Hochleistungs-Datenbanksysteme – Vergleich und Bewertung aktueller Architekturen und ihrer Implementierung

High Performance Database Systems – Comparison and Evaluation of Current Architectural Approaches and their Implementation

T. Härder und E. Rahm, Universität Kaiserslautern

Künftige Hochleistungs-Datenbanksysteme müssen besonderen Ansprüchen bezüglich Durchsatz und Antwortzeit, Verfügbarkeit, Erweiterungsfähigkeit sowie Handhabbarkeit gerecht werden. Die Erfüllung solcher Anforderungen verlangt den Einsatz von Mehrrechner-Datenbanksystemen, wobei sich vor allem zwei Architekturklassen – DB-Sharing und DB-Distribution – für die Realisierung anbieten.

Beide Architekturklassen werden in der Fachwelt sehr kontrovers beurteilt; man spricht gelegentlich schon – überspitzt formuliert – von einem Religionskrieg. Ziel dieses Beitrages ist es deshalb, Merkmale und Eigenschaften beider Ansätze zu vergleichen und dabei ihre Unterschiede nach verschiedenen Kriterien zu bewerten. Nach einer Diskussion grundlegender Konzepte, die die Hochleistungs-Eigenschaft von Datenbanksystemen wesentlich bestimmen, werden zunächst allgemeine Systemaspekte beider Architekturklassen wie Verfügbarkeit und Erweiterbarkeit miteinander verglichen. Den Kern des Aufsatzes bildet eine Darstellung der Möglichkeiten und Probleme bei der Realisierung derjenigen Systemkomponenten, für die im Vergleich zu zentralisierten Systemen neue Konzepte und Lösungen erforderlich sind. Dabei wird auch untersucht, wie mit Hilfe einer nahen Rechnerkopplung (z. B. über gemeinsam benutzte Halbleiterspeicher) verschiedene Verarbeitungsaspekte in Mehrrechner-Datenbanksystemen optimiert werden können. Als Abschluß versucht eine zusammenfassende Bewertung die Tauglichkeit der beiden Architekturklassen im Hinblick auf verschiedenartige Systemaspekte durch Noten auszudrükken.

Future high performance database systems are assumed to satisfy demanding requirements as far as throughput, response time, availability, modular growth, and manageability is concerned. To realize such design goals, use of multiprocessor database systems seems to be mandatory. Currently attention is focused on two different architectural approaches called DB-Sharing and DB-Distribution.

A controversial debate on the merits of both architectures accompanies the design considerations for a number of systems; somewhat exaggerated, this expert controversy is occasionally called 'a religious war'. Therefore, it is our primary goal to compare essential properties of both approaches thereby evaluating their differences according to various criteria. After a discussion of general concepts, which essentially determine the high performance qualities of database systems, we compare general system aspects of both architectures such as availability and extensibility. Then we concentrate on a description of design alternatives and implementation problems for system components which require new concepts and solutions as compared to centralized database systems. In addition, we investigate the potential of a close coupling of processors (e.g. using shared solid state memory) to optimize various processing aspects of multiprocessor database systems. We conclude our architectural evaluation by expressing the capabilities of both architectures by marks for important system aspects.

1 Einleitung

Der schnell zunehmende Rechnereinsatz in (fast) allen Bereichen unseres Lebens hat zur Folge, daß auch immer mehr Anwendungen konzipiert und eingeführt werden, die eine gemeinsame Nutzung von Daten voraussetzen. Da für die Abwicklung solcher Anwendungen vor allem Datenbanksysteme (DBS) vorteilhaft eingesetzt werden können, wachsen sowohl ihre Funktions- als auch ihre Leistungsanforderungen rasant. Sogenannte Non-Standard-Anwendungen für DBS wie die Durchführung von Ingenieuraufgaben (CAD/CAM) oder die Unterstützung von regelbasierten Ableitungen (Expertensysteme) zeichnen sich durch eine große Anzahl von mächtigen Benutzeroperationen aus, deren Ausführung durch komplexe Transaktionen zusammenzufassen ist. Natürlich impliziert ihr vorwiegender Einsatz in interaktiven Entwurfssystemen harte Zeitrestriktionen für die Antwortzeit, was schon bei konkurrierenden Operationen weniger Benutzer oder gar bei komplexen Operationen eines Benutzers die Ausnutzung von Parallelität im Rahmen von **Mehrrechner-Datenbanksystemen** nahelegt oder sogar erzwingt [HMMS87, RDPSZ86]. Aber auch in konventionellen Datenbankanwendungen (Buchungen, Reservierungen usw. in vielen verschiedenen Bereichen) beobachtet man eine rapide Steigerung des Leistungsbedarfs, da Einsatzspektrum und Nutzungsfrequenz solcher Transaktionssysteme [HM86a] ständig größer werden. Obwohl in solchen Systemen in der Regel nur kurze Transaktionen

mit wenigen einfachen Datenbankoperationen ausgeführt werden, erfordern auch hier enge Antwortzeitgrenzen und sehr hohe Transaktionsraten [Gr85] Mehrrechner-Datenbanksysteme zur Bewältigung entsprechender Lasten.

Entwurfsziele, Architekturen und allgemeine Systemeigenschaften solcher Mehrrechner-Datenbanksysteme wurden in [HR86] ausführlich beschrieben. Sie müssen mit mindestens dem gleichen Funktionsumfang wie zentrale DBS ausgestattet sein; darüber hinaus sollen sie weitere wichtige Eigenschaften aufweisen wie

- hohe Verfügbarkeit und Robustheit gegen alle erwarteten Fehlerfälle
- modulare Erweiterungsfähigkeit bei (annähernd) linearem Durchsatzwachstum und gleichbleibenden Antwortzeiten
- leichte Handhabbarkeit und einfache Verwaltung des Gesamtsystems, wobei für seine Administration, Wartung und Bedienung ein **single system image** angeboten werden sollte.

Ihre Schlüsseleigenschaft ist jedoch ein hohes Leistungsvermögen. Hinge dieses nur von der Bereitstellung ausreichender Rechnerleistung ab, so könnte es vergleichsweise einfach durch Kopplung von n Rechnern erreicht und durch zusätzliche Rechner erweitert werden. Das Kernproblem solcher Mehrrechner-Datenbanksysteme liegt darin begründet, daß allen Anwendungen das Bild einer **zentralen Datenbank** gezeigt und aus Konsistenzgründen serialisierbare Transaktionsabläufe erzwungen werden müssen – unabhängig davon, ob alle Rechner auf die ganze Datenbank oder nur auf eine Partition von ihr zugreifen können. Deshalb ergibt sich vor allem für Synchronisation und Recovery ein Mehraufwand, der bei schlechten Lösungen und hohen Konfliktwahrscheinlichkeiten beim Datenzugriff zu einem asymptotischen Leistungsverhalten des Gesamtsystems führen kann.

