Ausfall eines Primary-Prozesses, auf den Backup-Prozeß umzuschalten und ihn mit den benötigten Nachrichten zu versorgen /Gra86a/.

VI Literatur

Verwendete Abkürzungen:

TODS = ACM Transactions on Database Systems
TOCS = ACM Transactions on Computer Systems
TOSE = IEEE Transactions on Software Engineering
CACM = Communications of the ACM
IFE = Informatik - Forschung und Entwicklung
IT = Informationstechnik - Computer, Systeme, Anwendungen
VLDB = International Conference on Very Large Data Base Systems
SIGMOD = ACM SIGMOD Int. Conf. on Management of Data
DCS = Int. Conf. on Distributed Computing Systems
PODS = ACM SIGACT-SIGMOD Symp. on Principles of Database Systems
SOSP = ACM SIGOPS Symp. on Operating Systems Principles
SRDSDB = IEEE Symp. on Reliability in Distributed Software and Database Systems
SIGMETRICS = ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems
BTW = GI-Fachtagung über Datenbanken in Büro, Technik und Wissenschaft
LNCS = Lecture Notes in Computer Sciences
IFB = Informatik Fachberichte
IB = Interner Bericht
FB = Fachbereich

/ABGS87/ D. Agrawal, A.J. Bernstein, P. Gupta, S. Sengupta: *Distributed Optimistic Concurrency Control With Reduced Rollback.* Distributed Computing 2 (1), 1987, 45-59

/AbLi80/ M. Abida, B. Lindsay: *Database Snapshots.* Proc. 6th VLDB 1980, 86-91

/ACL85/ R. Agrawal, M.J. Carey, M. Livny: *Models for Studying Concurrency Control Performance: Alternatives and Implications.* Proc. SIGMOD, 1985, 108-121

/AgCa85/ R. Agrawal, M.J. Carey: *The Performance of Concurrency Control and Recovery Algorithms for Transaction-Oriented Database Systems.* IEEE Database Engineering 8 (2), 1985, 58-67

/AgDe85a/ R. Agrawal, D.J. DeWitt: *Recovery Architectures for Multiprocessor Database Machines.* Proc. SIGMOD, 1985, 131-145

/AgDe85b/ R. Agrawal, D.J. DeWitt: *Integrated Concurrency Control and Recovery Mechanisms: Design and Performance Evaluation.* TODS 10 (4), 1985, 529-564

/Agh83/ H. Aghili et al.: *A Prototype for a Highly Available Database System.* IBM Research Report RJ 3755, San Jose, 1983

/Agh86/ H. Aghili et al.: *Highly Available Communication.* IBM Research Report RJ 5068, IBM Almaden Research Center, San Jose, 1986

/Agr85/ R. Agrawal: *A Parallel Logging Algorithm for Multiprocessor Database Machines.* Proc. 4th Int. Workshop on Database Machines, Springer 1985, 256-276

/AIM86/ *AIM/SRCF Functions and Facilities.* Facom OS Techn. Manual 78SP4900E, Fujitsu Limited, 1986

/AlDa76/ P.A. Alsberg, J.D. Day: *A Principle of Resilient Sharing of Distributed Resources.* Proc. 2nd Int. Conf. on Software Engineering, 1976, 562-570

/Amb85/ V. Ambadar: *Data Base Machines.* Proc. 18th Annual Hawaii Int. Conf. on System Sciences, 1985, 352-372

/Amm85/ A.C. Amman et al.: *Design of a Memory Resident DBMS.* Proc. IEEE Spring CompCon, 1985, 54-57

/Anon85/ Anon et al.: *A Measure of Transaction Processing Power.* Datamation, April 1985, 112-118

/APS84/ R. Augustin, U. Prädel, H.A. Scholten: *Leistungsanalyse von Concurrency-Control-Algorithmen in Datenbanksystemen: Ein Überblick.* Bericht 17/1984, Univ. Dortmund, FB Informatik, 1984

/ArBa86/ J. Archibald, J.-L. Baer: *Cache Coherence Protocols: Evaluation using a Multiprocessor Simulation Model.* TOCS 4 (4), 1986, 273-298

/Ari83/ I. Arifin: *Empirische Untersuchungen von Sperrprotokollen für Datenbanksysteme.* Diplomarbeit, FB Informatik, Univ. Kaiserslautern, 1983

/ASP84/ R. Augustin, H.A. Scholten, U. Prädel: *Modelling Database Concurrency Control Algorithms using a General Purpose Performance Evaluation Tool.* Proc. Performance 84, North-Holland 1984, 69-85

/Bad79/ D.Z. Badal: *Correctness of Concurrency Control and Implications in Distributed Databases.* Proc. IEEE COMPSAC, 1979, 588-593

/BaMc84/ D.Z. Badal, W. McElyea: *A Robust Adaptive Concurrency Control for Distributed Databases.* Proc. IEEE INFOCOM, 1984, 382-391

/Bar78/ J.F. Bartlett: *A 'Non Stop' Operating System.* Proc. 11th Hawaii Int. Conf. on System Sciences, 1978, 103-117

/Bar81/ J.F. Bartlett: *A NonStop Kernel.* Tandem TR 81.4, 1981

/Bay76/ R. Bayer: *Integrity, Concurrency, and Recovery in Databases.* Proc. ECI Conf., LNCS 44, Springer 1976, 79-106

/Bay83/ R. Bayer: *Database System Design for High Performance.* Proc. IFIP 9th World Computer Congress, North-Holland 1983, 147-155

/Bay86/ R. Bayer: *Consistency of Transactions and Random Batch.* TODS 11 (4), 1986, 397-404

/BBG83/ A. Borg, J. Baumbach, S. Glazer: *A Message System Supporting Fault Tolerance.* Proc. 9th SOSP, 1983, 90-99

/BBG86/ C. Beeri, P.A. Bernstein, N. Goodman: *A Model for Concurrency in Nested Transactions Systems.* Technical Report TR-CS-86-1, Dept. of Computer Science, Hebrew Univ. Jerusalem, 1986

/BCFP87/ C. Boksenbaum, M. Cart, J. Ferrie, J.-F. Pons: *Concurrent Certification by Intervals of Timestamps in Distributed Database Systems.* TOSE 13 (4), 1987, 409-419

/BeGo81/ P.A. Bernstein, N. Goodman: *Concurrency Control in Distributed Database Systems.* ACM Computing Surveys 13 (2), 1981, 185-221

/BeGo82/ P.A. Bernstein, N. Goodman: *A Sophisticate's Introduction to Distributed Database Concurrency Control.* Proc. 8th VLDB, 1982, 62-76

/BeGo83/ P.A. Bernstein, N. Goodman: *Multiversion Concurrency Control - Theory and Algorithms.* TODS 8 (4), 1983, 465-483

/BEHR80/ R. Bayer, K. Elhardt, H. Heller, A. Reiser: *Distributed Concurrency Control in Database Systems.* Proc. 6th VLDB, 1980, 275-284

