Mehrrechner-Datenbanksysteme für Transaktionssysteme hoher Leistungsfähigkeit

Multiprocessor database systems for high performance transaction systems

T. Härder und E. Rahm, Universität Kaiserslautern

Eine Untersuchung der Anforderungen an Transaktionssysteme zeigt, daß künftige Hochleistungs-Datenbanksysteme folgende Schlüsseleigenschaften besitzen müssen: Abwicklung hoher Transaktionsraten, Gewährleistung hoher Verfügbarkeit, Fähigkeit modularen Wachstums sowie leichte Handhabbarkeit und einfache Verwaltung. Zu ihrer Realisierung werden charakteristische Architekturmerkmale gewonnen, die dann unser Klassifikationsschema prägen. Zwei Architekturklassen von Mehrrechner-Datenbanksystemen – DB-Sharing und DB-Distribution – erweisen sich als geeignet für die Implementierung von Transaktionssystemen hoher Leistungsfähigkeit. Ein Vergleich und eine Bewertung der allgemeinen Systemeigenschaften versuchen die praktische Tauglichkeit dieser Architekturansätze zu beurteilen.

A requirement analysis of transaction systems reveals a number of key properties to be offered by future high performance database systems: high transaction rates, high availability, modular growth as well as simple administration and maintenance. These properties determine characteristic architectural features leading to our classification scheme. Two architectural approaches of multiprocessor database systems – DB-sharing and DB-distribution – seem to be adequate to implement transaction systems satisfying the most important requirements. By comparing and evaluating the general system properties we try to determine the practical usefulness of both architectural approaches.

1 Einleitung

Transaktionssysteme bestimmen zunehmend mehr Abläufe in unserem täglichen Leben – sei es in der Berufswelt oder im privaten Bereich. Solche Systeme werden typischerweise dort eingesetzt, wo eine große Anzahl von Auskunfts-, Buchungs- oder Datenerfassungsvorgängen anfallen, also etwa in Kassenschaltersystemen bei Banken, in Reservierungssystemen bei Fluggesellschaften und Reisebüros, bei der „aktenlosen" Sachbearbeitung bei Behörden sowie in Auftrags- oder Bestellungssystemen und Materialüberwachungssystemen bei Industrieunternehmen. Für diese Einsatzbereiche waren Banken wegen ihrer großen Datenmengen und ihrer hohen Leistungs- und Sicherheitsanforderungen schon immer eine maßgebliche Triebfeder der Entwicklung. So impliziert z. B. die künftige Verbreitung von „Point of sale"-Anschlüssen die Verfügbarkeit hoch-paralleler Transaktionssysteme, deren Leistungsfähigkeit heutige Systeme weit übertrifft. Aber auch im privaten Bereich wird mit einem raschen Wachstum eines solchen Leistungsbedarfs gerechnet (BTX-Anschlüsse, Home-Banking).

Der Kern eines Transaktionssystems wird von einem DB/DC-System gebildet. Anwendungsbezogene Aktivitäten (Vorgänge der realen Welt) werden mit Hilfe von Transaktionsprogrammen durchgeführt, die jeweils genau eine Nachricht empfangen und das (Zwischen-)Ergebnis als eine Nachricht senden. Die Bearbeitung einer Transaktion ist in Bild 1 skizziert. Sind nur eine Ein- und eine Ausgabenachricht erforderlich, so spricht man auch von einer Einschritt-Transaktion, sonst von einer Mehrschritt-Transaktion.

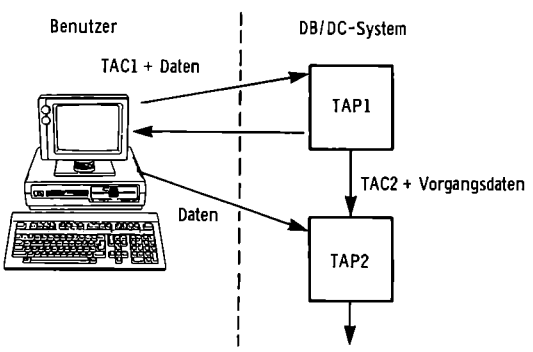

Bild 1: Prinzipieller Ablauf der Verarbeitung in einem Transaktionssystem.

Ein Benutzer schickt eine Eingabenachricht (Bildschirmformular mit Transaktionscode und Daten) an das DC-System (TP-Monitor). Dieses verwaltet die Eingabenachricht und aktiviert nach Verfügbarkeit ein Transaktionsprogramm (TAP). Nach Bearbeiten der Nachricht wird eine Ausgabenachricht erzeugt und das Transaktionsprogramm deaktiviert. Soll die Benutzertransaktion (als Mehrschritt-Transaktion) fortgesetzt werden, so wird vorher das Nachfolge-TAP bestimmt. Nachrichten-, Programm- und Speicherverwaltung übernimmt das DC-System, während das Datenbanksystem für alle Funktionen der Datenhaltung und -sicherung zuständig ist. Typisch für die Arbeitsweise des DC-Systems ist die dynamische Zuordnung von Transaktionsprogrammen zu Benutzern aufgrund von Terminalnachrichten, die nur für die kurze Zeit ihrer Ausführung (ms) bestehen bleibt. Bei einer Terminalausgabe

werden deshalb (fast) alle zugeordneten Ressourcen wie Transaktionsprogramm und Speicherbereiche freigegeben, da die nächste Eingabe für die Fortsetzung einer Mehrschritt-Transaktion vergleichsweise lange auf sich warten lassen kann (Denkzeit im Sekunden- oder Minutenbereich). Damit intern ein Vorgang fortgesetzt werden kann, muß das DC-System dem Nachfolge-Transaktionsprogramm (TAP2) Vorgangsdaten in einem vereinbarten Format als „Gedächtnis" zur Verfügung stellen.

Bei der effizienten Realisierung von DB/DC-Systemen sind die speziellen Betriebscharakteristika von Transaktionssystemen zu berücksichtigen [1]:

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

Ginge es in Transaktionssystemen nur um die Bereitstellung von Rechnerleistung, so könnten beliebige Leistungsanforderungen leicht durch die Zuordnung zusätzlicher Rechner befriedigt werden. Das Hauptproblem bei Mehrrechner-Systemen ist jedoch die gemeinsame Datenhaltung, da in jedem Rechner den Anwendungen das Bild einer **zentralen Datenbank,** auf die im logischen Einbenutzer-Betrieb zugegriffen werden kann, geboten werden muß. Herkömmliche Datenbanksysteme (DBS) erfüllen diese Forderung nicht, da sie als integrierte Softwaresysteme zum Aufbau, zur Verwaltung und zur Kontrolle von Datenbeständen konzipiert sind, die **einem** Rechner zugeordnet bleiben. Die Schlüsseleigenschaften zukünftiger Transaktionssysteme, zu denen hohe Leistung, „permanente" Verfügbarkeit und modulares Wachstum gehören, verlangen jedoch Datenbanksysteme, die als ein logisches System verteilt auf mehreren gekoppelten Rechnern ablaufen können [2, 3]. Solche Mehrrechner-Datenbanksysteme müssen einerseits den gleichen Funktionsumfang wie zentrale Datenbanksysteme bieten, sollen jedoch andererseits wesentliche zusätzliche Anforderungen erfüllen, die in Kap. 2 näher diskutiert werden. Das Spektrum ihrer Implementierungsformen wird in Kap. 3 untersucht und durch ein Klassifikationsschema so geordnet, daß sich zwei Architekturklassen als allgemein geeignete Ansätze zur Realisierung von Mehrrechner-Datenbanksystemen (MRDBS) herausschälen. Kap. 4 versucht dann eine Bewertung der praktischen Tauglichkeit dieser Architekturen im Hinblick auf die für Transaktionssysteme hoher Leistungsfähigkeit geforderten Schlüsseleigenschaften.

2 Anforderungen an künftige Transaktionssysteme

Bevor Architekturvorschläge für MRDBS detaillierter untersucht werden können, diskutieren wir eine Reihe wichtiger Anforderungen an künftige Transaktionssysteme, die wesentlich ihre Schlüsseleigenschaften bestimmen werden.

2.1 Leistung

Transaktionssysteme lassen sich charakterisieren durch hoch-parallele Abwicklung von typischerweise kurzen und vorgeplanten Transaktionen gleichen oder verschiedenen Typs, wobei vor allem kurze Antwortzeiten zu gewährleisten sind (in der Regel wenige Sekunden). Die „Kontenbuchung" (DEBIT_CREDIT [1, 4]) kann als treffendes Beispiel für solche Transaktionen herangezogen werden. Sie stellt eine Einschritt-Transaktion mit einer Ein- und Ausgabe-Nachricht vom/zum Terminal dar. Als dominierender Transaktionstyp bei Bankanwendungen wird sie sogar als Maßeinheit bei der Bewertung von Transaktionssystemen vorgeschlagen (Transaktionen pro Sekunde (TPS) als Durchsatzmaß bei einer Antwortzeit $\leqslant$ 1 sec für 95% der Transaktionen [5]).