Im praktischen Betrieb ist von einem stark schwankenden Leistungsbedarf auszugehen. Deshalb sind Architekturen wünschenswert, bei denen man zur Bearbeitung von Lastspitzen Rechner zum Verbund dazuschalten und in Schwachlastzeiten Rechner für andere Aufgaben abziehen kann. Eine solche Entwurfsforderung ist nicht trivial, da die Erreichbarkeit aller Daten auf Externspeichern gewährleistet bleiben muß. Die damit gewonnene Flexibilität erlaubt jedoch eine einfache Behandlung eines Rechnerausfalls oder des modularen Wachstums, so daß beide Fälle nicht zwangsläufig zu einer Betriebsunterbrechung führen müssen. Aber auch bei einer stabilen Rechneranzahl ist mit (kurzfristig) stark schwankenden Transaktionslasten zu rechnen. Da die Transaktionen (TA) selbst bei gemeinsamer Abarbeitung untereinander Unverträglichkeiten beim Datenzugriff aufweisen können, sollte eine Lastkontrolle neben einer gleichmäßigen Belastung der Rechner dafür sorgen, daß allzu konfliktträchtige TA hintereinander oder, wenn erforderlich, gemeinsam auf demselben Rechner ausgeführt werden, um eine lokale Konfliktbewältigung durchführen zu können.

Die skizzierten Probleme müssen in allen Mehrrechner-Datenbanksystemen gelöst werden. Während die allgemeinen Architekturmerkmale solcher Systeme gleich bleiben, dürften sich in den speziellen Algorithmen abhängig von den zu unterstützenden Anwendungen starke Unterschiede ergeben. Da sich die DB-Unterstützung für Non-Standard-Anwendungen noch in der Phase intensiver Forschung befindet, bevorzugen wir hier die Beschreibung von Hochleistungs-DBS für Transaktionssysteme, da ihre Systemstrukturen und Algorithmen weiter entwickelt und Leistungsaussagen teilweise bereits durch Prototyp-Entwicklungen abgesichert sind.

Im nächsten Kapitel werden zunächst einige Schlüsselkonzepte zur Realisierung von Hochleistungs-DBS vorgestellt, die auch in zentralisierten DBS einsetzbar sind. Den Schwerpunkt der Arbeit bildet ein Vergleich zwischen zwei allgemeinen Mehrrechnerarchitekturen – DB-Sharing und DB-Distribution –, die primär zur Realisierung von Hochleistungs-DBS in Frage kommen. Dabei werden zuerst allgemeinere Aspekte betrachtet, wie Auswirkungen der Systemarchitektur auf Verfügbarkeit, Erweiterbarkeit und Handhabbarkeit sowie auf TA-Verarbeitung, DB-Entwurf und Ortsverteilung. In Kapitel 4 folgt dann eine Darstellung von Möglichkeiten und Problemen bei der Realisierung derjenigen Systemkomponenten, für die im Vergleich zu zentralisierten Systemen neue Konzepte erforderlich sind. Dazu gehören die Systempufferverwaltung, die Synchronisation, die Lastkontrolle, Logging und Recovery. Nach der Untersuchung von Optimierungsmöglichkeiten über eine nahe Kopplung der Rechner (z. B. über gemeinsam benutzte Halbleiterspeicher), wird dann eine zusammenfassende Bewertung der beiden Architekturklassen für Mehrrechner-DBS vorgenommen.

2 Allgemeine Konzepte zur Realisierung von Hochleistungs-Datenbanksystemen

Bevor die speziellen Aspekte der Transaktionsverarbeitung in Mehrrechner-DBS diskutiert werden, sollen in diesem Kapitel zunächst auch in zentralisierten Datenbanksystemen einsetzbare Techniken angeführt werden, die zur Erreichung hoher TA-Raten und kurzer Antwortzeiten wesentlich sind. Von besonderer Wichtigkeit sind dabei zum einen eine effiziente Synchronisation der TA, mit der eine möglichst hohe Parallelität erzielbar sein sollte, und zum anderen eine weitgehende Reduzierung des E/A-Aufwandes durch Wahl von geeigneten Speicherungsstrukturen und Zugriffspfaden, durch eine leistungsstarke Systempufferverwaltung, die sehr große Hauptspeicher zu nutzen vermag, sowie durch schnelle Logging- und Recovery-Techniken. In Mehrrechner-DBS ist zusätzlich die Minimierung des Kommunikationsoverheads von zentraler Bedeutung. Weitere leistungsbestimmende Punkte, auf die hier jedoch nicht näher eingegangen werden kann, betreffen die Betriebssystemeinbettung des DBS (Prozeßstruktur,

Prozeßkommunikation und -synchronisation, u.ä.) sowie die Zusammenarbeit zwischen DB- und DC-System [HM86b]. Von zunehmender Wichtigkeit werden auch für zentralisierte DBS automatische Maßnahmen zur Systemkonfigurierung und Lastkontrolle angesehen, mit denen etwa die Anzahl der DBS-Prozesse oder andere Betriebsparameter (z.B. Systempuffergröße, Parallelität) dynamisch angepaßt werden können.

Synchronisation

Zur Erreichung hoher TA-Raten und kurzer Antwortzeiten sollte die Synchronisation, die zur Vermeidung von DB-Inkonsistenzen und anderer Anomalien im Mehrbenutzerbetrieb notwendig ist, mit einem Mindestmaß an TA-Blockierungen und -Rücksetzungen durchführbar sein. Dazu darf jedoch das Synchronisationsgranulat nicht zu grob gewählt werden, weil sonst unnötige Konflikte erzeugt würden; so ist statt einer Synchronisation auf Seitenebene eine Synchronisation auf Satz- bzw. Eintragsebene für Hochleistungsanforderungen besonders wichtig. Hierarchische Synchronisationskonzepte eignen sich zur Eingrenzung des Verwaltungsaufwands.

In existierenden DBS wird zumeist das einfache RX-Sperrprotokoll zur Synchronisation benutzt, wobei jedoch i.a. aus Leistungsgründen auf Serialisierbarkeit verzichtet wird, sondern zur Ermöglichung einer höheren Parallelität Lesesperren nur kurz gehalten werden (Konsistenzebene 2). *Optimistische Synchronisationsverfahren* versprechen eine hohe Parallelität auch bei Konsistenzebene 3, da die Änderungen in privaten Kopien vorgenommen werden, so daß parallele Leser immer auf eine konsistente Objektversion zugreifen können. Allerdings bleibt ihr nutzbringender Einsatz (auch nach vielfältigen Verbesserungen gegenüber dem ursprünglichen Vorschlag von [KR81]) auf weitgehend konfliktarme Anwendungen beschränkt. Allgemein einsetzbar sind sie nur in Kombination mit einem Sperrverfahren [Ra87b], was natürlich eine entsprechend hohe Komplexität des Synchronisationsprotokolls zur Folge hat.

Vielversprechender dagegen ist der Einsatz eines *Multiversion-Konzeptes* (etwa in Kombination mit einem Sperrverfahren), bei dem Lese-TA den bei ihrem TA-Start gültigen DB-Zustand zugänglich gemacht bekommen (durch Führen von Objektversionen), so daß Lese-TA bei der Synchronisation nicht mehr zu berücksichtigen sind. Die damit erreichbare Reduzierung der Synchronisationskonflikte erfordert jedoch die Verwaltung eines sogenannten Versionenpools, der zur Begrenzung des E/A-Aufwandes (weitgehend) im Hauptspeicher gehalten werden sollte. Wenn Konsistenzebene 2 ausreichend ist, lassen sich Synchronisationskonflikte mit Lese-TA auch ohne den Aufwand eines Multiversion-Konzeptes vermeiden, falls die Änderungen in einer privaten Objektkopie vorgenommen werden. Lese-TA sehen dann immer die aktuellste Objektversion, jedoch u.U. verschiedene Versionen desselben Objektes [Ra87b].