/BEHR82/ R. Bayer, K. Elhardt, H. Heller, A. Reiser: *Dynamic Timestamp Allocation for Transactions in Database Systems.* Proc. 2nd Int. Conf. on Distributed Database Systems, North-Holland 1982, 9-20

/BEKK84/ R. Bayer, K. Elhardt, W. Kießling, D. Killar: *Verteilte Datenbanksysteme - Eine Übersicht über den heutigen Entwicklungsstand.* Informatik Spektrum 7 (1), 1984, 1-19

/Ber86/ P.A. Bernstein: *Sequoia - A Fault-Tolerant Tightly-Coupled Computer for Transaction Processing.* IEEE Database Engineering 9 (1), 1986, 17-23

/BGH83/ P.A. Bernstein, N. Goodman, V. Hadzilacos: *Recovery Algorithms for Database Systems.* Proc. IFIP 9th World Computer Congress, North-Holland 1983, 799-807

/BGH86/ J. Bartlett, J. Gray, B. Horst: *Fault Tolerance in Tandem Computer Systems.* Tandem Technical Report TR 86.2, 1986

/BGL83/ P.A. Bernstein, N. Goodman, M. Lai: *Analyzing Concurrency Control Algorithms When User and System Operations Differ.* TOSE 9 (3), 1983, 233-239

/Bha82/ B. Bhargava: *Resiliency Features of the Optimistic Concurrency Control Approach for Distributed Database Systems.* Proc. 2nd SRDSDB, 1982, 19-32

/BHG87/ P.A. Bernstein, V. Hadzilacos, N. Goodman: *Concurrency Control and Recovery in Database Systems.* Addison Wesley 1987

/BHR80/ R. Bayer, H. Heller, A. Reiser: *Parallelism and Recovery in Database Systems.* TODS 5 (2), 1980, 139-156

/BiDe86/ P. Bitar, A.M. Despain: *Multiprocessor Cache Synchronization - Issues, Innovations, Evolution.* Proc. 13th Annual Int. Symp. on Comp. Architecture, 1986, 424-433

/BKK85/ F. Bancilhon, W. Kim, H.F. Korth: *A Model of CAD Transactions.* Proc. 11th VLDB, 1985, 25-33

/BLDN87/ G. v. Bültzingsloewen, R.-P. Liedtke, K. Dittrich, M. Nollau: *Ein Echtzeitdatenbank-Server im Automatisierungsnetz - Anforderungen und Lösungsansätze auf Multi-Computer-Basis.* Proc. 2. BTW, IFB 136, Springer 1987, 460-464

/BoGo84/ H. Boral, I. Gold: *Towards a Self-Adapting Centralized Concurrency Control Algorithm.* Proc. SIGMOD, 1984, 18-32

/Boh85/ V. Bohn: *Simulative Bewertung eines DB-Distribution-Systems.* Diplomarbeit, FB Informatik, Univ. Kaiserslautern, 1985

/Bor81/ A. Borr: *Transaction Monitoring in ENCOMPASS: Reliable Distributed Transaction Processing.* Proc. 7th VLDB, 1983, 155-165

/Bor84/ A. Borr: *Robustness to Crash in a Distributed Database: A Non-Shared-Memory Multi-Processor Approach.* Proc. 10th VLDB, 1984, 445-453

/BoRe85/ H. Boral, S. Redfield: *Database Machine Morphology.* Proc. 11th VLDB, 1985, 59-71

/Bur85/ M. Burman: *Aspects of a High-Volume Production Online Banking System.* Proc. IEEE Spring CompCon, 1985, 244-248

/Cai87/ J. Cai: *Simulation and Evaluation of Distributed Database Systems.* Proc. 4. GI/ITG-Fachtagung über Messung, Modellierung und Bewertung von Rechensystemen, IFB 154, Springer 1987, 313-326

/CaLu86/ M.J. Carey, H. Lu: *Load Balancing in a Locally Distributed Database System.* Proc. SIGMOD, 1986, 108-119

/CaMu86/ M.J. Carey, W.A. Muhanna: *The Performance of Multiversion Concurrency Control Algorithms.* TOCS 4 (4), 1986, 338-378

/Car83/ M.J. Carey: *Granularity Hierarchies in Concurrency Control.* Proc. 2nd PODS, 1983, 156-165

/Car87/ M.J. Carey: *Improving the Performance of an Optimistic Concurrency Control Algorithm through Timestamps and Versions.* TOSE 13 (6), 1987, 746-751

/CaSt84/ M.J. Carey, M.R. Stonebraker: *The Performance of Concurrency Control Algorithms for Database Management Systems.* Proc. 10th VLDB, 1984, 107-118

/CDH85/ A. Chan, U. Dayal, M. Hsu: *Providing Database Management Capabilities for Mission Critical Applications.* Proc. Int. Workshop on High Performance Transaction Systems, Asilomar, 1985

/CDY86/ D.W. Cornell, D.M. Dias, P.S. Yu: *On Multi-System Coupling through Function Request Shipping.* TOSE 12 (10), 1986, 1006-1017

/CeOw82/ S. Ceri, S. Owicki: *On the Use of Optimistic Methods for Concurrency Control in Distributed Databases.* Proc. 6th Berkeley Workshop on Distr. Data Management and Computer Networks, 1982, 117-129

/CePe84/ S. Ceri, G. Pelagatti: *Distributed Databases, Principles and Systems.* Mc Graw-Hill 1984

/Cha82/ A. Chan et al.: *The Implementation of an Integrated Concurrency Control and Recovery Scheme.* Proc. SIGMOD, 1982, 184-191

/Che86/ D.R. Cheriton: *Problem-Oriented Shared Memory: A Decentralized Approach to Distributed System Design.* Proc. 6th DCS, 1986, 190-197

/ChGr85/ A. Chan, R. Gray: *Implementing Distributed Read-Only Transactions.* TOSE 11 (2), 1985, 205-212

/Chr87/ H.-P. Christmann: *Ein Algorithmus zur Systempufferverwaltung und Synchronisation in einem lose gekoppelten Mehrrechner-Datenbanksystem.* Proc. GI/NTG-Tagung Kommunikation in Verteilten Systemen, IFB 130, Springer 1987, 412-425

/CKS86/ G. Copeland, R. Krishnamurthy, M. Smith: *Recovery Using Safe RAM.* Technical Report, Microelectronics and Computer Technology Corp., Austin, Texas, 1986

/CLP87/ J.L. Carroll, D.E. Long, J. Paris: *Block-Level Consistency of Replicated Files.* Proc. 7th DCS, 1987, 146-153

/CLSW84/ J.M. Cheng, C.R. Loosley, A. Shibamiya, P.S. Worthington: *IBM Database 2 Performance: Design, Implementation, and Tuning.* IBM Systems Journal 23 (2), 1984, 189-210

/CoGa85/ R. Cordon, H. Garcia-Molina: *The Performance of a Concurrency Control Mechanism That Exploits Semantic Knowledge.* Proc. 5th DCS, 1985, 350-358

/Dad81/ P. Dadam: *Synchronisation in verteilten Datenbanken: Ein Überblick.* Informatik Spektrum 4, 1981, 175-184 und 261-270