Neben solchen kurzen Transaktionen laufen natürlich auch längere ab, die teilweise über mehrere Dialogschritte abgewickelt werden. Zur Charakterisierung der Systemlast versuchen wir den durchschnittlichen Ressourcenbedarf beider Transaktionsklassen abzuschätzen. Als leistungsbestimmende Größen sind dabei die Anzahl der Kilo-Instruktionen (/370-System), die Anzahl der benötigten Platten-Ein-/Ausgaben (E/A für Daten und Logging), die Anzahl der Ein-/Ausgabe-Nachrichten vom/zum Terminal (MSG), die anzufordernden Sperren und der Systemdurchsatz (TPS) zu berücksichtigen. Mit diesen Kenngrößen lassen sich die Leistungsgrenzen heutiger, optimierter Transaktionssysteme (auf der Basis von 10-12 MIPS-Rechnern) wie folgt angeben:

	K-Instr.	E/A (Lesen/ Schreiben)	MSG	Sperren	TPS
kurze TA	50	1/3	2	4	200
mittlere TA	500	6/4	2–4	20	15

Obwohl bei kurzen (abgemagerten) Transaktionen heute schon eine beachtliche Systemleistung erzielt wird, bestehen für komplexere Transaktionsmixe doch recht große Leistungsbeschränkungen. Dieser Sachverhalt kann nicht ausgeglichen werden durch ein höheres Leistungsangebot von zentralen Rechnersystemen, selbst wenn eine jährliche Steigerung der CPU-Leistung von 15–20% aufgrund technologischer Fortschritte unterstellt wird. Es ist eher zu erwarten, daß eine Reihe von Leistungsanforderungen wie

- steigende Anzahl von Transaktionstypen bei existierenden und neuen Anwendungen

- Bearbeitung komplexerer Vorgänge mit umfangreichen Integritätsbedingungen
- Benutzung höherer Programmiersprachen und komfortablerer Benutzerschnittstellen
- Einsatz von Programmgeneratoren, die wenig leistungsfähigen Code erzeugen

die Kapazitätsbeschränkungen herkömmlicher Transaktionssysteme erheblich verschärfen werden. Dazu ergeben sich höhere Ansprüche an die Qualität und Effizienz der Datenbankverwaltung – insbesondere bei Synchronisation und Logging/Recovery –, wenn in verstärktem Maße Mehrschritt-Transaktionen – sie entsprechen vielen Arbeitsabläufen in natürlicherer Weise – oder gar Stapelprogramme konkurrierend zu kurzen Transaktionen abgewickelt werden sollen.

Die neue Herausforderung an die DB-Entwickler wurde im letzten Jahr [6] durch die magische Größe „1 KTPS" vorgegeben. Bis 1990 sollen 1000 Transaktionen/sec vom Typ „Kontenbuchung" DB-gestützt abgewickelt werden. Solche Leistungen werden neben dem Bankbereich in anderen hoch-parallelen Systemen mit DB-Zugriff erforderlich („point of sale"-Anschlüsse, BTX-Terminals, Home-Banking). 10 KTPS für sehr einfache Transaktionen gelten schon als diskussionswürdig, auch wenn ihre Realisierung noch längere Zeit auf sich warten läßt.

2.2 Verfügbarkeit und Fehlertoleranz

Hohe Verfügbarkeit besitzt in Transaktionssystemen, die in Betriebsabläufe integriert sind, einen extrem hohen Stellenwert, da ein Systemausfall einen ganzen Betrieb lahmlegen kann [7]. Dabei sind alle Systemkomponenten in die Betrachtung einzubeziehen. Als Zahlenwerte für die Verfügbarkeit von Geldautomaten (ATM) gibt J. Gray in [8] folgende Kennzahlen an:

ATM	97%	(mechanischer Ausfall, kein Geld)
Kommunikationsnetz	99%	(100 min/Woche)
Host-Rechner	99,5%	(10 min/Woche)

Auf den ersten Blick erscheint der zentrale Rechner nicht als kritische Größe bei dieser Systembetrachtung. Selbst wenn sich 10 Minuten Ausfall pro Woche tolerieren lassen, darf nicht die isolierte Betrachtungsweise einzelner Systemkomponenten herangezogen werden. Die Anatomie eines Systemzusammenbruchs bei großen Terminalnetzen zeigt die Abhängigkeiten beim Wiederanlauf (Restart), so daß sich erhebliche **Wiederanlaufzeiten** aufaddieren:

- Die Feststellung und Analyse des Fehlers erfordert oft mindestens 10 Minuten.
- Die Ladezeit des Betriebssystems (Warmstart) beträgt typischerweise bis zu 10 Minuten.
- Der Warmstart des DBS und die Rekonstruktion einer transaktionskonsistenten Datenbank läßt sich mit optimierten Logging-, Checkpoint- und Recovery-Verfahren auch auf 10 Minuten (oder weniger) begrenzen.
- Der Restart-Aufwand des DC-Systems hängt erfahrungsgemäß von der Anzahl der angeschlossenen Terminals ab. Mit beobachteten Restart-Zeiten in der Größenordnung von 30 Minuten pro tausend Terminals ist bei großen Terminalnetzen mit einigen Stunden Ausfallzeit zu rechnen.

Eine Optimierung der DC-Recovery erscheint möglich und dringlich; außerdem kann eine parallele Ausführung der DB- und DC-Recovery die gesamte Ausfallzeit weiter reduzieren. Trotzdem ergeben sich spürbare Ausfallzeiten, die weder in Transaktionssystemen noch in anderen Systemen mit höchsten Verfügbarkeitsanforderungen akzeptabel sind. Deshalb ergibt sich aus der vorliegenden Zeitbilanz oder aus Verfügbarkeitsanforderungen zwangsläufig die Notwendigkeit, einen Totalausfall des Systems mit allen Mitteln zu verhindern. Wegen der großen Wiederanlaufzeiten läßt sich hohe Verfügbarkeit in Transaktionssystemen **nur** durch **Fehlertoleranz** bei allen Systemkomponenten erreichen. Durch geeignete Maßnahmen zur Fehlertoleranz werden „single points of failure" in allen (kritischen) Systemkomponenten vermieden, so daß zwar Teile des Systems ausfallen können, das Gesamtsystem jedoch (fast) immer arbeitsfähig bleibt. Es ist klar, daß diese Forderung eindeutig auf Mehrrechner-Architekturen hinweist.

Natürlich macht fehlertolerante Hardware noch kein fehlertolerantes Gesamtsystem aus [9]. Andere Fehlerursachen sind oft viel schwieriger zu bekämpfen; sie besitzen zudem einen viel größeren Einfluß auf die Verfügbarkeit des Gesamtsystems. Nach [8] teilen sich die Fehlerursachen in einem fehlertoleranten System folgendermaßen auf:

- Administration (Wartung, Betrieb, Konfiguration) 42%
- Software (Hersteller, Anwender) 25%
- Hardware (CPU, Platten, Bänder, Controller, Stromversorgung) 18%
- Umgebung (Stromversorgung, Kommunikationsnetz etc.) 14%

Mögen sich diese Zahlen auch im Einzelfall (andere Hersteller, andere Fehlerzuordnung) geringfügig verschieben, ihre Tendenz bleibt jedoch eindeutig (auch für konventionelle Systeme).

Fehlertoleranz läßt sich vor allem auch durch Fehlerisolation unterstützen. Auf der HW-Ebene bedeutet dies [8]

- eine hierarchische Systemzerlegung in Module, die jeweils mit einer MTBF (mean time between failure) > 10000 h zu entwerfen sind und „fail-stop"-Mechanismen besitzen
- eine Konfigurierung mit zusätzlichen Moduln, die bei Ausfall eines Moduls dessen Last innerhalb von Sekunden übernehmen können
- einen Verzicht auf gemeinsame Speicher wegen der besseren Fehlerisolation (lose gekoppelte Mehrrechner-Systeme).

Fehlertolerante Ausführung bedeutet im Fehlerfall die Fortführung eines Prozesses/einer Transaktion auf einem ausführungsbereiten Prozessor. Geeignete Softwarestrukturen (Prozesse), die ausschließlich über Nachrichten kommunizieren, sind eine wesentliche Voraussetzung für befriedigende Lösungen. Da Prozesse ortstransparent adressiert werden müssen, unterstützt sie ein nachrichtenorientiertes Betriebssystem in natürlicher Weise. Fehlertoleranz bei der Ausführung der Anwendung kann durch Konzepte wie Prozeß-Paare oder Schattenprozesse erreicht werden. Zusammen mit dem Transaktionskonzept [10] zur logischen Strukturierung der Anwendung (TAP) läßt sich so eine konsistente und fehlertolerante Verarbeitung erzielen.

Weitere wichtige Systemaspekte sind die fehlertolerante Kommunikation (sitzungsorientierte Protokolle) und die fehlertolerante Speicherung der Daten (Logging, Spiegelplatten). Beispielsweise wird die MTBF einer Platte von 2 Jahren durch Spiegelung auf 1500 Jahre erhöht, wenn unterstellt wird, daß der Ausfall einer Platte in kurzer Zeit repariert werden kann.

Der Stellenwert fehlertoleranter Architekturen für Transaktionssysteme läßt sich durch zwei Zahlen am besten verdeutlichen. In einer umfangreichen empirischen Untersuchung [8] werden folgende MTBF-Werte als typisch angesehen, wobei wiederum nur die Größenordnung eine Rolle spielt:

MTBF in konventionellen Systemen (IBM-3081 MVS-XA, Mitte 1984)	~ 10 Tage,
MTBF in fehlertoleranten Systemen (TANDEM, 1985)	~ 10 Jahre.

Auch fehlertolerante Systeme weisen Systemausfälle auf (NonStop ist eine Übertreibung! [11]) und erzwingen folglich vorbereitende Maßnahmen zur Behebung des Fehlerfalls. Jedoch garantieren sie einen ungleich stabileren Transaktionsbetrieb.