Besonders problematisch ist die Synchronisation auf *Hot Spot- oder High Traffic-Datenelementen*, die daher auch möglichst bereits beim DB-Entwurf zu vermeiden sind [Gr85]. Ist dies nicht möglich, so sind zur Reduzierung der Synchronisationskonflikte ggf. spezielle Synchronisationsprotokolle vorzusehen, wie sie etwa in IMS Fast Path verwendet werden [Ga85]. Diese Verfahren bewirken jedoch, ebenso wie auch andere Vorschläge [Re82, ON86], bei denen zur Erhöhung der Parallelität die Semantik höherer Operationen ausgenutzt werden soll, zum Teil Erweiterungen an der Programmierschnittstelle, und sie können nur begrenzt eingesetzt werden. Die starken Behinderungen, die von langen (Änderungs-)TA verursacht werden *(Batch-Verarbeitung)*, sollten auch möglichst durch organisatorische Maßnahmen umgangen werden, indem solche TA entweder in Schwachlastzeiten laufen (z.B. nachts) oder die TA in eine Reihe sogenannter ‚mini-batches' zerlegt werden. Eine Alternative ist auch hier die Anwendung spezieller Synchronisationstechniken [Ba86].

Reduzierung des E/A-Aufwandes

Voraussetzung für eine geringe E/A-Häufigkeit ist eine auf die Anwendung zugeschnittene Wahl von Speicherungsstrukturen und Zugriffspfaden. So erlauben Hash-Verfahren und B*-Bäume schnellen Schlüsselzugriff auf Datensätze; *Clusterbildung* ermöglicht die effiziente Verarbeitung zusammengehöriger Daten.

Lokalität im Referenzverhalten von TA wird von der *Systempufferverwaltung* zur Minimierung physischer E/A-Vorgänge genutzt, wobei immer größer werdende Systempuffer (mit main memory databases als Grenzfall) weitere E/A-Einsparungen ermöglichen. Einen wichtigen Einfluß auf das gesamte E/A-Verhalten übt die Vorgehensweise beim Ausschreiben geänderter Datenbankseiten aus. Die sogenannte *NOFORCE-Strategie* [HR83], bei der Änderungen bei EOT nur gesichert, die geänderten Seiten jedoch nicht in die physische Datenbank ausgeschrieben werden, hilft im allgemeinen, E/A einzusparen. Denn damit kann eine Seite mehrfach geändert werden, ohne ausgeschrieben zu werden; außerdem werden die Antwortzeiten von Änderungs-TA nicht unnötig erhöht. Weitere E/A-Einsparungen verspricht man sich von einer erweiterten Speicherhierarchie, insbesondere durch seitenadressierbare Halbleiterspeicher (‚expanded storage'), die Zugriffszeiten um 1 ms erlauben (siehe Kap. 5). 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.

Für Hochleistungs-Datenbanksysteme müssen vor allem die Ausgaben der Log-Daten optimiert werden, wobei Duplex-Logging aus Verfügbarkeitsgründen als obligatorisch angesehen werden muß. Das *Logging* sollte v.a. mittels sequentieller E/A (bzw. chained I/O) anstelle von wahlfreier E/A erfolgen, da jede wahlfreie E/A die Antwortzeit einer TA um etwa 25–40 ms verlängert (ohne CPU-bezogene Kosten für E/A-Vorbereitung, Prozeßwechsel usw.). Unterstellt man eine sequentielle Log-Datei und eine Log-E/A pro TA, so würde dadurch die Leistung des Systems auf etwa 60 Ände-

rungs-TA pro Sekunde begrenzt, da üblicherweise zwischen zwei E/A's eine volle Plattenumdrehung (15–20 ms) liegt. Höhere TA-Raten erfordern daher ein sogenanntes *Gruppen-Commit*, d.h. die Verzögerung des EOT einer Gruppe von TA, bis die zugehörigen Log-Daten nach synchronem Schreibaufruf die Log-Datei sicher erreicht haben. Für einen effektiven Einsatz des Gruppen-Commits sowie zur Begrenzung des Log-Umfangs ist eintragsweises Logging vorzusehen; damit benötigt man dann etwa 0.1–0.2 Log-E/A pro TA [Gr85].

Andere Optimierungsmöglichkeiten ergeben sich durch den Einsatz von nichtflüchtigen Halbleiterspeichern (solid state disks). Da bei diesen ‚Platten' Positionier- und Umdrehungswartezeiten wegfallen, ist sequentielle und wahlfreie E/A gleich schnell, und der Unterschied zu chained I/O verschwindet. Ihre Verwendung für Logging bietet ein Optimierungspotential von etwa einem Faktor 10 verglichen mit sequentieller E/A auf herkömmlichen Platten. Solche ‚elektronischen Platten' besitzen gegenwärtig eine Speicherkapazität bis zu 500 MB und können über 4 Kanäle angesteuert werden. Bei Nutzung eines Kanals (3–4.5 MB/sec) lassen sich 400–600 Seiten zu 4 KB pro Sekunde sequentiell in eine Datei ausschreiben [Ga87]. Solche Platten sind zur Zeit noch erheblich teurer als herkömmliche Platten (bis zu einem Faktor von 50), was lediglich ihren selektiven Einsatz für zeitkritische Funktionen gestattet.

Die als Vorsorgemaßnahmen für den Fehlerfall anfallende E/A ist in Hochleistungs-DBS wegen der hohen Transaktionsraten und der kurzen Antwortzeiten als potentieller Engpaß ersten Grades anzusehen. Deshalb scheiden alle Lösungen auf der Basis von Seitenlogging aus. Zur Eingrenzung des E/A-Aufwandes erscheinen von den vorgenannten Punkten vor allem eine NOFORCE-Strategie bei der Systempufferverwaltung sowie Eintragslogging als unabdingbar. Ein solcher Implementierungsansatz impliziert ein Sicherungspunktverfahren als unterstützende Maßnahme zur Beschleunigung der Crash-Recovery. Da bei sehr großen Puffern (> 50 MB) eine direkte Erstellung von Sicherungspunkten den Systembetrieb sehr belasten würde, sollten Verfahren eingesetzt werden, die lange Totzeiten zum Ausschreiben der geänderten Pufferinhalte vermeiden. Dazu bieten sich sogenannte ‚fuzzy checkpoints' an [HR83].

Recovery

Hier ist v.a. eine schnelle TA-Recovery (Rücksetzen einer TA) wichtig, da sie wesentlich häufiger vorkommt als etwa eine Crash- oder Plattenrecovery. Das UNDO einer TA ist besonders einfach, wenn alle Änderungen in einer privaten Kopie vorgenommen werden und diese Kopien nicht in die Datenbank verdrängt werden; in diesem Fall brauchen lediglich die Kopien der gescheiterten TA weggeworfen zu werden (keine E/A). Für eine schnelle Crash-Recovery muß vor allem der Ausfall des Rechners frühzeitig erkannt werden, um zu verhindern, daß das gesamte Terminalnetz neu aktiviert werden muß, da dies bei mehreren tausend Terminals leicht eine Stunde oder mehr dauern kann. Eine unterbrechungsfreie Plattenrecovery wird durch Einsatz von Spiegelplatten ermöglicht.