/DaPö87/ P. Dadam, E. Pörtner: *Synchronisation paralleler Operationen auf Index-Bäumen.* Manuskript, IBM Wiss. Zentrum Heidelberg, 1987

/DeW84/ D.J. DeWitt et al.: *Implementation Techniques for Main Memory Database Systems.* Proc. SIGMOD, 1984, 1-8

/DGS85/ S.B. Davidson, H. Garcia-Molina, D. Skeen: *Consistency in Partitioned Networks.* ACM Computing Surveys 17 (3), 1985, 341-370

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

/DIY86/ D.M. Dias, B.R. Iyer, P.S. Yu: *On Coupling Many Small Systems for Transaction Processing.* Proc. IEEE 13th Annual Int. Symp. on Computer Architecture, 1986, 104-110

/Dre81/ I. Dreschers: *Empirische Untersuchung des Seitenreferenzverhaltens eines CODASYL-Datenbanksystems.* Diplomarbeit, FB Informatik, TH Darmstadt, 1981

/DYB87/ D.M. Dias, P.S. Yu, B.T. Bennett: *On Centralized versus Geographically Distributed Database Systems.* Proc. 7th DCS, 1987, 64-71

/EfHä84/ W. Effelsberg, T. Härder: *Principles of Database Buffer Management.* TODS 9 (4), 1984, 560-595

/EGLT76/ K.P. Eswaran, J.N. Gray, R.A. Lorie, I.L. Traiger: *The Notions of Consistency and Predicate Locks in a Database System.* CACM 19 (11), 1976, 624-633

/Eic87/ M. Eich: *A Classification and Comparision of Main Memory Database Recovery Techniques.* Proc. IEEE 3rd Int. Conf. on Data Engineering, 1987, 332-339

/EiJa86/ M.H. Eich, A. James: *Design of a MMDB DBM.* Technical Report TR86-CSE-23, Southern Methodist Univ., Dept. of Comp. Science and Engineering, Dallas, Texas, 1986

/ElBa84/ K. Elhardt, R. Bayer: *A Database Cache for High Performance and Fast Restart in Database Systems.* TODS 9 (4), 1984, 503-525

/Ele86/ *Tandem Makes a Good Thing Better.* In: Electronics, April 14, 1986, 34-38

/Elm86/ A.K. Elmagarmid: *A Survey of Distributed Deadlock Detection Algorithms.* ACM SIGMOD Record 15 (3), 1986, 37-45

/FrRo85/ P. Franaszek, J.T. Robinson: *Limitations of Concurrency in Transaction Processing.* TODS 10 (1), 1985, 1-28

/GaKi85/ D. Gawlick, D. Kinkade: *Varieties of Concurrency Control in IMS/VS Fast Path.* IEEE Database Engineering 8 (2), 1985, 3-10

/Gar83/ H. Garcia-Molina: *Using Semantic Knowledge for Transaction Processing in a Distributed Database.* TODS 8 (2), 1983, 186-213

/Gar86/ H. Garcia-Molina: *The Future of Data Replication.* Proc. 5th SRDSDB, 1986, 13-19

/GaTs84/ G.A. Galatanos, W. Tsai: *Performance Evaluation of Database Update Synchronization on Ethernet Environments.* Proc. 4th DCS, 1984, 503-512

/Gaw85a/ D. Gawlick: *Processing 'Hot Spots' in High Performance Systems.* Proc. IEEE Spring CompCon, 1985, 249-251

/Gaw85b/ D. Gawlick: *High Availability with Large Transaction Systems.* Proc. Int. Workshop on High Performance Transaction Systems, 1985

/GaWi82/ H. Garcia-Molina, G. Wiederhold: *Read-Only Transactions in a Distributed Database.* TODS 7 (2), 1982, 209-234

/Ger83/ E. Gerstner: *Empirische Untersuchungen optimistischer Synchronisationsverfahren in Datenbanksystemen.* Diplomarbeit, FB Informatik, Univ. Kaiserslautern, 1983

/GhMa85/ F.F. Ghertal, S. Mamrat: *An Optimistic Concurrency Control Mechanism for an Object Based Distributed System.* Proc. 5th DCS, 1985, 236-245

/GHOK81/ J. Gray, P. Homan, R. Obermarck, H. Korth: *A Straw Man Analysis of Probability of Waiting and Deadlock.* IBM Research Report RJ 3066, San Jose, 1981

/Gif79/ D.K. Gifford: *Weighted Voting for Replicated Data.* Proc. 7th SOSP, 1979, 150-162

/GiSp84/ D. Gifford, A. Spector: *The TWA Reservation System.* CACM 27 (7), 1984, 650-665

/GlLu84/ V.D. Gligor, G.L. Luckenbaugh: *Interconnecting Heterogeneous Database Management Systems.* IEEE Computer, Jan. 1984, 33-43

/GlPo86/ V. Gligor, R. Popescu-Zeletin: *Transaction Management in Distributed Heterogeneous Database Management Systems.* Information Systems 11 (4), 1986, 287-297

/GLPT76/ J.N. Gray, R.A. Lorie, G.R. Putzolu, I. Traiger: *Granularity of Locks and Degrees of Consistency in a Shared Data Base.* Proc. IFIP Working Conf. on Modelling in Data Base Management Systems, North-Holland 1976, 365-394

/GLV84/ H. Garcia-Molina, R.J. Lipton, J. Valdes: *A Massive Memory Machine.* IEEE Trans. on Computers 33 (5), 1984, 391-399

/Gra78/ J. Gray: *Notes on Data Base Operating Systems.* In: 'Operating Systems - An Advanced Course', LNCS 60, Springer 1978, 393-481

/Gra80/ J. Gray: *A Transactional Model.* In: LNCS 85, Springer 1980, 282-298

/Gra81/ J. Gray: *The Transaction Concept: Virtues and Limitations.* Proc. 7th VLDB, 1981, 144-154

/Gra85/ J. Gray et al.: *One Thousand Transactions per Second.* Proc. IEEE Spring CompCon, 1985, 96-101

/Gra86a/ J. Gray: *Why Do Computers Stop and What Can Be Done About It.* Proc. 5th SRDSDB, 1986, 3-12

/Gra86b/ J. Gray: *An Approach to Decentralized Data Management Systems.* TOSE 12 (6), 1986, 684-692

/GrAn85/ J. Gray, M. Anderton: *Distributed Databases - Four Case Studies.* Tandem Technical Report TR 85.5, 1985

/GSH85/ I. Gold, O. Shmueli, M. Hofri: *The Private Workspace Model Feasibility and Applications to 2PL Performance Improvements.* Proc. 11th VLDB, 1985, 192-208

/Hab86/ F. Haberhauer: *Simulation eines Shared Database Systems mit Primary Copy Synchronisation anhand von Datenbankobjekt-Referenzstrings.* Diplomarbeit Nr. 424, Institut für Informatik, Univ. Stuttgart, 1986

/Hag86/ R.B. Hagman: *A Crash Recovery Scheme for a Memory-Resident Database System.* IEEE Trans. on Computers 35 (9), 1986, 839-843