2.3 Wachstum

Modulares Wachstum ist eine weitere zentrale Forderung für ein Transaktionssystem. Seine Architektur – bezogen auf Hardware und Software – soll ein mit den Anwendungen schritthaltendes Wachstum erlauben. Diese Forderung – wie auch die Verfügbarkeitsforderung – schließt monolithische Systeme von vornherein aus; dazu zählen auch eng gekoppelte Mehrprozessorsysteme, die oft durch überlastete Speicher oder Kommunikationsstrukturen frühzeitig in die Sättigung geraten (3–4 Prozessoren).

Hardware- und Software-Architektur müssen aus kleineren Subsystemen gebildet werden, die Einheiten des **Entwurfs,** der **Kontrolle** und des **Ausfalls** sind [6]. Dadurch läßt sich ein granulares Wachstum des Systems erreichen; ein lineares Wachstum seiner Leistungsfähigkeit, das idealerweise anzustreben ist, kann damit noch nicht garantiert werden.

Linearität der transaktionsbezogenen Rechnerleistung ist jedoch eine wichtige Entwurfsforderung. Sie bedeutet, daß eine (annähernd) lineare Durchsatzerhöhung bei (annähernd) gleichbleibender Antwortzeit zu bewerkstelligen ist. Es wird in [12] geschätzt, daß Anwender höchstens 10–20% längere Antwortzeiten tolerieren, wenn ein zentrales Rechensystem durch ein Mehrrechnersystem ersetzt wird. Wegen der zusätzlichen Kommunikation der verteilten Kontrolle etc. ist diese Forderung sehr schwierig zu erreichen. Folgende Aspekte zeigen jedoch den Kern des Problems:

1. Zur Beibehaltung des Durchsatzes ist zunächst eine Erhöhung der internen Parallelität erforderlich, da durch längere Kommunikation höhere Antwortzeiten zu erwarten sind. Dies impliziert allerdings mehr Terminals (Personal) zur Aktivierung der Transaktionen.
2. Mehr offene Transaktionen bedeuten längere Wartezeiten durch zunehmende Behinderungen und Rücksetzungen (z. B. wegen Deadlocks) beim Zugriff auf gemeinsame Datenstrukturen, was wiederum eine Erhöhung der Antwortzeit zur Folge hat.
3. Eine Durchsatzerhöhung verschlechtert die Antwortzeit weiter (siehe 1). Nur effiziente Synchronisation und optimale Ausnutzung von Lokalität der Verarbeitung versprechen die Lösung des Antwortzeitproblems.

Wachstum läßt sich in einfacher Weise in Systemen modularer Struktur, deren Komponenten ausschließlich über Nachrichten kommunizieren, erreichen. Diese Lehrmeinung gilt vor allem für Systeme, die nur schwach über gemeinsame Daten gekoppelt sind. In Transaktionssystemen sollen jedoch alle Verarbeitungskomponenten potentiell auf alle Daten zugreifen können. Neben den Verarbeitungskomponenten wachsen in solchen Systemen auch die Datenbestände. Bei vorhandenen Datentypen (Satztypen) nimmt die Anzahl der Ausprägungen zu; durch neue Anwendungen müssen neue Datentypen zugeordnet werden. Alle Daten sollen im Normalbetrieb und im Fehlerfall gleichermaßen gut erreichbar und bearbeitbar sein, was geeignete Systemstrukturen berücksichtigen müssen.

2.4 Handhabbarkeit und einfache Verwaltung

Wachstums- und Verfügbarkeitsanforderungen stellen auch besondere Ansprüche an **Handhabbarkeit** und **einfache Verwaltung.** Die Benutzerschnittstelle heutiger Transaktionssysteme – durch Bildschirmformulare und Transaktionscodes realisiert – bietet Daten- und Programmunabhängigkeit [1] und ist sehr einfach zu benutzen. Ebenso befriedigend gelöst ist die Programmierschnittstelle, da beim Entwurf von TAPs – in höheren Programmiersprachen – immer nur die Verarbeitung einer Nachricht im logischen Einbenutzerbetrieb berücksichtigt werden muß; sie ist kommunikations-, kontroll- und datenunabhängig [1]. Die Schnittstellen für die Systembedienung, -administration und -wartung erfüllen jedoch oft bei weitem nicht bestimmte Mindestansprüche. Wegen der zusätzlichen Komplexität kommt dem Entwurf dieser Schnittstellen in Mehrrechnersystemen große Bedeutung zu. Die Größe und Un-

überschaubarkeit solcher Systeme erzwingt besondere Maßnahmen zur einfachen Verwaltung, leichten Konfigurierbarkeit sowie dynamischen Erweiterbarkeit und Änderbarkeit.

Bei der Diskussion der Verfügbarkeit wurde schon herausgestellt, daß der Aspekt „Administration" für 42% der Systemfehler (-ausfälle) verantwortlich ist. Deshalb kann in diesem Bereich der größte Fortschritt in Richtung „Verfügbarkeit" und gleichzeitig „Vereinfachung" der Verwaltung erwartete werden. Als Zielvorstellung ist ein **single system image** des Mehrrechner-DBS an der Schnittstelle der Administration, Wartung und Bedienung anzustreben. Die Zahl der Steuerungs- und Eingriffsmöglichkeiten soll nicht mit der Zahl der Rechner wachsen; sondern es soll vielmehr das Gesamtsystem über eine zentrale Kontrollinstanz (Masterkonsole) verwaltet werden können.

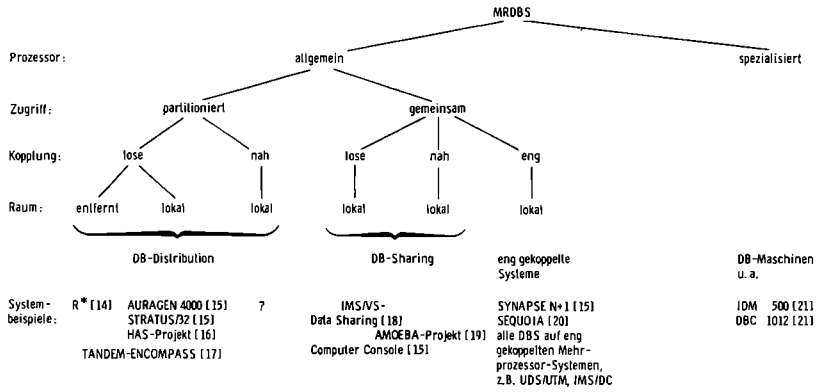

Bild 2: Klassifikationsschema für Mehrrechner-Datenbanksysteme.

Die Liste weiterer Entwurfsanforderungen an zukünftige Mehrrechner-DBS ist lang, wenn neben hoher Leistungsfähigkeit die Aspekte „Modulares Wachstum" und „Höchste Verfügbarkeit" (continuous operation, 24-h-Betrieb an 365 Tagen) keine Schlagwörter bleiben sollen:

- Reorganisationsfreiheit aller Speicherungsstrukturen eines DBS
- Erweiterung des Transaktionssystems im laufenden Betrieb (Anzahl der Benutzer, neue Transaktionsprogramme, Übergang zu neuen Betriebssystem- oder DBS-Versionen etc.)
- Geographische Verteilung des Systems zur besseren Anpassung an Organisationsstrukturen
- Automatisches Erkennen und Beheben von Fehlern
- Unterbrechungsfreie Reintegration von ausgefallenen Systemkomponenten
- Vorsorgemaßnahmen für unerwartete Ereignisse (Katastrophen-Recovery), die bestimmte Formen der Ortsverteilung voraussetzen.

3 Mehrrechner-Datenbanksysteme – eine Klassifikation

Ziel unserer Klassifikation ist die Beschreibung der Eigenschaften von MRDBS, mit denen die in Kap. 2 skizzierten Anforderungen erfüllt werden können. Wahl der Merkmale und ihre Anordnung im Schema wurden so vorgenommen, daß drei wichtige Architekturklassen von MRDBS [3, 13] separiert und in Bild 2 veranschaulicht werden konnten:

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

Diese Klassenbildung soll durch das Klassifikationsschema erklärt werden, d. h., es sollen die wesentlichen Unterschiede zwischen den Klassen und eine Feinaufteilung innerhalb der Klassen herausgearbeitet werden.

Vor allem mit Argumenten aus Kap. 2 wird unsere Betrachtung auf den Einsatz allgemeiner Prozessoren eingeschränkt. Die Merkmale „Zugriffsform auf Externspeicher" und „Rechnerkopplung" sind verantwortlich für das Herausbilden der drei Architekturklassen. Das Kriterium „räumliche Aufstellung" soll noch einmal verdeutlichen, welche Lösungen an einen Aufstellungsort gebunden sind.

Die Architekturklasse „DB-Distribution" läßt sich vor allem durch die Partitionierung der Externspeicher (und Daten) charakterisieren. Eine lose Kopplung erlaubt die räumlich entfernte Aufstellung der Prozessoren. Deshalb lassen sich auch verteilte Datenbanksysteme (VDBS) in diese Architekturklasse einordnen. Bei eng gekoppelten Systemen läuft ein Datenbankverwaltungssystem (DBVS) wie im zentralisierten Fall in einem oder mehreren Prozessen (Servern) ab; seine Prozessorzuordnung bleibt ihm jedoch verborgen (Abbildung durch das Betriebssystem (BS)). In der Architekturklasse „DB-Sharing" sind auf der Ebene der DB-Verwaltung alle Prozessoren sichtbar; deshalb muß die Abbildung der einzelnen DB-Prozesse auf die verschiedenen Prozessoren und ihre Kooperation explizit geleistet werden. Bei DB-Sharing als auch bei DB-Distribution könnte ein Knoten durch einen eng gekoppelten Rechnerverbund realisiert sein. Dadurch ergeben sich (außer Prozessorleistung) jedoch keine neuen Aspekte.