Kommunikationsaufwand

Ein globales Optimierungsziel in Mehrrechner-DBS ist es, den Kommunikationsoverhead möglichst gering zu halten bzw. die Anzahl von die Antwortzeit verschlechternden (synchronen) Nachrichten zu minimieren. Wichtig dafür sind ein schnelles Kommunikationssystem und effiziente Kommunikationsprimitive; weitere Einsparungen werden durch Bündelung und gemeinsame Übertragung von Nachrichten möglich (group request). Das tatsächliche Nachrichtenaufkommen für eine bestimmte TA-Last ist jedoch von der jeweiligen Architektur (DB-Sharing oder DB-Distribution) sowie den verwendeten Algorithmen abhängig. Darüber hinaus ergeben sich in Mehrrechner-DBS weitere leistungsbestimmende Merkmale, die nicht unmittelbar mit dem Kommunikationsaufwand zusammenhängen (z.B. Möglichkeiten der Lastbalancierung).

3 DB-Sharing vs. DB-Distribution: allgemeine Aspekte

Die Erfüllung der zentralen Anforderungen an Hochleistungs-DBS – hohe TA-Raten (z.B. 1000 TA/sec), hohe Verfügbarkeit und modulare Erweiterungsfähigkeit – bedingen den Einsatz von Mehrrechner-DBS. Auf der Basis allgemeiner Rechner kommen dabei neben eng gekoppelten Multiprozessoren (‚shared memory') im wesentlichen zwei generelle Klassen von Mehrrechner-DBS, nämlich DB-Sharing (‚shared disk') und DB-Distribution (‚shared nothing'), in Betracht [HR86]. Eng gekoppelte Mehrrechner-DBS erlauben zwar die einfachste Realisierungsform, eine effiziente Kommunikation und Kooperation zwischen den Rechnern (-> kurze Antwortzeiten) und bieten ein ‚single system image' nicht nur für Endbenutzer und Anwendungsprogram-

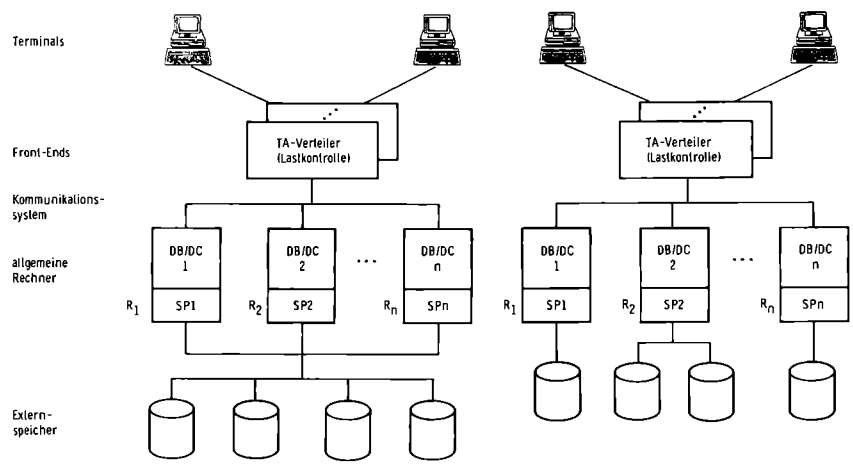

Bild 1. Grobarchitektur von DB-Sharing- und DB-Distribution-Systemen.

mierer, sondern auch für Systemprogrammierer und Operateure, dennoch können sie die Anforderungen an Hochleistungs-DBS v.a. hinsichtlich Verfügbarkeit und Erweiterbarkeit aber auch bezüglich des Durchsatzes i.a. nicht erfüllen [HR86]. Daher konzentrieren wir uns hier vorwiegend auf DB-Sharing- und DB-Distribution-Systeme, deren Grobarchitektur in Bild 1 skizziert ist.

DB-Sharing und DB-Distribution-Systeme bestehen aus einer Menge von autonomen Rechnern, die lose oder nah gekoppelt sind. Jeder der Rechner besitzt einen eigenen Hauptspeicher sowie eine separate Kopie von Betriebssystem und Datenbankverwaltungssystem (DBVS). Weiterhin soll in jedem Rechner ein eigener Systempuffer (SP) sowie eine eigene DC-Komponente vorliegen. In lose gekoppelten Systemen erfolgt die Kommunikation zwischen den Rechnern ausschließlich über Nachrichten, während bei einer nahen Kopplung eine effizientere Zusammenarbeit z.B. über gemeinsam benutzte Halbleiterspeicher angestrebt wird. Um die Diskussion überschaubar zu halten, werden zunächst nur lose gekoppelte DB-Sharing und DB-Distribution-Systeme betrachtet; auf Optimierungsmöglichkeiten über eine nahe Rechnerkopplung wird dann in Kapitel 5 eingegangen.

Der offensichtliche Unterschied zwischen DB-Sharing und DB-Distribution besteht 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. Die physische Anbindung an die Platten erfordert, daß alle Prozessoren lokal angeordnet sind (z.B. in einem Raum).
- 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. Dieser Ansatz wird bei vielen verteilten DBS verfolgt, eignet sich jedoch auch bei lokaler Anordnung der Rechner.

Im weiteren Verlauf gehen wir, wenn nichts anderes gesagt wird, von einer räumlich benachbarten Anordnung der Rechner aus, weil nur so die für hohe TA-Raten und kurze Antwortzeiten erforderliche schnelle Rechnerkopplung (z.B. 10–100 MB/sec) möglich wird. Die Terminal-Anbindung kann dann, wie in Bild 1 angedeutet, über mehrere Front-Ends geschehen, die gewisse Aufgaben des DC-Systems wie etwa Verteilung der TA und die 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.

Im Rest dieser Arbeit wird versucht, einen umfassenden Vergleich zwischen DB-Sharing und DB-Distribution durchzuführen und eine Bewertung vorzunehmen hinsichtlich der Tauglichkeit zur Realisierung eines Hochleistungs-DBS bzw. bezüglich des damit verbundenen Aufwandes. Die Analyse stellt eine Weiterführung der Untersuchungen in [Hä85] und [HR86] dar. Andere vergleichende Betrachtungen zwischen DB-Sharing und DB-Distribution wurden schon in [Tr83, Sh86, St86, Re86b, CDY86] vorgenommen, jedoch wurden in diesen Arbeiten nur eine Teilmenge unserer Vergleichskriterien zur Bewertung herangezogen, wobei zum Teil wichtige Gesichtspunkte unberücksichtigt blieben. In [CDY86] wurde sogar versucht, einen quantitativen Vergleich vorzunehmen, jedoch wurden dabei zu viele Vereinfachungen vorgenommen. Es folgt nun zunächst eine Gegenüberstellung der beiden Architekturklassen hinsichtlich allgemeiner Aspekte wie Ermöglichung hoher Verfügbarkeit und Erweiterbarkeit, Handhabbarkeit, TA-Bearbeitung, DB-Entwurf, Ortsverteilung sowie existierenden Lösungen. In Kapitel 4 werden dann die Systemkomponenten untersucht, für die im Vergleich zu zentralisierten DBS neue Techniken benötigt werden und deren Realisierung die Leistungsfähigkeit des Systems entscheidend beeinflußt. Nach der Darstellung von Optimierungsmöglichkeiten über eine nahe Rechnerkopplung folgt dann in Kap. 6 eine zusammenfassende Bewertung.