/HäMe86a/ T. Härder, K. Meyer-Wegener: *Transaktionssysteme und TP-Monitore - Eine Systematik ihrer Aufgabenstellung und Implementierung.* IFE 1 (1), 1986, 3-25

/HäMe86b/ T. Härder, K. Meyer-Wegener: *Die Zusammenarbeit von TP-Monitoren und Datenbanksystemen in DB/DC-Systemen.* IFE 1 (3), 1986, 101-122

/HäPe84/ T. Härder, P. Peinl: *Evaluating Multiple Server DBMS in General Purpose Operating Systems Environments.* Proc. 10th VLDB, 1984, 129-140

/HäPe87/ T. Härder, E. Petry: *Evaluation of a Multiple Version Scheme for Concurrency Control.* Information Systems 12 (1), 1987, 83-98

/Här78/ T. Härder: *Implementierung von Datenbanksystemen.* Carl Hanser 1978

/Här79/ T. Härder: *Leistungsanalyse von Datenbanksystemen.* Angewandte Informatik 4/79, 141-150

/Här84/ T. Härder: *Observations on Optimistic Concurrency Control.* Information Systems 9 (2), 1984, 111-120

/Här86/ T. Härder: *DB-Sharing vs. DB-Distribution - die Frage nach dem Systemkonzept zukünftiger DB/DC-Systeme.* Proc. 9. NTG/GI-Tagung über Architektur und Betrieb von Rechensystemen, NTG-Fachberichte 92, VDE 1986, 151-165

/Här87a/ T. Härder: *Handling Hot Spot Data in DB-Sharing Systems.* IBM Research Report RJ 5523, IBM Almaden Research Center, San Jose, 1987

/Här87b/ T. Härder: *Fehlertoleranz-Aspekte in Transaktionssystemen.* Proc. 3. Int. Tagung über Fehlertolerierende Rechensysteme, IFB 147, Springer 1987, 324-335

/Här87c/ T. Härder: *On Selected Performance Issues of Database Systems.* Proc. 4. GI/ITG-Fachtagung über Messung, Modellierung und Bewertung von Rechensystemen, IFB 154, Springer 1987, 294-312

/HäRa85/ T. Härder, E. Rahm: *Quantitative Analyse eines Synchronisationsalgorithmus für DB-Sharing.* Proc. 3. GI/NTG-Fachtagung über Messung, Modellierung und Bewertung von Rechensystemen, IFB 110, Springer 1985, 186-201

/HäRa86/ T. Härder, E. Rahm: *Mehrrechner-Datenbanksysteme für Transaktionssysteme hoher Leistungsfähigkeit.* IT 28 (4), 1986, 214-225

/HäRa87/ T. Härder, E. Rahm: *Hochleistungsdatenbanksysteme - Vergleich und Bewertung aktueller Architekturen und ihrer Implementierung.* IT 29 (3), 1987, 127-140

/HäRe80/ T. Härder, A. Reuter: *Abhängigkeiten von Systemkomponenten in Datenbanksystemen.* Proc. 10. GI-Jahrestagung, IFB 33, Springer 1980, 243-257

/HäRe83a/ T. Härder, A. Reuter: *Concepts for Implementing a Centralized Database Management System.* Proc. Int. Computing Symposium ICS, Teubner 1983, 28-59

/HäRe83b/ T. Härder, A. Reuter: *Principles of Transaction-Oriented Database Recovery.* ACM Computing Surveys 15 (4), 1983, 287-317

/HäRe85/ T. Härder, A. Reuter: *Architektur von Datenbanksystemen für Non-Standard-Anwendungen.* Proc. BTW, IFB 94, Springer 1985, 253-286

/HäRo87a/ T. Härder, K. Rothermel: *Concepts for Transaction Recovery in Nested Transactions.* Proc. SIGMOD, 1987, 239-248

/HäRo87b/ T. Härder, K. Rothermel: *Concurrency Control Issues in Nested Transactions.* IBM Research Report RJ 5534, IBM Almaden Research Center, 1987

/Häu85/ F. Häussermann: *Verteilte Transaktionsverarbeitung und verteilte Datenhaltung - eine Übersicht und Lösungen.* Elektron. Rechenanlagen 27 (1), 1985, 15-22

/HeGo87/ G. Hermann, G. Gopal: *The Case for Orderly Sharing.* Proc. 2nd Int. Workshop on High Performance Transaction Systems, 1987

/Hel85a/ P. Helland: *Transaction Monitoring Facility (TMF).* IEEE Database Engineering 8 (2), 1985, 11-18

/Hel85b/ P. Helland: *High Transaction Rates in a Distributed System.* Proc. Int. Workshop on High Performance Transaction Systems, Asilomar, 1985

/Hen83/ G. Hendrie: *A Hardware Solution to Part Failures Totally Insulates Programs.* Electronics, Jan. 27, 1983, 103-105

/Her87a/ M. Herlihy: *Optimistic Concurrency Control for Abstract Data Types.* ACM Operating Systems Review 21 (2), 1987, 33-44

/Her87b/ M. Herlihy: *Extending Multiversion Time-Stamping Protocols to Exploit Type Information.* IEEE Trans. on. Computers, 36 (4), 1987, 443-448

/HHMM87/ T. Härder, C. Hübel, K. Meyer-Wegener, B. Mitschang: *Coupling Engineering Workstations to a Database Server.* Proc. Conf. on Data and Knowledge Systems for Engineering and Manufacturing, 1987, 30-39

/HMMS87/ T. Härder, K. Meyer-Wegener, B. Mitschang, A. Sikeler: *PRIMA - A DBMS Prototype Supporting Engineering Applications.* Proc. 13th VLDB, 1987, 433-442

/HoCh85/ R.W. Horst, T.C.K. Chou: *An Architecture for High-Volume Transaction Processing.* Proc. IEEE Comp. Architecture, 1985, 240-245

/HPR85a/ T. Härder, P. Peinl, A. Reuter: *Performance Analysis of Synchronization and Recovery Schemes.* IEEE Database Engineering 8 (2), 1985, 50-57

/HPR85b/ T. Härder, P. Peinl, A. Reuter: *Optimistic Concurrency Control in a Shared Database Environment.* Manuskript, FB Informatik, Univ. Kaiserslautern/Stuttgart, 1985

/Hsi83/ D.K. Hsiao (Hrsg.): *Advanced Database Machine Architectures.* Prentice Hall 1983

/IMS81/ *Administering IMS/VS Systems that Share Data.* In: IMS/VS V1, System Administration Guide, Release 2, SH20-9178-1, 1981, 357-390

/Inm86/ W. Inmon: *A New Measure of Software Speed Narrows DBMS Buyer's Choice.* Computerworld, Sep. 8, 1986, 77-81

/IsMa80/ S.S. Isloor, T.A. Marsland: *The Deadlock Problem: An Overview.* IEEE Computer, Sep. 1980, 58-72

/IYD84/ B.R. Iyer, P.S. Yu, L. Donatiello: *Comparative Analysis of Fault-Tolerant Architectures for Multiprocessors.* IBM Research Report RC 10876, Yorktown Heights, 1984