Nach dieser groben Kennzeichnung der Klassenbildung erfolgt eine detailliertere Diskussion und Begründung ihrer Merkmale.

3.1 Wahl der Prozessoren

Natürlich kommen in jedem Mehrrechner-Verbund allgemeine und spezialisierte Prozessoren vor, wenn man beispielsweise E/A- oder Kommunikationsprozessoren mit in die Betrachtung einbezieht. Hier sollen jedoch nur die Prozessoren zur Klassifikation herangezogen werden, die für die DB-Verwaltung Verarbeitungsleistung bereitstellen; sie bestimmen das Architekturmerkmal „Prozessortyp".

Spezialisierte Prozessoren implizieren eine funktionsorientierte Verarbeitung. Die Leistungsfähigkeit einer

aus dedizierten Prozessoren aufgebauten Architektur wird wesentlich durch die Wahl geeigneter Funktionsgranulate bestimmt. Als kritische Größen sind einerseits ihre Aufrufhäufigkeiten, die effiziente Kontextwechsel und schnelle Kommunikationsprotokolle zwischen den Prozessoren erfordern, und andererseits mangelnde Möglichkeiten der Lastkontrolle und -optimierung anzusehen. Da in herkömmlichen Systemkonzepten solche Dienste synchron angefordert werden, belasten sie die Antwortzeit proportional zu ihrer Aufrufhäufigkeit. Der Leistungsaspekt von DB-Maschinen ist auch stärker ausgerichtet auf die Ausnutzung von Parallelität bei der Bearbeitung einer mächtigen DB-Operation (Suchfrage, Verbund); in den Anwendungen von DB/DC-Systemen dominieren dagegen sehr kurze DB-Operationen mit Satzzugriff.

Neben dem Leistungsaspekt sind hier vor allem noch Argumente zur Verfügbarkeit und zum Wachstum zu bedenken. Die Redundanzforderung würde auch bei wenig ausgelasteten Funktionen eine Duplizierung der Prozessoren erzwingen. Lineares Wachstum würde nun nicht mehr in Einheiten eines einzelnen Prozessors vorgenommen werden können, weil beispielsweise mehrere Funktionsprozessoren gemeinsam ihre Leistungsgrenze erreicht haben könnten und ein zusätzlicher Spezialprozessor nur einen Engpaß beseitigen würde.

Aus diesen Gründen konzentrieren wir uns auf Architekturen mit mehreren allgemeinen Verarbeitungsprozessoren. Sie erlauben die flexiblere Form der Lastkontrolle und gestatten prinzipiell die Abarbeitung einer Transaktion in einem Prozessor. Bei Ausfall eines Prozessors kann seine Last – zumindest hardwareseitig – linear auf die verbleibenden Prozessoren aufgeteilt werden. Analog verhält es sich mit der allgemein nutzbaren Rechnerleistung bei Zuschalten eines Prozessors.

Natürlich können spezialisierte Prozessoren für gewisse Aufgaben in Verbunden mit allgemeinen Prozessoren – z. B. im Rahmen einer Speicherhierarchie oder als Filterprozessor – herangezogen werden. Solche Mischformen (für periphere Aufgaben) verändern jedoch die wesentlichen Architekturprinzipien nicht; sie sind deshalb in unserer Klassifikation nicht als eigene Klasse berücksichtigt.

3.2 Zugriffsarten auf Externspeicher

Das wesentlichste Unterscheidungsmerkmal der betrachteten Architekturen ist die Form der Anbindung der Externspeicher an die Verarbeitungsprozessoren. Beim **partitionierten Zugriff** ist jede Platte genau einem Rechner zugeordnet, während beim **gemeinsamen Zugriff** alle Rechner auf alle Platten zugreifen können.

1. Partitionierter Zugriff (Bild 3a)

Jedes Plattenlaufwerk wird also von einem Prozessor verwaltet; alle Zugriffe und Wartungsarbeiten werden von ihm vorgenommen. Da ein Rechner jedoch nur Zugriff zu seiner Partition hat, ist die Transaktionsverarbeitung in der Regel verteilt.

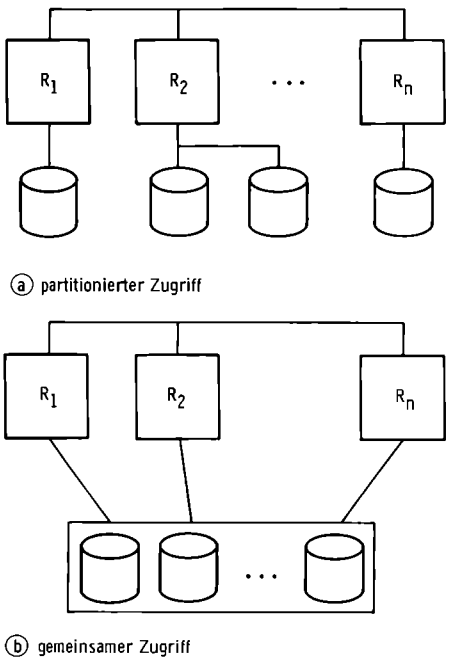

Bild 3: Arten der Zuordnung von Plattenlaufwerken zu Prozessoren.

Die Plattenzuordnung ist prinzipiell statisch; sie kann zwar geändert werden – z. B. beim Rechnerausfall oder bei Konfigurationsänderung –, jedoch ist ein solches Ereignis mit erheblichem Aufwand verbunden (Multi-Ports, Verlegen von Kabeln u. ä.).

Die Einheit der Verteilung sind Plattenlaufwerke und nicht Daten. Bei VDBS, für die wegen der Ortsverteilung nur partitionierter Zugriff möglich ist, findet man deshalb oft eine Charakterisierung der Datenverteilung, bei der Replikationen berücksichtigt werden (volle und partielle Redundanz sowie Partionierung der Daten [2]). Beim partitionierten Zugriff sind – zur Beschleunigung von Abfrageoperationen und zur Verbesserung der Verfügbarkeit – prinzipiell auch partielle oder volle Redundanz möglich. In VDBS kann Redundanz jedoch wegen des Aufwands zur Konsistenzerhaltung von Kopien nur bei geringer Änderungsintensität sinnvoll eingesetzt werden. In lokalen MRDBS scheint die Benutzung von Redundanz noch weniger praktisch bedeutsam zu sein als in VDBS. Geringe Kommunikationskosten machen eine Kopienhaltung für effiziente Abfrageoperationen überflüssig. Die Verfügbarkeit kann wesentlich einfacher durch Spiegelplatten – also durch Replikation auf demselben Rechner – erhöht werden, da wegen der zentralen Kontrolle die Konsistenz der Spiegelplatten mit sehr geringem Aufwand gewährleistet werden kann. Fehlertoleranz bei Externspeicherfehler ist hier offensichtlich; bei einem Rechnerausfall bleiben alle Daten verfügbar, wenn ein anderer Rechner die Kontrolle übernehmen kann.

2. Gemeinsamer Zugriff (Bild 3b)

Beim gemeinsamen Zugriff hat jeder Prozessor Zugriff zu allen Platten. Wegen der erforderlichen Kanalkopplung impliziert diese Zugriffsform eine lokale Aufstellung aller Rechner. Wegen der Erreichbarkeit aller Platten ist stets gewährleistet, daß die Transaktionsverarbeitung vollständig auf einem Rechner erfolgen kann.

Nach einem Rechnerausfall bleibt der Zugriff auf alle Platten erhalten. Andererseits kann eine Platten-Recovery von jedem Rechner übernommen werden, ohne daß vorher technische Veränderungen vorgenommen werden müssen. Konfigurationsänderungen sind auf dieser abstrakten Betrachtungsebene ebenfalls kein Problem. Neu hinzukommende Rechner können auf alle Platten zugreifen, und neu hinzukommende Platten können von allen Rechnern aus erreicht werden.

Beim Zugriff auf eine Platte ist ohne Koordination der Rechner mit verlängerten Zugriffsbewegungszeiten zu rechnen. Im eng gekoppelten Fall kann diese Aufgabe vom BS (eine Kopie) direkt übernommen werden. Bei DB-Sharing sieht jedes BS nur seine eigenen Plattenzugriffe, was keine globale Optimierung zuläßt.

Der Anschluß von n Rechnern an eine Platte dürfte selbst für $n \leqslant 10$ technisch sehr schwierig sein (Multi-Ports). Die Programmierung der E/A, die alle Fehler- und Sonderfälle zu berücksichtigen hat, würde äußerst komplex und damit fehleranfällig sein. Deshalb wird eine machbare und zuverlässige Lösung des gemeinsamen Zugriffs im Einsatz von Spezialprozessoren (intelligente Platten-Controller) zu suchen sein, die jeweils eine Platte verwalten, die Ausschlußsynchronisation der Rechner vornehmen, Synchronisationstabellen auf Datei-Ebene führen und für Optimierungen wie Minimierung der Zugriffsarmbewegungen verantwortlich sind.

3.3 Rechnerkopplung

Die Rechnerkopplung, die die Art der Kommunikation zwischen den Prozessoren festlegt, beeinflußt viele Systemeigenschaften in hohem Maße. Wir unterscheiden hier zwischen **loser** (loosely coupled) und **enger** Kopplung (tightly coupled). Die **nahe** Kopplung (closely coupled) wird zusätzlich als eine Art Optimierung und Kompromiß eingeführt. Bei der Beurteilung der Kopplungsart muß vor allem berücksichtigt werden, welche Einflüsse sich für die **Synchronisation, Kooperation** und **Interferenz** der Prozesse ergeben.