Verfügbarkeit

Für beide Architekturen kann von einer ausreichenden Redundanz zur Tolerierung von Hardware- und Softwarefehlern ausgegangen werden; mit der losen Rechnerkopplung ist eine vergleichbare Fehlerisolation zu erreichen. Nach Ausfall eines Rechners können bei DB-Sharing die überlebenden Rechner 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

Idealerweise erlaubt die Erweiterung des Systems um einen Rechner ein lineares Durchsatzwachstum mit gleichbleibenden Antwortzeiten (modulares Wachstum). Dies ist jedoch i.a. nur schwer zu erreichen, da die TA im neuen Rechner meist zusätzliche Synchronisationskonflikte und Kommunikationsoperationen verursachen. Die daraus resultierenden Leistungseinbußen können dann möglicherweise begrenzt werden, wenn neben dem zusätzlichen Rechner auch neue TA-Typen oder DB-Bereiche hinzukommen. Im Gegensatz zu DB-Sharing erzwingt die Hinzunahme eines Rechners bei DB-Distribution die Neuaufteilung der DB ($n -> n+1$), 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 der Daten sogar Programmänderungen zur Folge haben.

Bei DB-Sharing dagegen wird ein neuer Rechner an alle Platten angeschlossen; es ist keine Datenverteilung vorzunehmen. Eine Einschränkung besteht hier jedoch i.a. hinsichtlich der Anzahl der Rechner, die an eine Platte angeschlossen werden können (Multiports). Bei DB-Distribution läßt sich allerdings wegen der Notwendigkeit der Datenverteilung auch oft keine beliebige Anzahl von Rechnern einsetzen. Zudem haben analytische Untersuchungen gezeigt [DIY86], daß es unter Leistungsgesichtspunkten ratsamer ist, eine kleinere Anzahl ($\leq$ 10) leistungsfähiger Rechner zu koppeln als eine größere Anzahl kleinerer Rechner. Denn bei langsameren Rechnern verlängert sich die Antwortzeit der TA und damit die Anzahl der Synchronisationskonflikte (Sperren werden länger gehalten u.ä.). Bis zu 10 Rechner mit je 10–30 MIPS dürften zudem für praktische Systeme in der nächsten Zukunft keine Leistungseinschränkung bedeuten.

Das Hinzufügen neuer Platten/Daten ist bei DB-Sharing auch wesentlich einfacher möglich als bei DB-Distribution. Es ergeben sich insbesondere keine Auswirkungen für die Anwendungen und es braucht keine Entscheidung darüber getroffen zu werden, welchem Rechner die neuen Platten/Daten zuzuordnen sind.

Handhabbarkeit

Beide Ansätze verkörpern zunächst eine Menge homogener, aber separater Systeme, da separate Kopien von Betriebssystem und DB/DC-Systemen eingesetzt werden. Die erste Forderung betrifft die Bereitstellung eines ‚single system image' zumindest aus der Sicht des Anwendungsprogrammierers und Endbenutzers zur Maskierung von Fehlern, Isolation von Konfigurationsänderungen und besonders bei DB-Distribution zur Gewährleistung von Verteilungstransparenz der Daten. Noch wichtiger für die Verfügbarkeit und Zuverlässigkeit dürfte diese Eigenschaft aus der Sicht der Administration und des Betriebs sein. Mehrere Operateur- und Masterkonsolen implizieren eine schwierigere Handhabung und Verwaltung; schwerwiegender jedoch dürfte sein, daß sie gerade im Fehlerfall zur Verwirrung und zur Fehlbedienung beitragen. Die Untersuchungen in [Gr86] zeigen, daß in solchen Bedienfehlern eine Hauptursache für Systemausfälle liegt. Bei DB-Distribution ist die Bestimmung und Änderung der Datenverteilung trotz Verwendung von Entwurfswerkzeugen und Tuninghilfen i.a. nicht vollständig automatisierbar, sondern bedarf der Mitwirkung des Systemverwalters und des Datenbankadministrators. Die Erreichung eines ‚single system image' dürfte daher gegenüber diesen Personengruppen kaum realisierbar sein. Die ortsverteilte Anordnung der Rechner erlaubt zwar eine Abstimmung auf lokale Bedürfnisse (z.B. innerhalb einer großen Organisation), erschwert jedoch die Verwaltung des Gesamtsystems und erfordert in der Regel qualifiziertes Bedienpersonal an jedem Knoten.

TA-Bearbeitung

Bei DB-Sharing kann eine TA vollständig in einem Rechner bearbeitet werden, da jeder Rechner Zugriff zu allen Daten hat. Für DB-Distribution ist dies nur möglich, wenn die TA ausschließlich lokale Daten benötigt. Daten fremder Partitionen können durch Daten- oder Funktionsaustausch besorgt werden (I/O-request shipping vs. function-request/DB-call shipping [YCDT86]). Die übliche Vorgehensweise ist hierbei das Verschicken von DML-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.

DB-Distribution läßt sich für mengenorientierte DB-Operationen, die Daten von verschiedenen Rechnern betreffen, zur parallelen Bearbeitung einer TA auf mehreren Rechner nutzen. Für DB-Sharing wäre eine solche Parallelität innerhalb einer TA (mit Sub-TA's) prinzipiell auch möglich, wobei man noch mehr Freiheitsgrade bei der Auswahl der Rechner hätte. Wegen der Kommunikationskosten zum Start der Sub-TA, dem Zurückgehen der Ergebnisse (bei einer Anfrage) und dem notwendig werdenden Mehr-Phasen-Commit dürfte sich dies jedoch nur in Einzelfällen lohnen; es wird daher für DB-Sharing nicht weiter betrachtet.

Datenbankentwurf

Für DB-Sharing kann der DB-Entwurf wie für zentrale DBS vorgenommen werden, da keine Partitionierung der Datenbank erforderlich ist. Insbesondere können existierende Datenbanken und Anwendungsprogramme beim Übergang von einem zentralisierten System zu DB-Sharing ohne Änderung verwendet werden.

Bei DB-Distribution dagegen ist im physischen DB-Entwurf eine Zuordnung von Datenpartitionen zu Rechnern zu finden, so daß dann in Zusammenarbeit mit der Lastkontrolle eine möglichst effiziente TA-Bearbeitung ermöglicht wird. Da die Partitionierung eine relativ statische Entscheidung ist, die nur sehr umständlich geändert werden kann, ist sie besonders sorgfältig zu treffen. Die Erstellung einer effektiven Datenverteilung muß daher zwangsläufig einen Kompromiß für die zu bearbeitenden TA-Lasten, deren Referenzverhalten bekannt sein muß, darstellen. Stärkere Lastschwankungen, wie sie mehrmals täglich vorkommen können, führen daher unweigerlich zu Leistungseinbußen, während DB-Sharing eine flexible Anpassung durch die Lastkontrolle erlaubt (s.u.). Eine Änderung der Datenverteilung wird bei DB-Distribution i.a. notwendig beim Hinzukommen neuer Anwendungen und TA-Typen, ebenso beim Hinzufügen neuer Rechner und Daten/Platten (s.o.). Vergleichsweise einfach kann dagegen der Ausfall eines Rechners (Übernehmen der Partition

durch einen vorbestimmten Rechner) sowie seine Reintegration (Rückgängigmachen der Umverteilung) behandelt werden, wenn auch nicht so leicht wie bei DB-Sharing.

Wie bereits erwähnt, erlauben hierarchische und netzwerkartige Datenmodelle keine ausreichende Flexibilität bezüglich der Datenverteilung, um Minimalanforderungen an Wachstum, Rekonfiguration und bei Lastprofil-Änderungen erfüllen zu können. Für diese Datenmodelle kann daher in vielen Einsatzfällen nur mit DB-Sharing den Anforderungen an Hochleistungs-DBS entsprochen werden.