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

/Jon83/ S. E. Jones: *The Synapse Approach to High System and Database Availability.* IEEE Database Engineering 6 (2), 1983, 29-34

/Kas83/ P.S. Kastner: *A Fault-Tolerant Transaction Processing Environment.* IEEE Database Engineering 6 (2), 1983, 20-28

/Kee82/ W.N. Keene: *Data Sharing Overview.* In: IMS/VS V1, DBRC and Data Sharing User's Guide, Release 2, G320-5911-0, 1982

/Kel85/ U. Kelter: *Parallele Transaktionen in Datenbanksystemen.* Bibliographisches Institut, Reihe Informatik 51, 1985

/KiLa83/ W. Kiessling, G. Landherr: *A Quantitative Comparision of Lockprotocols for Centralized Databases.* Proc. 9th VLDB, 1983, 120-130

/Kim84/ W. Kim: *Highly Available Systems for Database Applications.* ACM Computing Surveys 16 (1), 1984, 71-98

/KiPf85/ W. Kiessling, H. Pfeiffer: *A Comprehensive Analysis of Concurrency Control Performance for Centralized Databases.* Proc. 4th Int. Workshop on Database Machines, Springer 1985, 277-299

/KLMP84/ W. Kim, R. Lorie, D. McNabb, W. Plouffe: *A Transaction Mechanism for Engineering Design Databases.* Proc. 10th VLDB, 1984, 355-362

/KLS86/ N.P. Kronenberg, H.M. Levy, W.D. Strecker: *VAX clusters: A Closely Coupled Distributed System.* TOCS 4 (2), 1986, 130-146

/KlSt86/ K. Kleissner, M. Stumptner: *Performance of Concurrency Control Algorithms in Distributed Databases with Tight Coupling of Multi-Processors at Each Node.* Proc. 6th DCS, 1986, 80-87

/Koh81/ W.H. Kohler: *A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems.* ACM Computing Surveys 13 (2), 1981, 149-183

/KoJe86/ W.H. Kohler, B.P. Jenq: *Performance Evaluation of Integrated Concurrency Control and Recovery Algorithms using a Distributed Transaction Processing Testbed.* Proc. 6th DCS, 1986, 130-139

/Kor83/ H. F. Korth: *Locking Primitives in a Database System.* Journal of the ACM 30 (1), 1983, 55-79

/KuRo81/ H.T. Kung, J.T. Robinson: *On Optimistic Methods for Concurrency Control.* TODS 6 (2), 1981, 213-226

/LaKe82/ A.M. Law, W.D. Kelton: *Simulation Modeling and Analysis.* Mc Graw-Hill 1982

/Lam78/ L. Lamport: *Time, Clocks, and the Ordering of Events in a Distributed System.* CACM 21 (7), 1978, 558-565

/Lau82/ G. Lausen: *Concurrency Control in Database Systems: A Step Towards the Integration of Optimistic Methods and Locking.* Proc. ACM Annual Conf., 1982, 64-68

/LaWi84/ M. Lai, K. Wilkinson: *Distributed Transaction Management in JASMIN.* Proc. 10th VLDB, 1984, 466-470

/LeCa86/ T.J. Lehman, M.J. Carey: *Query Processing in Main Memory Database Management Systems.* Proc. SIGMOD, 1986, 239-250

/LeCa87/ T.J. Lehman, M.J. Carey: *A Recovery Algorithm for a High-Performance Memory-Resident Database System.* Proc. SIGMOD, 1987, 104-117

/LeRo85/ M.D.P. Leland, W.D. Roome: *The Silicon Database Machine.* Proc. 4th Int. Workshop on Database Machines, Springer 1985, 169-189

/Lie86/ C. Liebelt: *Dynamische Optimierung in einem Shared Database Management System mit Primary Copy Synchronisation.* Diplomarbeit Nr. 423, Institut für Informatik, Univ. Stuttgart, 1986

/LiNo83/ W.K. Lin, J. Nolte: *Basic Timestamp, Multiple Version Timestamp, and Two-Phase Locking.* Proc. 9th VLDB, 1983, 109-119

/Lin85/ B. Lindsay: *A Retrospective of R* A Distributed Database Management System.* IBM Research Report RJ 4859, San Jose, 1985

/LoSc87/ P.C. Lockemann, J.W. Schmidt (Hrsg.): *Datenbank-Handbuch.* Reihe Informatik-Handbücher, Springer 1987

/Luc86/ M. Luczak: *Implementierung und quantitative Analyse des Primary-Copy-Sperrverfahrens für DB-Sharing.* Diplomarbeit, FB Informatik, Univ. Kaiserslautern, 1986

/Lyn83/ N.A. Lynch: *Multilevel Atomicity - A New Correctness Criterion for Database Concurrency Control.* TODS 8 (4), 1983, 484-502

/Mal84/ F.J. Malabarba: *Review of Available Database Machine Technology.* Proc. Trends and Applications: Making Database Work, 1984, 14-17

/Man86/ T. Manzke: *Erstellung und Filterung von Referenzstrings für UDS/UTM-Anwendungen.* Diplomarbeit, FB Informatik, Univ. Kaiserslautern, 1986

/MeNa82/ D.A. Menasce, T. Nakanishi: *Optimistic versus Pessimistic Concurrency Control Mechanisms in Database Management Systems.* Information Systems 7 (1), 1982, 13-27

/Met86/ L. Mett: *Empirische Untersuchungen eines Mehrrechner-Datenbanksystems unter Verwendung eines hierarchischen Sperrkonzeptes.* Diplomarbeit, FB Informatik, Univ. Kaiserslautern, 1986

/Mey86/ K. Meyer-Wegener: *Transaktionssysteme - eine Untersuchung des Funktionsumfangs, der Realisierungsmöglichkeiten und des Leistungsverhaltens.* Dissertation, FB Informatik, Univ. Kaiserslautern, 1986

/Mey87/ K. Meyer-Wegener: *Transaktionssysteme - verteilte Verarbeitung und verteilte Datenhaltung.* IT 29 (3), 1987, 120-126

/MGG86/ J.E.B. Moss, N.D. Griffeth, M.H. Graham: *Abstraction in Recovery Management.* Proc. SIGMOD, 1986, 72-83

/MKM84/ S. Muro, T. Kameda, T. Minoura: *Multi-Version Concurrency Control Scheme for a Database System.* Journal of Comp. and System Sciences 29, 1984, 207-224

/MLC87/ J.E. Moss, B. Leban, P.K. Chrysanthis: *Finer Grained Concurrency for the Database Cache.* Proc. IEEE 3rd Int. Conf. on Data Engineering, 1987, 96-103

/MLO86/ C. Mohan, B. Lindsay, R. Obermarck: *Transaction Management in the R* Distributed Database Management System.* TODS 11 (4), 1986, 378-396

/Moh80/ C. Mohan: *Distributed Data Base Management: Some Thoughts and Analyses.* Proc. ACM Annual Conf., 1980, 399-410

/Moh84/ C. Mohan: *Recent and Future Trends in Distributed Data Base Management.* Proc. New York Univ. Symp. on New Directions for Database Systems, 1984.