1. Feste oder enge Kopplung (Bild 4a)

Von enger Kopplung spricht man, wenn alle Rechner sich **einen Hauptspeicher teilen** und nur **eine Kopie des BS** (und auch des DBVS) existiert. Üblicherweise hat aber jeder Rechner seinen eigenen Cache-Speicher zur Unterstützung sehr schneller Speicherreferenzen. Da nur eine BS-Kopie existiert, kann jeder Rechner auf alle Platten zugreifen.

Dieser Ansatz bringt einige wesentliche **Vorteile** und ist auch relativ weit verbreitet (Bi- und Quadroprozessoren). Durch die zwei oder mehr Prozessoren ist die Leistungsfähigkeit und auch die Verfügbarkeit besser als für Monoprozessoren. Darüber hinaus bleibt die Tatsache, daß n Prozessoren zur Verfügung stehen, nach außen hin unsichtbar, d.h., es wird sowohl für Endbenutzer und Anwenderprogrammierer als auch für Systemprogrammierer und Operateure ein **single system image** gewährleistet.

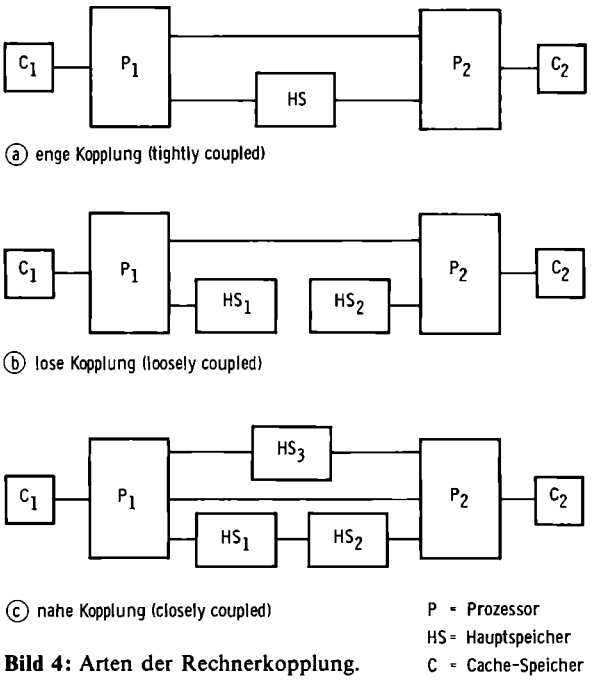

Bild 4: Arten der Rechnerkopplung.

Zur Synchronisation zwischen den Prozessen auf gleichen oder verschiedenen Prozessoren können direkte Protokolle (Semaphore) auf gemeinsamen Datenstrukturen verwendet werden. Auch die Kooperation zwischen den Prozessen sowie Logging und Recovery kann wie im Ein-Prozessor-Fall erfolgen. Die Verarbeitung ist damit durch folgende Charakteristika gekennzeichnet:

- Globale Dienste können synchron (ohne Deaktivierung) angefordert werden (z.B. Unterprogrammaufruf an einen Sperrverwalter).
- Globale Variablen (z.B. Systempuffer) sind nur in einer Kopie vorhanden.
- Kommunikation zwischen Prozessen kann sowohl durch Kopieren von Daten (MOVE-Mode) wie durch Referenzieren der Daten (LOCATE-Mode) erfolgen.

Dennoch hat die enge Kopplung entscheidende **Nachteile** im Hinblick auf hohe Leistungs- und Verfügbarkeitsforderungen. Aufgrund des zunehmenden Wettbewerbs auf dem gemeinsamen Hauptspeicher kann durch Vermehren der Prozessoren i.a. **kein lineares Durchsatzwachstum** erzielt werden. Die Zahl der Prozessoren, die eng gekoppelt werden, liegt daher meist unter 5. Eine weitere Interferenz stellt auch die Invalidierung von Datenobjekten in den lokalen Cache-Speichern dar **(Cache Invalidiation).** Um zu verhindern, daß auf veraltete Objekte in einem Cache zugegriffen wird, müssen Änderungen den anderen Rechnern (z.B. über aufwendige Hardware-Mechanismen durch Broadcasting) mitgeteilt werden, und ein geändertes Datum ist auf den langsamen Hauptspeicher zurückzuschreiben (store-through).

Neuere Architekturen [22] versuchen diese Nachteile durch extrem schnelle Kommunikationsverbindungen (Busse) und große Cache-Speicher zu vermeiden, um li-

neares Durchsatzwachstum auch für wesentlich mehr als 4 Prozessoren zu gewährleisten. Gesicherte Aussagen über das Betriebsverhalten solcher Systeme liegen aber noch nicht vor.

Ein Hauptgrund, der gegen den Einsatz enggekoppelter MRDBS spricht, ist das **Verfügbarkeitsproblem** wegen des gemeinsamen Hauptspeichers bzw. wegen der gemeinsam benutzten BS-Kopie. So kann ein Software-Fehler im Betriebssystem beispielsweise einen kritischen Abschnitt für immer gesperrt halten, woraufhin irgendwann alle Rechner darauf warten. Der gemeinsame Hauptspeicher birgt die Gefahr der Fehlerfortpflanzung, so daß z. B. eine verseuchte Datenstruktur einen Systemausfall nach dem anderen verursacht.

2. Lose Kopplung (Bild 4b)

Bei der losen Kopplung besitzt jeder Rechner einen eigenen Hauptspeicher und eine eigene Kopie des BS sowie des DBVS **(autonome Rechner).** Die Kommunikation zwischen den Rechnern geschieht ausschließlich über **Nachrichten,** die über Leitungen ausgetauscht werden. Für lokale MRDBS sind dabei eine Vielzahl von Kommunikationswegen denkbar, z. B. Kanalkopplungen (channel-to-channel-adapter), ein gemeinsamer Bus oder Kreuzpunkt-Schalter. Die Platten können von den Rechnern privat oder gemeinsam benutzt werden.

Die Autonomie der Rechner führt dazu, daß globale Variablen in bezug auf die DB-Verarbeitung ggf. in mehreren Kopien vorhanden sind. Eine mögliche Interferenz davon ist das sogenannte **Veralterungsproblem** [23], das durch mehrfache Systempuffer entstehen kann. Denn da eine Datenseite potentiell in mehreren Puffern vorliegt, erzeugt die Änderung in einem Rechner veraltete Kopien der Seite in den anderen Rechnern. Der Zugriff auf invalidierte Objekte muß also durch geeignete Maßnahmen verhindert werden. In ähnlicher Weise betrifft dieses Interferenzproblem alle mehrfach geführten Datenstrukturen.

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 teure Kommunikation zwischen den Rechnern. Trotz hoher Bandbreiten ist es nach Abschicken einer Nachricht nicht möglich, synchron (d. h. unter Beibehaltung der CPU) auf eine Antwort zu warten. Die Kommunikation erfolgt also **asynchron,** verbunden mit Prozeßwechseln. Da Kooperation und Synchronisation von Prozessen auf verschiedenen Prozessoren ausschließlich über nachrichtenbasierte Protokolle bewerkstelligt werden müssen, brauchen leistungsfähige Systeme eine schnelle Interprozessor-Kommunikation mit geringem Software-Overhead. Zusätzlich muß das Ausmaß der Kommunikation durch geeignete Algorithmen auf ein Minimum reduziert werden.

Im Gegensatz zur festen Kopplung liegt zunächst kein single system image vor. Es müßte durch eine komplette Software-Abbildung realisiert werden, um Wartung, Tuning und Handhabbarkeit zu erleichtern.

3. Nahe Kopplung (Bild 4c)

Wie bei der losen Kopplung verfügt hier jeder Rechner über einen eigenen Hauptspeicher und eine eigene Kopie von BS und DBVS. Der Unterschied besteht darin, daß die Kommunikation zwischen den Rechnern so effizient abgewickelt werden kann, daß es einem Prozeß möglich ist, **synchron** auf eine Antwort zu warten (Antwort im μs-Bereich). Dies läßt sich z. B. durch gemeinsam benutzte (möglicherweise nichtflüchtige) Speichersegmente oder Verwendung von speziellen Hardware-Einheiten erreichen.

Die Synchronisation zwischen Prozessen auf verschiedenen Rechnern kann damit wie im eng gekoppelten Fall über direkte Protokolle (Semaphore) oder durch spezielle Hardware erfolgen. Ein Informationsaustausch zur Kooperation ist entweder über eine gemeinsame Hauptspeicher-Partition oder durch expliziten Nachrichtenaustausch möglich. Interferenz kann hier durch globale (DBVS-)Variablen entstehen, die nur in einer Kopie vorliegen.

Wenn jeder Rechner seinen eigenen Systempuffer hat, kann es wiederum zum Veralterungsproblem kommen. Dennoch bietet die nahe Kopplung wegen der höheren physischen Entkopplung und stärkeren Separierung von BS-Variablen weniger Interferenz-Probleme als eng gekoppelte Rechner.

Die nahe Kopplung kann also den Hauptnachteil der losen Kopplung (teure Kommunikation) entschärfen, jedoch i. a. zu Lasten einer verringerten Verfügbarkeit. Es ist daher beispielsweise notwendig, dem Ausfall des gemeinsamen Speicherbereiches bzw. der speziellen Hardware vorzubeugen. Außerdem wird die Erweiterbarkeit beeinträchtigt, wenn nur eine bestimmte Anzahl von Rechnern nah gekoppelt werden können.