Ortsverteilung

In großen Unternehmen führen organisatorische Bedürfnisse (Systemleistung vor Ort), Kommunikationskosten usw. vielfach zur Forderung eines geographisch verteilten Systems. Ortsverteilung ist zudem Voraussetzung für eine schnelle Katastrophenrecovery (s. u.). DB-Distribution besitzt alle Konzepte und Prinzipien für den ortsverteilten Einsatz. DB-Sharing dagegen ist wegen der Externspeicheranbindung (Kanalkopplung) auf den lokalen Einsatz beschränkt. Eine Ortsverteilung wäre daher nur durch ein geographisch verteiltes Netz von DB-Sharing-Clustern denkbar. Dies wäre jedoch nur sinnvoll, wenn eine strikte Partitionierung der Last möglich wäre, da sonst Algorithmen für den lokalen als auch den (geographisch) verteilten Fall vorzusehen wären. Eine solche Mischlösung würde jedoch zu einer hohen Systemkomplexität und den damit verbundenen Nachteilen führen.

Für ein ortsverteiltes DB-Distribution-System stellt sich allerdings auch die Realisierung einzelner Systemkomponenten anders dar als im bisher unterstellten lokalen Fall. So ist z. B. die kurzfristige Übernahme der DB-Partition eines ausgefallenen Rechners nur noch möglich, wenn jeder Knoten mindestens zwei autonome Rechner umfaßt. Ebenso reduzieren sich die Möglichkeiten einer Lastkontrolle im ortsverteilten Fall, da in der Regel die Auslegung des Terminalnetzes festschreibt, an welchem Knoten eine TA zur Ausführung kommt. Eine Lastbalancierung ist weiterhin praktisch nur noch innerhalb eines Knotens möglich.

Aufgrund der langsamen Kommunikationsverbindungen dürften kurze Antwortzeiten und hoher Durchsatz nur erreichbar sein, wenn die weitaus meisten (> 95%) der benötigten Daten lokal vorliegen. Eine Replikation der Daten bringt unter Leistungsgesichtspunkten nur Vorteile, wenn eine sehr geringe Änderungsintensität gegeben ist.

Verfügbare Lösungen

DB-Sharing ist ein relativ neuer Ansatz. Eine erste Realisierung stellt das seit einigen Jahren verfügbare „Data Sharing" für IMS dar [Ke82], ein DB-Sharing-System für maximal zwei Rechner, das jedoch keinen Anspruch auf Hochleistungseigenschaften erheben kann. Das DB-Sharing-System von Computer Console [We83] erlaubt zwar bis zu acht (kleinere) Rechner, es lassen sich damit aber auch keine sehr hohen TA-Raten erzielen (z. B. ‚nur' 150 TA/sec). Die Prototyp-Erstellung leistungsfähigerer DB-Sharing-Systeme werden im DCS-Projekt [Se84] und im Amoeba-Projekt [Tr83, Sh85] angestrebt. Die DEC VAX-Cluster [KLS86], von denen 1985 bereits rund 2000 Installationen existierten, ähneln auch der DB-Sharing-Architektur, da jeder Rechner auf alle Platten zugreifen kann. Obwohl es sich dabei um kein Datenbanksystem handelt, werden einige DB-Sharing-relevante Probleme gelöst (z. B. flexible Anbindung der Platten an die Rechner, Synchronisation der Datenzugriffe auf Blockebene). Neben diesen Systemimplementierungen existieren für einzelne Komponenten (z. B. Synchronisation, Lastkontrolle) eines DB-Sharing-Systems bereits eine Reihe von Forschungsarbeiten sowie erste quantitative Analysen mit Simulationen und analytischen Modellen.

Dagegen kann für DB-Distribution auf eine Vielzahl von Lösungsvorschlägen und Implementierungen im Bereich der verteilten DBS zurückgegriffen werden, da dieses Gebiet seit über 10 Jahren intensiv bearbeitet wird. Mittlerweile verfügen die meisten DBS-Hersteller zumindest über ein einfaches verteiltes DBS, so daß die Realisierung eines leistungsfähigen DB-Distribution-Systems u. U. auf vorhandener Software aufbauen kann. Als Beispiel hierfür steht TANDEM, das sein ENCOMPASS-System verstärkt in Richtung „Hochleistungseigenschaft" entwickelt [He85].

4 Realisierung einzelner Systemkomponenten bei DB-Sharing und DB-Distribution

Die Beurteilung der beiden Mehrrechnerarchitekturen DB-Sharing und DB-Distribution erfordert eine Verfeinerung der allgemeinen Betrachtungen. Dazu werden Probleme und Konzepte für die wichtigsten Systemkomponenten, die in Eigenschaften, Realisierung und Aufwand Unterschiede aufweisen, genauer diskutiert. Dies betrifft vor allem Systempufferverwaltung, Synchronisation, Lastkontrolle, sowie Logging und Recovery. Bei der Entwicklung neuer Algorithmen und Techniken muß die starke gegenseitige Beeinflussung und Abhängigkeit von DBS-Funktionen (z. B. Synchronisation und Recovery) ebenso berücksichtigt werden wie ihre Tauglichkeit bezüglich der Hauptanforderungen an Hochleistungs-DBS.

Systempufferverwaltung

Das Hauptproblem bei DB-Sharing besteht in der Lösung des sogenannten *Veralterungsproblems*, das aus der Existenz eines Systempuffers in jedem Rechner resultiert (Pufferinvalidierung). Nach Änderung einer Seitenkopie in einem der Systempuffer sind eventuell vorhandene Kopien in fremden Puffern und ihr Abbild auf dem Externspeicher veraltet. Es ist also der Zugriff auf veraltete Seiten zu verhindern, und es muß ggf. die neueste Version eines Objektes zugänglich gemacht werden. Dazu kann ein direktes Verschicken geänderter

Objekte oder ihr Austausch über Platte herangezogen werden.

Der Zugriff auf veraltete Objekte kann sowohl durch allgemeine Verfahren als auch durch Erweiterung der Synchronisationskomponente verhindert werden. Eine allgemeine Lösung besteht z. B. darin, jede Seitenänderung allen Rechnern bei TA-Ende mitzuteilen, woraufhin veraltete Kopien aus den Puffern entfernt werden können. Weniger Nachrichtenaufwand ist allerdings erforderlich, wenn das Veralterungsproblem zusammen mit der Synchronisation behandelt wird [Ra86c].

Falls bei DB-Distribution keine Daten- sondern Funktionsanforderungen verschickt werden, ergibt sich eine Pufferverwaltung wie in zentralen DBS. Es entsteht kein Veralterungsproblem, da jeder Rechner nur Daten seiner Partition im Systempuffer hält.