/Mos82/ J.E.B. Moss: *Nested Transactions and Reliable Distributed Computing.* Proc. 2nd SRDSDB, 1982, 33-39

/Mos85/ J.E.B. Moss: *Nested Transactions: An Approach to Reliable Distributed Computing.* MIT Press 1985

/MoWo85/ R.J.T. Morris, W.S. Wong: *Performance Analysis of Locking and Optimistic Concurrency Control Algorithms.* Performance Evaluation 5 (2), 1985, 105-118

/MuSt82/ T.E. Murray, J.P. Strickland: *IMS/VS Data Sharing Enhancements.* IBM Techn. Discl. Bulletin 25 (7B), 1982, 3715-3717

/Neh87/ J. Nehmer: *Einführung in die Thematik.* IT 29 (6), Themenheft 'Verteilte Systeme', 1987, 377-378

/NeIn85/ E. Nestle, A. Inselberg: *The Synapse N+1 System: Architectural Characteristics and Performance Data of a Tightly-Coupled Multiprocessor System.* Proc. IEEE Comp. Architecture, 1985, 233-239

/New79/ G. Newton: *Deadlock Prevention, Detection and Resolution: An Annotated Bibliography.* ACM Operating Systems Review 13 (2), 1979, 33-44

/Nik87/ C.N. Nikolaou et al.: *Issues in the Design of a Highly Available Multiple Processor Network Attachment.* IBM Research Report RC 12594, Yorktown Heights, 1987

/NoAn87/ J.D. Noe, A. Andreassian: *Effectiveness of Replication in Distributed Computer Networks.* Proc. 7th DCS, 1987, 508-513

/Obe82/ R. Obermarck: *Deadlock Detection for All Resource Classes.* TODS 7 (2), 1982, 187-208

/OKS83/ R.A. Olson, B. Kumar, L.E. Shar: *Messages and Multiprocessing in the ELXI System 6400.* Proc. IEEE Spring CompCon, 1983, 21-24

/Ols85/ R. Olson: *Parallel Processing in a Message-Based Operating System.* IEEE Software, July 1985, 39-49

/ONe86/ P.E. O'Neil: *The Escrow Transactional Method.* TODS 11 (4),1986, 405-430

/Ong84/ K.S. Ong: *Synapse Approach to Database Recovery.* Proc. 3rd PODS, 1984, 79-85

/PaKa84/ C.H. Papadimitriou, P.C. Kanellakis: *On Concurrency Control by Multiple Versions.* TODS 9 (1), 1984, 89-99

/Pap86/ C. Papadimitriou: *The Theory of Database Concurrency Control.* Computer Science Press 1986

/Pei86/ P. Peinl: *Synchronisation in zentralisierten Datenbanksystemen - Algorithmen, Realisierungsmöglichkeiten und quantitative Bewertung.* Dissertation, FB Informatik, Univ. Kaiserslautern, 1986

/PeRe83/ P. Peinl, A. Reuter: *Empirical Comparison of Database Concurrency Control Schemes.* Proc. 9th VLDB, 1983, 97-108

/Pet84/ E. Petry: *Simulation und Analyse eines impliziten Versionenkonzepts für Datenbanksysteme.* Diplomarbeit, FB Informatik, Univ. Kaiserslautern, 1984

/Pet88/ G. Petry: *Quantitative Analyse eines verteilten optimistischen Synchronisationsprotokolls für DB-Sharing.* Diplomarbeit (in Vorbereitung), FB Informatik, Univ. Kaiserslautern, 1988

/PSU82/ U. Prädel, G. Schlageter, R. Unland: *Einige Verbesserungen optimistischer Synchronisationsverfahren.* Proc. 12. GI-Jahrestagung, IFB 57, Springer 1982, 684-698

/PSU86/ U. Prädel, G. Schlageter, R. Unland: *Redesign of Optimistic Methods: Improving Performance and Applicability.* Proc. IEEE 2nd Int. Conf. on Data Engineering, 1986, 466-473

/PuBe86/ K.H. Pun, G.G. Belford: *Optimal Granularity and Degree of Multiprogramming in a Distributed Database System.* Proc. IEEE 2nd Int. Conf. on Data Engineering, 1986, 13-20

/Rah84/ E. Rahm: *Quantitative Analyse eines Synchronisationsprotokolls für Mehrrechner-Datenbanksysteme.* Diplomarbeit, FB Informatik, Univ. Kaiserslautern, 1984

/Rah85a/ E. Rahm: *Weitergehende quantitative Untersuchungen eines Sperrverfahrens für DB-Sharing.* Technischer Bericht, FB Informatik, Univ. Kaiserslautern, 1985

/Rah85b/ E. Rahm: *Analyse von logischen Seitenreferenzstrings zur optimierten Lastkontrolle bei Simulationen von Mehrrechner-Datenbanksystemen.* Technischer Bericht, FB Informatik, Univ. Kaiserslautern, 1985

/Rah86a/ E. Rahm: *Buffer Invalidation Problem in DB-Sharing Systems.* IB 154/86, FB Informatik, Univ. Kaiserslautern, 1986

/Rah86b/ E. Rahm: *Nah gekoppelte Rechnerarchitekturen für ein DB-Sharing-System.* Proc. 9. NTG/GI-Fachtagung über Architektur und Betrieb von Rechensystemen, NTG-Fachberichte 92, VDE 1986, 166-180

/Rah86c/ E. Rahm: *DB-Sharing - eine Realisierungsform zukünftiger Hochleistungs-Datenbanksysteme.* Proc. 1. SAVE-Tagung, 1986, 271-286

/Rah86d/ E. Rahm: *Algorithmen zur effizienten Lastkontrolle in Mehrrechner-Datenbanksystemen.* Angewandte Informatik 4/86, 161-169

/Rah86e/ E. Rahm: *Concurrency Control in DB-Sharing Systems.* Proc. 16. GI-Jahrestagung, IFB 126, Springer 1986, 617-632

/Rah86f/ E. Rahm: *Primary Copy Synchronization for DB-Sharing.* Information Systems 11 (4), 1986, 275-286

/Rah87a/ E. Rahm: *Performance Analysis of Primary Copy Synchronization in Database Sharing Systems.* IB 165/87, FB Informatik, Univ. Kaiserslautern, 1987

/Rah87b/ E. Rahm: *Optimistische Synchronisationsverfahren in Datenbanksystemen: Ein Überblick.* IB 166/87, FB Informatik, Univ. Kaiserslautern, 1987

/Rah87c/ E. Rahm: *Integrated Solutions to Concurrency Control and Buffer Invalidation in Database Sharing Systems.* Proc. 2nd IEEE Int. Conf. on Computers and Applications, 1987, 410-417

/Rah87d/ E. Rahm: *A Reliable and Efficient Synchronization Protocol for Database Sharing Systems.* Proc. 3. Int. Tagung über Fehlertolerierende Rechensysteme, IFB 147, Springer 1987, 336-347