3.4 Räumliche Aufstellung

Bei loser Kopplung spielt die räumliche Aufstellung der Rechner keine Rolle, da alle Algorithmen nachrichtenbasiert ablaufen. Zusammen mit dem partitionierten Zugriff ergibt sich die Architektur von VDBS, für die bereits viele Algorithmen und Verfahren, z. B. zur Synchronisation und zur Recovery, bekannt sind. Dabei wird gewöhnlich für die ortsverteilte Kommunikation eine geringe Bandbreite (1–100 KBytes/sec) angenommen. Bei lokalen MRDBS, die wir hier hauptsächlich betrachten wollen, ergeben sich für die Klasse „DB-Distribution" eine Reihe von Optimierungen, die vor allem durch Ausnutzung deutlich höherer Bandbreiten bewirkt werden. Typischerweise hat eine LAN-Kopplung eine Bandbreite von 1–100 MBytes/sec.

Gemeinsamer Zugriff oder nahe bzw. enge Kopplung setzen die lokale Aufstellung der Rechner (in einem Raum) voraus. DB-Sharing ergibt sich dabei als neue Architekturklasse lokaler MRDBS.

4 Allgemeine Systemeigenschaften von DB-Sharing- und DB-Distribution-Architekturen

Die bisherige Diskussion hat gezeigt, daß eng gekoppelte MRDBS nicht geeignet erscheinen, die in Kap. 2 genannten Anforderungen an Hochleistungs-Datenbanksysteme (v.a. hinsichtlich Verfügbarkeit und Erweiterbarkeit) zu erfüllen. Dies kann auch durch die Vorteile dieser Realisierungsform (Einfachheit, effiziente Kommunikation und Kooperation, single system image) nicht wettgemacht werden. Die weiteren Überlegungen konzentrieren sich daher auf DB-Sharing- und DB-Distribution-Systeme, deren Grobarchitektur in Bild 5 skizziert ist [24]. Wir gehen dabei von einer losen Rechnerkopplung aus sowie, wenn nicht anders gesagt, von einer räumlich benachbarten Anordnung der Rechner. Einsatzformen einer nahen Kopplung, die zur Optimierung einzelner Systemfunktionen denkbar sind, werden in [23] betrachtet.

In Bild 5 wurde angenommen, daß jedem Rechner ein DBS (sowie ein DC-System) zugeordnet ist, das jeweils seinen eigenen Systempuffer (SP) verwaltet.

Kommunikations- und Synchronisationsanforderungen der beteiligten DBS implizieren eine breitbandige Kommunikationsmöglichkeit zwischen den Rechnern, z.B. einen Hochgeschwindigkeitsbus. Die Terminal-Anbindung ist in Bild 5 lediglich durch eine naheliegende Möglichkeit skizziert; mehrere Front-Ends (Fehlertoleranz) übernehmen gewisse Aufgaben des DC-Systems wie etwa die Verteilung der Transaktionen und die Lastkontrolle.

Die offensichtlichen Unterschiede beider Alternativen, die sich aus Bild 5 extrahieren lassen, dienen zunächst zu ihrer Grobcharakterisierung:

- **DB-Sharing:** Alle Rechner und damit alle DBS können auf alle Externspeicher und damit auf alle Daten der DB direkt zugreifen. Sie verwalten gemeinsam die DB; im Fehlerfall übernehmen alle oder einige der überlebenden Systeme kooperativ die zusätzlichen Aufgaben.
- **DB-Distribution:** Jeder Rechner bzw. das DBS verwaltet eine DB-Partition und kann jeweils nur auf seine Partition zugreifen. Daten aus anderen Partitionen müssen explizit angefordert und ausgetauscht werden. Systemausfälle erzwingen Übernahme oder Neuaufteilung der zugehörigen DB-Partition durch einen (oder mehrere) Rechner, was die Erreichbarkeit der Daten voraussetzt.

Architektur und Algorithmen eines MRDBS hängen sicher stark von seiner geplanten Einsatzgröße ab. Beide Alternativen können aus unterschiedlichen Gründen nicht beliebig wachsen (z.B. Kanalanbindung oder Datenpartitionierung). Deshalb orientieren wir uns bei unseren Überlegungen an einer Rechneranzahl ≤ 10; bei einer Leistung von 10–30 MIPS pro Rechner dürfte das für praktische Systeme in der nächsten Zukunft keine Einschränkung bedeuten.

Der Vergleich der beiden Architekturen erfolgt im wesentlichen mit Hinblick auf die Erfüllbarkeit der in Kap. 2 aufgestellten Anforderungen und die dabei zu erwartenden Schwierigkeiten und Kosten. Das Profil der abzuarbeitenden TA-Last hat dabei entscheidenden Einfluß darauf, wie effizient die Last von mehreren Rechnern abgearbeitet werden kann. Die Bewertung der Architekturen muß notwendigerweise qualitativ bleiben, da noch keine praktischen Vergleiche vorliegen und Simulationsergebnisse bestenfalls für einzelne Komponenten bekannt sind.

4.1 Leistung

Voraussetzung zur Erreichung hoher TA-Raten und kurzer Antwortzeiten in einem lose gekoppelten MRDBS ist, einen hohen Grad an Lokalität bei der TA-Verarbeitung (d.h. Minimierung der Außenbeziehungen) zu erzielen. Dies ist von der Lastkontrolle [26] anzustreben, in dem sie eine TA dem Rechner zuordnet, auf dem ihre Verarbeitung eine minimale Anzahl von externen Referenzen oder Kommunikationen erfordert.

- Da bei DB-Sharing jeder Rechner direkt auf alle Daten zugreifen kann, ist die eigentliche TA-Verarbeitung immer lokal; Kommunikation wird im wesentlichen zur Synchronisation der DB-Zugriffe (und möglicherweise zum Holen von Datenseiten aus fremden Puffern) erforderlich. Die Lastkontrolle hat also zu einer TA den Rechner zu bestimmen, auf dem eine weitgehend lokale Synchronisation möglich ist.
- Bei DB-Distribution ist derjenige Rechner zu bestimmen, in dessen Partition die meisten der von einer TA benötigten Daten vorliegen. Die Verarbeitung nichtlokaler Daten wird i.a. durch Verschicken von DB-Operationen (DML-Befehlen) sowie von Ergebnissen bewerkstelligt. Dies impliziert ein verteiltes Zwei-Phasen-COMMIT (und bei Scheitern einer TA ein verteiltes Rücksetzen) zwischen den beteiligten DBS.

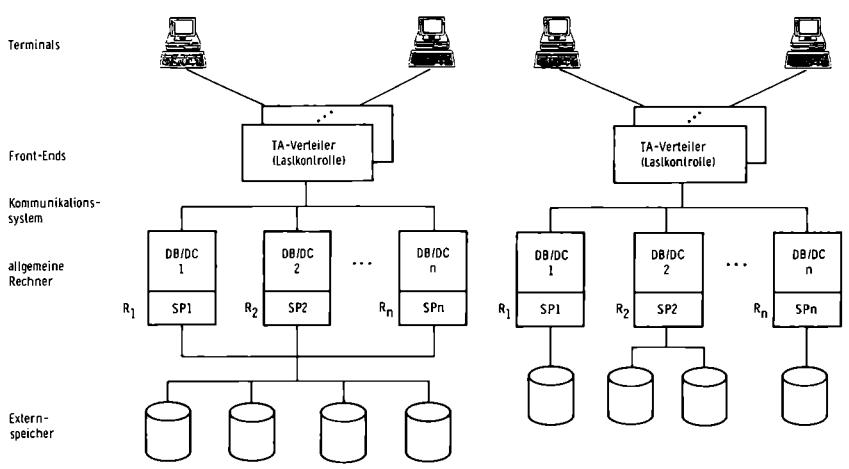

Bild 5: Grobarchitektur von DB-Sharing- und DB-Distribution-Systemen.

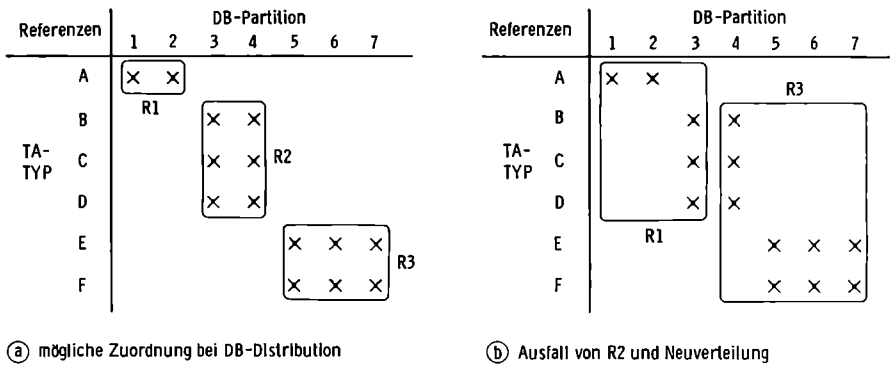

Bild 6: Eine ideale Last für ein Transaktionssystem.

Da die Datenverteilung stets bekannt ist, erlaubt DB-Distribution i. a. eine einfachere Realisierung der Lastkontrolle als DB-Sharing, bei dem eine Abstimmung mit dem verwendeten Synchronisationsalgorithmus notwendig ist. Allerdings schränkt die statische Aufteilung der Daten, die auf Grund von vagem Wissen bezüglich eines erwarteten Referenzverhaltens vorgenommen werden mußte, die Flexibilität der Lastkontrolle auch stark ein.

Neben der Unterstützung von Lokalität muß die Lastkontrolle auch eine gleichmäßige Auslastung der Rechner und die Verhinderung von Überlastsituationen gewährleisten. Dies ist bei DB-Distribution weitaus schwieriger zu realisieren als bei DB-Sharing, da ein Rechner alle Operationen (lokaler und externer TA) auf die ihm zugeordnete Datenpartition durchführen muß.