Synchronisation

Bei DB-Sharing besitzt die Realisierung der Synchronisationskomponente leistungsbestimmenden Einfluß. Das primäre Ziel bei der Entwicklung eines geeigneten Synchronisationsalgorithmus muß die Minimierung der externen Synchronisationsnachrichten sein, da sonst der Durchsatz und vor allem die Antwortzeiten stark beeinträchtigt werden können. Weiterhin ist aus Effizienzgründen eine integrierte Lösung für das Veralterungsproblem anzustreben, und das Verfahren muß robust gegenüber Rechnerausfällen sein. Einen Überblick über mögliche Synchronisationsprotokolle für DB-Sharing findet man in [RS84, Ra86b, Ra87b]. Als geeignet sind dabei vor allem Sperrverfahren und optimistische Verfahren anzusehen, die jeweils zentral auf einem Rechner oder unter verteilter Kontrolle ablaufen können. Das Primary Copy-Sperrverfahren [Ra86c, Ra87a] – bei dem eine Verteilung der Synchronisationsverantwortlichkeit für einzelne Partitionen auf die einzelnen Rechner vorgenommen wird – scheint ein erfolgversprechender Ansatz zu sein, wobei dynamisch änderbare Partitionen (durch interne Kontrollstrukturen repräsentiert) vorausgesetzt werden. In Zusammenarbeit mit der Lastkontrolle kann dabei das Ausmaß der Synchronisationsnachrichten reduziert sowie eine Anpassung an unterschiedliche Lastsituationen und Rechnerkonfigurationen vorgenommen werden.

Bei DB-Distribution verwaltet jeder Rechner die Daten seiner Partition. Daher kann auch die Synchronisation weitgehend wie in zentralen DBVS erfolgen, indem jeder Rechner die Zugriffe zu seiner lokalen Partition synchronisiert. Wenn Sperrverfahren eingesetzt werden, dann muß zusätzlich noch eine globale Deadlockbehandlung durchgeführt werden, wenn Deadlocks nicht vermieden (z. B. mit Zeitmarken an Transaktionen) werden können. Weitere neue Gesichtspunkte entstehen durch die Einbeziehung des Mehr-Phasen-Commit-Protokolls sowie bei optimistischen Verfahren durch die systemweite Validierung und das Einbringen von Änderungen. Zur Lösung dieser Probleme kann auf eine Vielzahl von Lösungen für verteilte DBS (siehe z. B. [Da81, BG81, Ra87b]) zurückgegriffen werden.

Ein Nachteil von DB-Sharing im Vergleich zu DB-Distribution ist, daß eine Synchronisation auf Satzebene nur äußerst schwierig zu realisieren ist. Denn das parallele Ändern einer Seite in verschiedenen Rechnern erfordert ein korrektes Mischen der Änderungen, um eine konsistente Kopie der Seite zu erhalten, was jedoch im Einzelfall sehr problematisch werden kann. Am ehesten ist das Mischen der Änderungen noch bei Seitentypen mit homogener und stabiler Binnenstruktur (z. B. bei Adreßumsetztabellen oder Seiten mit Freispeicherinformationen) möglich. Bei allgemeinen Seitentypen jedoch müssen bei DB-Sharing zumindest Änderungszugriffe i. a. auf Seitenebene synchronisiert werden, wodurch natürlich mehr Synchronisationskonflikte als bei satzorientierter Synchronisation entstehen.

Lastkontrolle

Die Realisierung einer dynamischen Lastkontrolle ist entscheidend für die Leistungsfähigkeit des Gesamtsystems. Ihre zentralen Aufgaben sind 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, in dem eine weitgehend lokale Synchronisation möglich ist. Für DB-Distribution ist der Rechner zu suchen, an dem die meisten der benötigten Daten direkt zugreifbar sind.

Damit zu einer TA der geeignete Rechner bestimmt werden kann, sind Informationen über das wahrscheinliche Referenzverhalten der TA notwendig (z. B. aus TA-Typ oder aktuellen Parametern ableitbar). Da jedoch pro TA nur eine begrenzte Anzahl von Instruktionen zur Bestimmung eines geeigneten Rechners möglich sind, sollte – eine zumindest über längere Zeit (Minuten, Stunden) – festgelegte Zuordnung vorgesehen werden [Re86a]. Diese kann oder muß umgestellt werden, wenn sich die Last entscheidend ändert oder ein Rechner ausfällt bzw. dazukommt.

Die Zuordnung einer TA zu einem Rechner (TA-Routing) 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 oder ein Rechnerausfall weit weniger gut verkraftet werden als bei DB-Sharing. Des weiteren 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. Eine gleichmäßige Auslastung der Rechner ist somit nur möglich, wenn auf jede Datenpartition in etwa gleich viele DML-Befehle abgesetzt werden! Die Möglichkeiten der sogenannten Query-Optimierung, mit der für mengenorientierte Operationen ein möglichst ‚kostengünstiger' Ausführungsplan erstellt wird, sind zu-

dem sehr begrenzt, da in kommerziellen DB-Anwendungen typischerweise Einzelsatzzugriffe dominieren. Außerdem kann bei der Query-Optimierung die aktuelle Rechnerauslastung i.a. ohnehin nicht berücksichtigt werden, 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 Lastbalancierung. Denn jede DB-Operation und somit jede TA kann von jedem der Rechner ausgeführt werden; 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. Eine besonders effektive Zusammenarbeit zwischen Lastkontrolle und Synchronisationskomponente wird mit dem schon erwähnten Primary Copy-Sperrverfahren möglich, da hiermit vergleichsweise einfach der Rechner bestimmt werden kann, auf dem eine TA weitgehend lokal synchronisiert werden kann. Die mit einer solchen Kooperation erzielbare Lokalität innerhalb der Rechner bewirkt nicht nur die Einsparung von externen Sperranforderungen, sondern erhöht auch die Trefferraten im Systempuffer (weniger physische Lesevorgänge) und begrenzt das Ausmaß der Pufferinvalidierungen (weniger häufiges Holen geänderter Seiten aus fremden Systempuffern) [Ra87a].

Logging

Für DB-Distribution führt jeder Rechner die Log-Daten bezüglich der DB-Partition, für die er verantwortlich ist. Hierfür können weitestgehend die Verfahren aus zentralisierten DBVS verwendet werden [HR83].

Bei DB-Sharing ist neben den lokalen Log-Dateien noch eine globale Log-Datei durch Mischen der lokalen Dateien zu erstellen. In den lokalen Log-Dateien werden die Änderungen von TA des betreffenden Rechners protokolliert. Sie werden benutzt zum Rücksetzen von TA und zur Recovery nach einem Rechnerausfall. Die globale Log-Datei, die die Änderungen von allen Rechnern enthält, wird geführt, um die Plattenfehler-Recovery durchführen zu können. Das Mischen der lokalen Log-Daten kann dabei typischerweise asynchron zur normalen TA-Verarbeitung erfolgen, etwa durch einen eigenen Prozeß (Prozessor), für dessen Ausfall z.B. ein Schattenprozeß vorzusehen ist. Für hohe Raten von Änderungs-TA kann der Umfang der globalen Log-Datei zu Problemen führen.

Nachrichten-Logging ist sowohl für Eingabe- als auch Ausgabenachrichten erforderlich und wird durch die in jedem Rechner angesiedelte DC-Komponente vorgenommen. Der Nachrichten-Log kann z.B. nach einem Rechnerausfall dazu benutzt werden, in Bearbeitung befindliche TA ohne erneute Interaktion des Benutzers auf einem der überlebenden Rechner zu wiederholen. Die Konzepte hierzu dürften sich für DB-Sharing und DB-Distribution kaum unterscheiden. Zur Erzielung einer hohen TA-Rate mit kurzen TA muß die Nachrichtenverwaltung besonders effizient sein. So kann man z.B. die Antwortnachricht als Quittung für die Eingabenachricht verwenden anstatt jede Nachricht eigens zu quittieren.