/Rah87e/ E. Rahm: *Design of Optimistic Methods for Concurrency Control in Database Sharing Systems.* Proc. 7th DCS, 1987, 154-161

/Rah88/ E. Rahm: *Optimistische Synchronisationskonzepte in zentralisierten und verteilten Datenbanksystemen.* IT 30 (1), 1988, 28-47

/Ree78/ D.P. Reed: *Naming and Synchronization in a Decentralized Computer System.* PhD Thesis, M.I.T. Dept. of Electrical Engineering, 1978

/ReSh84/ A. Reuter, K. Shoens: *Synchronization in a Data Sharing Environment.* Technischer Bericht, IBM San Jose Research Lab., 1984

/Reu81/ A. Reuter: *Fehlerbehandlung in Datenbanksystemen.* Carl Hanser 1981

/Reu82/ A. Reuter: *Concurrency on High-Traffic Data Elements.* Proc. 1st PODS, 1982, 83-93

/Reu83/ A. Reuter: *Schnelle Recovery-Algorithmen für Datenbanksysteme.* IB 69/83, FB Informatik, Univ. Kaiserslautern, 1983

/Reu84/ A. Reuter: *Performance Analysis of Recovery Techniques.* TODS 9 (4), 1984, 526-559

/Reu85a/ A. Reuter: *Database Sharing.* Informatik Spektrum (Das aktuelle Schlagwort) 8 (4), 1985, 225-226

/Reu85b/ A. Reuter: *The Transaction Pipeline Processor.* Proc. Int. Workshop on High Performance Transaction Systems, Asilomar, 1985

/Reu86a/ A. Reuter: *Load Control and Load Balancing in a Shared Database Management System.* Proc. IEEE 2nd Int. Conf. on Data Engineering, 1986, 188-197

/Reu86b/ A. Reuter: *Mehrprozessor-Datenbanksysteme - Ein Überblick über die wichtigsten Entwurfsprobleme.* Proc. 9. NTG/GI-Tagung über Architektur und Betrieb von Rechensystemen, NTG-Fachberichte 92, VDE 1986, 141-150

/Reu87/ A. Reuter: *PROSPECT: Ein System zur effizienten Bearbeitung komplexer Transaktionen durch Parallelverarbeitung.* Proc. 2. BTW, IFB 136, Springer 1987, 475-480

/RiSt77/ D.R. Ries, M. Stonebraker: *Effects of Locking Granularity in a Database Management System.* TODS 2 (3), 1977, 233-246

/RiSt79/ D.R. Ries, M. Stonebraker: *Locking Granularity Revisited.* TODS 4 (2), 1979, 210-227

/Rob85/ J.T. Robinson: *A Fast General-Purpose Hardware Synchronization Mechanism.* Proc. SIGMOD, 1985, 122-130

/RSL78/ D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis: *System Level Concurrency Control for Distributed Database Systems.* TODS 3 (2), 1978, 178-198

/RyTh85/ I.K. Ryu, A. Thomasian: *Analysis of Database Performance with Dynamic Locking.* IBM Research Report RC 11428, Yorktown Heights, 1985

/RyTh86/ I.K. Ryu, A. Thomasian: *Performance Analysis of Dynamic Locking with the No-Waiting Policy.* IBM Research Report RC 11929, Yorktown Heights, 1986

/San86/ J. Sanguinetti: *Performance of a Message-Based Multiprocessor.* IEEE Computer, Sep. 1986, 47-55

/SaWi85/ D. Sacca, G. Wiederhold: *Database Partitioning in a Cluster of Processors.* TODS 10 (1), 1985, 29-56

/Sch81/ G. Schlageter: *Optimistic Methods for Concurrency Control in Distributed Database Systems.* Proc. 7th VLDB, 1981, 125-130

/Sch82/ G. Schlageter: *Problems of Optimistic Concurrency Control in Distributed Database Systems.* ACM SIGMOD Record 12 (3), 1982, 62-66

/Sch83/ H.-J. Schneider (Hrsg.): *Lexikon der Informatik und Datenverarbeitung.* Oldenbourg 1983

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

/ShSp84/ P.M. Schwarz, A.Z. Spector: *Synchronizing Shared Abstract Types.* TOCS 2 (3), 1984, 223-250

/Sek84/ A. Sekino et al.: *The DCS - A New Approach to Multisystem Data Sharing.* Proc. National Computer Conf., 1984, 59-68

/Sel80/ P.G. Selinger: *Replicated Data.* In: 'Distributed Data Bases', Cambridge Univ. Press 1980, 223-231

/Ser84/ O. Serlin: *Fault-Tolerant Systems in Commercial Applications.* IEEE Computer, Aug. 1984, 19-30

/Ser85/ O. Serlin: *Fault Tolerant Blues.* Datamation, March 1985, 82ff

/Sev83/ K.C. Sevcik: *Comparision of Concurrency Control Methods Using Analytic Models.* Proc. IFIP 9th World Computer Congress, North-Holland 1983, 847-858

/SGA87/ K. Salem, H. Garcia-Molina, R. Alonso: *Altruistic Locking: A Strategy for Coping with Long Lived Transactions.* Proc. 2nd Int. Workshop on High Performance Transaction Systems, Asilomar, 1987

/ShLi86/ A.P. Sheth, M.T. Liu: *Integrating Locking and Optimistic Concurrency Control in Distributed Database* Systems. Proc. 6th DCS, 1986, 89-99

/Sho85/ K. Shoens et al.: *The AMOEBA Project.* Proc. IEEE Spring CompCon, 1985, 102-105

/Sho86/ K. Shoens: *Data Sharing vs. Partitioning for Capacity and Availability.* IEEE Database Engineering 9 (1), 1986, 10-16

/Siw77/ J.E. Siwiec: *A High-Performance DB/DC-System.* IBM Systems Journal 16 (2), 1977, 169-195

/SNM85/ M.K. Sinha, P.D. Nanadikar, S.L. Mehndiratta: *Timestamp Based Certification Schemes for Transactions in Distributed Database Systems.* Proc. SIGMOD, 1985, 402-411

/Son87/ S.H. Son: *Synchronization of Replicated Data in Distributed Systems.* Information Systems 12 (2), 1987, 191-202

/Sto79/ M. Stonebraker: *Concurrency Control and Consistency of Multiple Copies in Distributed INGRES.* TOSE 5 (3), 1979, 188-194

/Sto80/ M. Stonebraker: *The Argument Against CODASYL.* In: 'Distributed Data Bases', Cambridge Univ. Press 1980, 361-370

/Sto86/ M. Stonebraker: *The Case for Shared Nothing.* IEEE Database Engineering 9 (1), 1986, 4-9

/Str85/ R. Strong: *Problems in Fault-Tolerant Distributed Systems.* Proc. IEEE Spring CompCon, 1985, 300-306

/StRo81/ R.E. Stearns, D.J. Rosenkrantz: *Distributed Database Concurrency Controls using Before-Values.* Proc. SIGMOD, 1981, 74-83

/SUW82/ J.P. Strickland, P.P. Uhrowczik, V.L. Watts: *IMS/VS: An Evolving System.* IBM Systems Journal 21 (4), 1982, 490-510