Daher ist eine gleichmäßige Auslastung der Rechner nur möglich, solange auf jede Datenpartition in etwa gleich viele DML-Befehle abgesetzt werden. DB-Sharing erlaubt dagegen mehr Freiheitsgrade bei der Aufteilung der TA-Last, da die Daten den Rechnern nicht fest zugeordnet sind.

Hohe TA-Raten mit kurzen Antwortzeiten sind vergleichsweise leicht zu realisieren, wenn eine **„ideale" Last** (Bild 6a) vorliegt.

Die ideale Last weist eine Partitionierung von TA-Typen und Daten derart auf, daß

- eine Zuordnung von disjunkten Mengen von TA-Typen zu disjunkten Datenbereichen vorgenommen werden kann
- eine gleichmäßige Verteilung die Last, was Aufwand der Verarbeitung betrifft, statisch auf die beteiligten Rechner erfolgen kann und sich das Lastaufkommen auch zeitlich nicht ändert.

Solche Lasten, bei denen zusätzlich geringe Synchronisationsprobleme unterstellt werden, werden nach [13] als „delightful transactions" bezeichnet:

- DB-Distribution: Es verlangt eine statische Zuordnung der DB-Partitionen, wie in Bild 6a gezeigt. Damit ist die dynamische Verteilung der TA (z. B. bei Veränderung der Lastzusammensetzung) eingeschränkt (mit Leistungseinbußen verbunden). Im idealen Fall wird die gesamte TA-Verarbeitung lokal bewerkstelligt.
- DB-Sharing: Eine statische Zuordnung der DB-Partitionen zu Rechnern ist nicht erforderlich. Abhängig von der jeweiligen Lastsituation und der momentanen Verarbeitungslokalität in den Rechnern kann eine beliebige Verteilung der TA dynamisch vorgenommen werden. Auch hier ist eine (weitgehend) lokale Verarbeitung bei idealen Lasten möglich.

Es ist also zu erwarten, daß eine ideale Last von beiden Architekturen im Normalfall gleich gut und effizient bearbeitet werden kann, da sie eine hohe Verarbeitungslokalität bei gleichmäßiger Rechnerauslastung zuläßt. Wie aus empirischen Untersuchungen hervorgeht, weisen **reale Lasten** jedoch deutlich andere Charakteristika auf als die ideale Last, so daß ein effektiver Einsatz von MRDBS erschwert wird:

- In jeder Last gibt es dominierende TA-Typen und DB-Partitionen. Häufig referenzierte DB-Partitionen können bei DB-Distribution zu einer Überlastung des Rechners führen, dem sie zugeordnet sind. Für einen TA-Typ kann Lokalität nur innerhalb **eines** Rechners unterstützt werden; Leistungseinbußen sind daher unvermeidlich, sobald die Aktivierungen eines TA-Typs von einem Rechner nicht mehr bewältigt werden können.
- Dominierende TA-Typen referenzieren oft fast alle wichtigen DB-Partitionen. Dies führt zwangsläufig zu einer hohen Kommunikationsnotwendigkeit, da dann dieselben Partitionen von TA auf verschiedenen Rechnern referenziert werden. Bei DB-Sharing läßt sich Kommunikation einsparen, solange nur lesend auf dieselben Objekte zugegriffen wird (setzt jedoch ein geeignetes Synchronisationsverfahren voraus). Eine Entschärfung des Problems ist u. U. durch einen angepaßten Entwurf der TA-Typen möglich, indem man z. B. einen allgemeinen TA-Typ durch mehrere zugeschnittene TA-Typen ersetzt.
- In jeder realen Anwendung ist mit stärkeren zeitlichen Schwankungen im Lastaufkommen zu rechnen (kurzfrisitige Perioden jeden Tag und langfristige Verschiebungen im Verlauf von Monaten/Jahren). Während DB-Sharing solche Lastschwankungen relativ leicht durch die Lastkontrolle verkraften kann, ergeben sich bei DB-Distribution Leistungseinbußen, da es z. B. ausgeschlossen ist, die Datenverteilung mehrmals täglich anzupassen.

Diese Ausführungen zeigen, daß die von der DB-Sharing-Architektur ermöglichte Flexibilität bei der Lastkontrolle bei realen Lasten auf Leistungsvorteile von DB-Sharing hindeutet. Allerdings setzt dies voraus, daß die komplexere Realisierung der Lastkontrolle (und der Synchronisation) zufriedenstellend gelöst werden kann. In [26] werden mögliche Lösungen genannt, die eine Überlegenheit von DB-Sharing in diesem Punkt bestärken.

4.2 Verfügbarkeit und Fehlertoleranz

Für beide Ansätze kann von einer ausreichenden Redundanz zur Tolerierung von Hardware- und Software-Fehlern ausgegangen werden; die lose Rechnerkopplung begrenzt die Ausbreitung von Fehlern. Da bei DB-Distribution auch die Platten partitioniert sind („shared nothing"), dürften Modularität und Fehlerisolation noch besser als für DB-Sharing sein.

Welche Probleme ergeben sich jedoch bei Ausfall eines Rechners für die TA-Verarbeitung?

- DB-Distribution: Die DB-Partition des ausgefallenen Rechners muß durch andere Rechner übernommen werden, um die Erreichbarkeit der Daten zu gewährleisten. Hierzu muß jede Platte mit mindestens zwei Rechnern verbunden sein, wobei jedoch im Normalbetrieb nur von einem Rechner aus auf die Platte zugegriffen wird. Eine Möglichkeit der Zuordnung im Fehlerfall für die Last aus Bild 6a (R2 ist ausgefallen) ist in Bild 6b veranschaulicht. Es ergibt sich zwangsläufig eine ungünstigere Lastverteilung während des Ausfalls, wenn unterstellt wird, daß eine externe Referenz teurer als eine lokale ist.
- DB-Sharing: Es ist keine Neuaufteilung der Daten erforderlich, da alle Rechner alle Daten erreichen können. Ein überlebender Rechner oder alle zusammen können die Last des ausgefallenen Rechners übernehmen. Die Lokalität der Verarbeitung wird hier von der Qualität der (dynamischen) Lastkontrolle bestimmt.

Der Aspekt der Verfügbarkeit, der die Erreichbarkeit der Daten betrifft, spricht also zugunsten von DB-Sharing.

4.3 Wachstum

Die Forderung nach modularem Wachstum, mit dem v.a. einer (langfristigen) Erhöhung der Last begegnet werden soll, äußert sich in unterschiedlichen Bereichen. Zum einen wächst die Zahl der Datenausprägungen in vorhandenen DB-Partitionen, und es kommen neue DB-Partitionen hinzu. Zum anderen bleibt auch die Anzahl der Transaktionen/Zeiteinheit nicht gleich; neue Anwendungen erfordern auch neue TA-Typen. Für die gestiegene Last muß zusätzliche Leistung und zusätzlicher Speicherplatz bereitgestellt werden. Bei den Prozessoren kann in der Regel ein Teil der Last durch leistungsfähigere abgefangen werden; es muß jedoch auch die Notwendigkeit der Integration eines weiteren Rechners in Betracht gezogen werden. Die Erhöhung der Kapazität einzelner Platten ist nicht immer eine Ausweg, wenn eine Platte nicht überlastet werden soll (Kanal $\leq 30\%$, Platte $\leq 50\%$ Auslastung). Deshalb wird das Wachstum der Daten oft eine **physische** Neuverteilung vorhandener DB-Partitionen und eine Zuordnung neuer DB-Partitionen auf zusätzliche Platten erzwingen. Das entstehende Problem ist grob in Bild 7 angedeutet:

- DB-Distribution: Ein neuer Rechner impliziert die (Neu-)Aufteilung der DB ($n \rightarrow n+1$). Dabei sind die Verteilungseinheiten oft durch das Datenmodell vorgegeben (z.B. Area in CODASYL-DBS), so daß keine beliebigen Anpassungen (Vorsorge beim DB-Schema-Entwurf) möglich sind. Eine Änderung in der Externspeicherzuordnung ist zudem sehr umständlich und erschwert eine gute Lastverteilung. Das Wachstum bestehender Partitionen kann nur durch Plattenzuordnung innerhalb von Partitionen (also relativ grob und in großen Einheiten) aufgefangen werden.
- DB-Sharing: Da alle Platten gemeinsam von allen Rechnern verwaltet werden, bilden das Wachstum der Daten und ihre physische Zuordnung vergleichsweise geringe Probleme. Die physische Verteilung der Daten hat nur zu berücksichtigen, daß keine Überlastung einzelner Platten auftritt. Wegen der dynamischen Lastverteilung kann auch ein weiterer Rechner ohne Neuverteilung integriert werden. Insbesondere ergeben sich durch die gleiche logische Sicht aller DBS auf die Daten keine Rückwirkungen auf den DB-Schema-Entwurf.

Referenzen		DB-Partition 1	2	3	4	5	6	7	8
TA-TYP	A	×	×						
	B			×	×				
	C			×	×				
	D			×	×				
	E					×	×	×	
	F					×	×	×	
	G	×					×		×
	H				×				×

Bild 7: Modulares Wachstum bei der idealen Last.

Der Aspekt des modularen Wachstums verspricht wegen der Systemkopplung über gemeinsame Daten (DB) einfachere Lösungen beim DB-Sharing-Ansatz. Inwieweit ein lineares Durchsatzwachstum (ohne deutliche Antwortzeitverschlechterungen in Kauf zu nehmen) erreicht werden kann, hängt natürlich stark von der abzuarbeitenden TA-Last und der Qualität der Lastkontrolle ab. Hierzu gelten die in 4.1 gemachten Aussagen.

4.4 Handhabbarkeit und einfache Verwaltung

Der Aufwand hierfür (insbesondere zur Realisierung eines single system image) dürfte bei lokaler Anordnung der Rechner für DB-Distribution und DB-Sharing in etwa gleich hoch sein. Eine Ortsverteilung der Rechner erfordert i.a. eine lokale Systemverwaltung in jedem Knoten. Dies bedeutet einerseits einen Mehrbedarf an qualifiziertem Bedienpersonal und erschwert die Verwaltung des Gesamtsystems. Andererseits wird den einzelnen Knoten eine relativ hohe Autonomie zugestanden, die in großen Organisationen vielfach wünschenswert ist. Inbesondere kann jeder Knoten eine Abstimmung auf lokale Bedürfnisse vornehmen.

5 Zusammenfassung

Es wurden die wichtigsten Anforderungen an Transaktionssysteme – Leistung, Verfügbarkeit, Wachstum, Handhabbarkeit – spezifiziert und genauer erläutert. Viele Argumente verweisen auf die Einführung von MRDBS zu ihrer Realisierung. Deshalb wurden die Lösungsmöglichkeiten für MRDBS klassifiziert und auf ihre Tauglichkeit für Transaktionssysteme hin bewertet. Zwei Systemarchitekturen – DB-Distribution und DB-Sharing genannt – wurden daraufhin näher untersucht, wobei insbesondere ihre Eigenschaften zur Gewährleistung hoher Verfügbarkeit, zur Möglichkeit modularen Wachstums und zur Bearbeitung hoher Transaktionsraten analysiert wurden.

Bei dem Vergleich der beiden Systemarchitekturen ergaben sich für DB-Distribution deutliche Nachteile hinsichtlich Verfügbarkeit und Erweiterbarkeit, da Änderungen an der Datenverteilung, wie sie nach Ausfall oder Hinzukommen eines Rechners notwendig werden, umständlich und nicht beliebig möglich sind. Obwohl die gewählte Betrachtungsebene keine abschließende Beurteilung über die Leistung (hohe TA-Raten, kurze Antwortzeiten) zuläßt, verspricht auch hier DB-Sharing dank seines größeren Optimierungspotentials bei der Lastkontrolle effizientere Lösungen. Eine genauere Bewertung muß insbesondere den Implementierungsaufwand einzelner Systemkomponenten berücksichtigen; hier deutet bei DB-Sharing vieles auf eine komplexe Realisierung einzelner Funktionen (Lastkontrolle, Synchronisation) hin [3, 25]. Fragen der Ortsverteilung, die in der Praxis auch oft eine Rolle spielen [24], werden natürlich klar zugunsten von DB-Distribution entschieden.

Literatur

[1] *Härder, T.; Meyer-Wegener, K.:* Transaktionssysteme und TP-Monitore – Eine Systematik ihrer Aufgabenstellung und Implementierung, in: Informatik – Forschung und Entwicklung, Vol. 1, No. 1, 1986, S. 3–25.

[2] *Bayer, R.; Elhardt, K.; Kießling, W.; Killar, D.:* Verteilte Datenbanksysteme. Eine Übersicht über den heutigen Entwicklungsstand, in: Informatik-Spektrum, Band 7, Heft 1, Feb. 1984, S. 1–19.

[3] *Reuter, A.:* Mehrprozessor-Datenbanksysteme – ein Überblick über die wichtigsten Entwurfsprobleme, in: Tagungsband zur 9. GI/NTG-Fachtagung über „Architektur und Betrieb von Rechensystemen", Stuttgart, März 1986, VDE-Verlag, NTG-Fachberichte, S. 141–150.

[4] *Gray, J.:* Notes on Database Operating Systems, in: Operating Systems: an Advanced Course, Bayer, R.; Graham, R. M., Seegmüller, G. (eds.), Lecture Notes on Computer Science 60, Springer Verlag, 1978, pp. 393–481.

[5] *Anon et al.:* A Measure of Transaction Processing Power, in: Datamation, April 1985.

[6] *Gray, J.; Good, B.; Gawlick, D.; Homan, P.; Sammer, H.:* One Thousand Transactions per Second, in: Proc. IEEE-Spring CompCon, San Francisco, Feb. 1985, pp. 96–101.

[7] *Kim, W.:* Highly Available Systems for Database Applications, in: ACM Computing Surveys, Vol. 16, No. 1, March 1984, pp. 71–98.

[8] *Gray, J.:* Why do Computers Stop and what can be done about it, in: Proc. „Büroautomation '85", Wedekind, H., Kratzer, K. (Hrsg.), Berichte des German Chapter of the ACM 25, Teubner-Verlag, Okt. 1985, S. 128–145.

[9] *Serlin, O.:* Fault-Tolerant Systems in Commercial Applications, in: IEEE Computer, August 1984, pp. 19–30.

[10] *Härder, T.; Reuter, A.:* Principles of Transaction-Oriented Database Recovery, in: ACM Computing Surveys, Vol. 15, No. 4, Dezember 1983, pp. 287–317.

[11] *Bartlett, J. F.:* A 'NonStop' Operating System, in: Proc. International Conference on System Sciences, Honolulu, Hawaii, Jan. 1978.

[12] *Reuter, A.; Shoens, K.:* Synchronization in a Data Sharing Environment, IBM San Jose Research Laboratory (in Vorbereitung).

[13] *Stonebraker, M.:* The Case for Shared Nothing, Int. Workshop on High Performance Transaction Systems, Asilomar, Sept. 1985.

[14] *William, R., et al.:* R* – An Overview of the Architecture, in: Improving Database Usability and Responsiveness, P. Scheuermann (ed.), Academic Press, 1982, pp. 1–27.

[15] *IEEE Quarterly Bulletin of Database Engineering,* Vol. 6, No. 2, Juni 1983 (special issue on fault-tolerant systems).

[16] *Aghili, H., et al.:* A Prototype for a Highly Available Database System, IBM Research Report RJ 3755, San Jose, Calif., Jan. 1983.

[17] *Helland, P.:* Transaction Monitoring Facility (TMF), in: Database Engineering, Vol. 8, No. 2, Juni 1985, pp. 11–18.

[18] *Keene, W. N.:* Data Sharing Overview, in: IMS/VS Vl, DBRC and Data Sharing User's Guide, Release 2, G320-5911-0, August 1982.

[19] *Shoens, K., et al.:* The AMOEBA Project, in: Proc. IEEE Spring CompCon, San Francisco, Feb. 1985, pp. 102–105.

[20] *Bernstein, P. A.:* Sequoia – a Fault-tolerant Tightly-coupled Computer for Transaction processing, Int. Workshop on High Performance Transaction Systems, Asilomar, Sept. 1985.

[21] *Boral, H.; Redfields, S.:* Database Machines Morphology, in: Proc. 11th VLDB, Stockholm, 1985, pp. 59–71.

[22] *Olson, R.:* Parallel Processing in a Message-Based Operating System, in: IEEE Software, Juli 1985, pp. 39–49.

[23] *Rahm, E.:* Nah gekoppelte Rechnerarchitekturen für ein DB-Sharing-System, in: Tagungsband zur 9. GI/NTG-Fachtagung über „Architektur und Betrieb von Rechensystemen", Stuttgart, März 1986, S. 166–180.

[24] *Härder, T.:* DB-Sharing vs. DB-Distribution – die Frage nach dem Systemkonzept zukünftiger DB/DC-Systeme, in: Tagungsband zur 9. GI/NTG-Fachtagung über „Architektur und Betrieb von Rechensystemen", Stuttgart, März 1986, S. 151–165.

[25] *Reuter, A.:* Load Control und Load Balancing in a Shared Database Management System, Proc. 2nd Data Engineering Conf., 1986.

[26] *Rahm, E.:* Algorithmen zur effizienten Lastkontrolle in Mehrrechner-Datenbanksystemen, in: Angewandte Informatik, Vol. 28, No. 4, April 1986, S. 161–169.

Diese Arbeit wurde teilweise von der Siemens AG finanziell unterstützt.

Prof. Dr. *Theo Härder* (40), Studium der Elektrotechnik an der TH Darmstadt (1966–1971), seit 1972 Mitarbeit im Fachbereich Informatik (Datenverwaltungssysteme) der TH Darmstadt, 1975 Promotion, 1976 Post-doctoral Fellow in IBM Research, San Jose (Mitarbeit im Projekt „System R"), 1977 Dozent, 1978 Professor für Informatik an der TH Darmstadt, seit 1980 Professor für Praktische Informatik (Datenverwaltungssysteme) an der Universität Kaiserslautern. Besondere Forschungsinteressen: Leistungsanalyse von DB- und DB/DC-Systemen, Mehrrechner-Datenbanksysteme, Non-Standard-Anwendungen für DBS (CAD/CAM), Datenbanken in Netzen.

Dipl.-Inform. *Erhard Rahm* (27), Studium der Informatik (1979–1984) an der Universität Kaiserslautern mit Nebenfach Mathematik und Schwerpunkt Datenbanksysteme. Seitdem wissenschaftlicher Mitarbeiter am Fachbereich Informatik der Universität Kaiserslautern. Interessen- und Forschungsschwerpunkte: Mehrrechner-Datenbanksysteme, Leistungsbewertung von Datenbanksystemen sowie Synchronisation und Recovery.

Anschrift der Verfasser: Universität Kaiserslautern, Postfach 3049, D-6750 Kaiserslautern.