/Tan87/ The Tandem Database Group: *NonStop SQL, A Distributed, High-Performance, High-Availability Implementation of SQL.* Tandem Technical Report 87.4, 1987

/TaSu84/ Y.C. Tay, R. Suri: *Choice and Performance in Locking for Databases.* Proc. 10th VLDB, 1984, 119-128

/Tay87/ Y.C. Tay: *Locking Performance in Centralized Databases.* Perspectives in Computing, Vol. 14, Academic Press 1987

/TCB83/ C. Thanos, C. Carlesi, E. Bertino: *Performance Evaluation of Two-Phase-Locking Algorithms in a System for Distributed Databases.* Proc. 3rd SRDSDB, 1983, 57-69

/TGGL82/ I.L. Traiger, J. Gray, C.A. Galtieri, B.G. Lindsay: *Transactions and Consistency in Distributed Database Systems.* TODS 7 (3), 1982, 323-342

/TGS84/ Y.C. Tay, N. Goodman, R. Suri: *Performance Evaluation of Locking in Databases: A Survey.* Techn. Report TR 17-84, Harvard Univ., Cambridge, Massachussetts, 1984

/TGS85/ Y.C. Tay, N. Goodman, R. Suri: *Locking Performance in Centralized Databases.* TODS 10 (4), 1985, 415-462

/Tho79/ R.H. Thomas: *A Majority Consensus Approach to Concurrency Control for Multiple Copies Data Bases.* TODS 4 (2), 1979, 180-209

/ThRy85/ A. Thomasian, I.K. Ryu: *Analysis of Some Optimistic Concurrency Control Schemes Based on Certification.* Proc. SIGMETRICS, 1985, 192-203

/Tra83/ I. Traiger: *Trends in Systems Aspects of Database Management.* Proc. 2nd Int. Conf. on Databases (ICOD-2), 1983, 1-20

/UDS82/ *Universelles Datenbanksystem UDS V3.2, Entwerfen und Definieren.* Manual-Nr.: U929-J-Z55-1, Siemens AG, München, 1982

/Unl85/ R. Unland: *Optimistische Synchronisationsverfahren und ihre Leistungsfähigkeit im Vergleich zu Sperrverfahren.* Dissertation, FB Mathematik und Informatik, Fernuniversität Hagen, 1985

/Unt84/ K. Unterauer: *Überarbeitung des Log-Verfahrens bei UDS (DB-Cache, DSA).* UDS-Leistungsbeschreibung, Siemens AG, München, 1984

/UPS83/ R. Unland, U. Prädel, G. Schlageter: *Design Alternatives for Optimistic Concurrency Control Schemes.* Proc. 2nd Int. Conf. on Databases (ICOD-2), 1983, 288-297

/Vig87/ D. Viguers: *IMS/VS Version 2 Release 2 Fast Path Benchmark (ONEKAY).* Proc. 2nd Int. Workshop on High Performance Transaction Systems, Asilomar, 1987

/ViRa85/ K. Vidyasankar, V.V. Raghavan: *Highly Flexible Integration of the Locking and the Optimistic Approaches of Concurrency Control.* Proc. IEEE COMPSAC, 1985, 489-494

/Vor87/ F. Vormittag: *Simulation eines Token-Ring-basierten optimistischen Synchronisationsverfahrens in einem DB-Sharing-System.* Diplomarbeit Nr. 499, Institut für Informatik, Univ. Stuttgart, 1987

/Wal83/ B. Walter: *Using Redundancy for Implementing Low-Cost Read-Only Transactions in a Distributed Database System.* Institut für Informatik, Univ. Stuttgart, 1983

/Wal84/ B. Walter: *Nested Transactions With Multiple Commit Points: An Approach to the Structuring of Advanced Database Applications.* Proc. 10th VLDB, 1984, 161-171

/WaMo85/ Y. Wang, R. Morris: *Load Sharing in Distributed Systems.* IEEE Trans. on Computers 34 (3), 1985, 204-217

/Wei86a/ G. Weikum: *A Theoretical Foundation of Multi-Level Concurrency Control.* Proc. 5th PODS, 1986, 31-42

/Wei86b/ G. Weikum: *Pros and Cons of Operating System Transactions for Data Base Systems.* Proc. Fall Joint Comp. Conf., 1986, 1219-1225

/Wei87/ G. Weikum: *Enhancing Concurrency in Layered Systems.* Proc. 2nd Int. Workshop on High Performance Transaction Systems, Asilomar, 1987

/Weih87/ W.E. Weihl: *Distributed Version Management for Read-Only Actions.* TOSE 13 (1), 1987, 55-64

/WeSc84/ G. Weikum, H.J. Schek: *Architectural Issues of Transaction Management in Multi-Layered Systems.* Proc. 10th VLDB, 1984, 454-465

/WIH83/ J.C. West, M.A. Isman, S.G. Hannaford: *PERPOS Fault-Tolerant Transaction Processing.* Proc. 3rd SRDSDB, 1983, 189-194

/Wil80/ P. Wilms: *Qualitative and Quantitative Comparision of Update Algorithms in Distributed Database.* Proc. Distributed Databases, North-Holland 1980, 275-294

/WiLa84/ W.K. Wilkinson, M. Lai: *Managing Replicate Data in JASMIN.* Proc. 4th SRDSDB, 1984, 54-60

/YBL86/ P.S. Yu, S. Balsamo, Y. Lee: *Dynamic Load Sharing in Distributed Database Environments.* Proc. Fall Joint Comp. Conf., 1986, 675-683

/YCDI87/ P.S. Yu, D.W. Cornell, D.M. Dias, B.R. Iyer: *Analysis of Affinity Based Routing in Multi-System Data Sharing.* Performance Evaluation 7 (2), 1987, 87-109

/YCDT86/ P.S. Yu, D.W. Cornell, D.M. Dias, A. Thomasian: *On Coupling Partitioned Data Systems.* Proc. 6th DCS, 1986, 148-157

/YIL86/ P.S. Yu, B.R. Iyer, Y. Lee: *Transaction Recovery in Distributed DB/DC-Systems: A Progressive Approach.* Proc. 5th SRDSDB, 1986, 207-214

/Yu85a/ P.S. Yu et al.: *Modelling of Centralized Concurrency Control in a Multi-System Environment.* Proc. SIGMETRICS, 1985, 183-191

/Yu85b/ P.S. Yu et al.: *Distributed Concurrency Control Analysis for Data Sharing.* Proc. 16th Comp. Measurement Group Conf., 1985, 13-20

/YuCh84/ C.T. Yu, C.C. Chang: *Distributed Query Processing.* ACM Computing Surveys 16 (4), 1984, 399-433

/YYF85/ W.C. Yen, D.W.L. Yen, K. Fu: *Data Coherence Problem in a Multicache System.* IEEE Trans. on Computers 34 (1), 1985, 56-65

/Zöb83/ D.D. Zöbel: *The Deadlock Problem: A Classifying Bibliography.* ACM SIGOPS Operating Systems Review 17 (2), 1983, 6-15