Recovery

Das Zurücksetzen einer TA (partial UNDO) stellt bei DB-Sharing kein Problem dar und kann wie im zentralisierten Fall ggf. unter Verwendung der lokalen Log-Daten erfolgen. Für DB-Distribution jedoch erfordert das Zurücksetzen einer TA die Kooperation mehrerer Rechner, falls die TA nicht-lokale Daten geändert hat.

Nach Ausfall eines Rechners muß die (Crash-)Recovery für den gescheiterten Prozessor durch die überlebenden Rechner vorgenommen werden, damit gehaltene Sperren u.ä. möglichst schnell wieder allgemein verfügbar werden. Die Recovery wird dabei sowohl für DB-Sharing als auch für DB-Distribution mit der lokalen Log-Datei des ausgefallenen Rechners durchgeführt. Bei DB-Sharing ist insbesondere die REDO-Recovery für erfolgreich beendete TA komplizierter als bei DB-Distribution, da hierzu nicht einfach die After-Images von der lokalen Log-Datei in die Datenbank geschrieben werden dürfen. Denn dies ist nur dann möglich, wenn sichergestellt ist, daß nicht auf einem anderen Rechner eine aktuellere Änderung vorgenommen wurde, da sonst möglicherweise die aktuelle Objektkopie der Datenbank durch ein veraltetes After-Image überschrieben würde. Des weiteren müssen ggf. verloren gegangene Datenstrukturen rekonstruiert werden, um eine ordnungsgemäße Fortführung der Synchronisation sowie der Behandlung des Veralterungsproblems zu gewährleisten.

Nach Ausfall einer Platte (media failure) kann der aktuelle Plattenzustand aus einer Archivkopie und dem Archiv-Log rekonstruiert werden, was jedoch sehr zeitaufwendig ist und damit die Verfügbarkeit beeinträchtigt. Mit Spiegelplatten läßt sich die Ausfallwahrscheinlichkeit drastisch verbessern, so daß nur noch in extrem seltenen Fällen eine solch langsame Platten-Recovery notwendig wird.

Eine effektive *Katastrophen-Recovery* erfordert das redundante Führen aller Daten in mindestens zwei geographisch verteilten Rechenzentren. Im Normalfall wird die TA-Last unter den beiden Zentren aufgeteilt und die Änderungen werden untereinander ausgetauscht; im Katastrophenfall (Erdbeben, Explosion u.ä.) übernimmt das überlebende Zentrum die gesamte Last. Für DB-Distribution ist das Führen einer solchen ortsverteilten Redundanz vergleichsweise einfach; mögliche Verfahren zur Übermittlung von Änderungen werden in [GA85] angesprochen. Diese Art der Ortsverteilung ist für DB-Sharing prinzipiell auch realisierbar, jedoch kann der Austausch von Änderungen bzw. die Bereitstellung der aktuellsten Kopie eines Objektes nicht so natürlich in die Algorithmen integriert werden wie bei DB-Distribution.

Diskussion

Aus den vorangegangenen Ausführungen geht hervor, daß sich für DB-Sharing bei den fünf in diesem Kapitel angesprochenen Systemfunktionen eine komplexere Realisierung ergibt als bei DB-Distribution. Dazu kommt, daß sich die einzelnen Systemkomponenten wesentlich stärker gegenseitig beeinflussen als bei DB-Distribution oder zentralisierten DBS. So erfordert die Lösung des Veralterungsproblems eine verstärkte Zusammenarbeit der Systempufferverwaltung mit der Synchronisations- sowie (nach einem Rechnerausfall) mit der Recoverykomponente. Die Strategie zur Verteilung der TA-Last sollte ebenfalls mit dem Synchronisationsverfahren abgestimmt sein und muß nach Ausfall eines Rechners angepaßt werden. Schließlich ergeben sich zusätzliche Abhängigkeiten zwischen Recoverykomponente und Synchronisation, auf die hier jedoch nicht näher eingegangen werden kann.

Andererseits ist bei DB-Distribution bereits durch die (statische) Datenverteilung das Ausmaß der Interprozessor-Kommunikation sowie die Auslastung der Rechner – und damit letztlich die Leistungsfähigkeit des Systems – weitgehend festgelegt; durch eine ungeschickte Aufteilung der TA-Last können höchstens noch zusätzliche Leistungseinbußen verursacht werden. Die DB-Sharing-Architektur dagegen erlaubt nicht nur eine einfachere Adaption an eine verminderte oder erhöhte Anzahl von Rechnern sondern auch eine flexible Anpassung an Lastschwankungen, so daß diese nicht notwendigerweise eine verringerte Leistungsfähigkeit zur Folge haben. Des weiteren bestehen wesentlich mehr Möglichkeiten zur Lastbalancierung und um die Anzahl der Nachrichten zu reduzieren. Wenn diese Flexibilität durch geeignete Algorithmen und Techniken genutzt werden kann, so deutet vieles auf Leistungsvorteile für DB-Sharing hin.

5 Einsatzmöglichkeiten einer nahen Rechnerkopplung

Bei einer lokalen Anordnung der Rechner ist zur Beschleunigung von Interprozessor-Kommunikationen für DB-Sharing und DB-Distribution eine nahe Kopplung der Rechner möglich. Ziel einer nahen Kopplung ist die Gewährleistung höchster Leistungsfähigkeit bei ausreichender Verfügbarkeit und Erweiterbarkeit. Am ehesten dürfte dabei eine nahe Kopplung über gemeinsame (d. h. von allen Rechnern erreichbare) Halbleiterspeichersegmente realisierbar sein. Der Einsatz solcher gemeinsamen Speicherpartitionen sollte jedoch auf die Realisierung weniger Funktionen beschränkt werden, um einen Engpaß zu verhindern. Man muß sich weiterhin darüber im klaren sein, daß ein gemeinsamer Speicherbereich zur Ausbreitung von Fehlern führen kann und daß einem Ausfall geeignet vorgebeugt werden muß (z. B. durch Ersatz-Speichersegmente).

Die Nutzung eines gemeinsamen Halbleiterspeichers ist im wesentlichen durch die Speicher-Charakteristika sowie die vorgesehene Verwendung im System bestimmt. Dabei können bezüglich der Eigenschaften drei Arten von Halbleiterspeichern in Betracht gezogen werden [Ra86a]:

1. Der Halbleiterspeicher ist *flüchtig und* für alle Rechner *instruktionsadressierbar*. Dies bedeutet, daß das gemeinsame Speichersegment für jeden Rechner als Teil seines Hauptspeichers angesehen werden kann.
2. Der Halbleiterspeicher ist *flüchtig und* für alle Rechner *seitenadressierbar*.
3. Der Halbleiterspeicher ist *nichtflüchtig und seitenadressierbar*.

Mögliche Einsatzgebiete solcher gemeinsamen Halbleiterspeicher liegen für DB-Sharing in der Unterstützung der Synchronisation, des globalen Logging und der Lastkontrolle sowie in der Verwendung als globaler Systempuffer. Für DB-Distribution ist v. a. eine Unterstützung bei der TA-Bearbeitung, der Synchronisation und ebenfalls bei der Lastkontrolle möglich. Allerdings ist bei DB-Distribution eine Optimierungsmöglichkeit über gemeinsam benutzte Speicherbereiche von vorneherein weniger interessant, da sie einem Hauptvorteil dieses Architekturtyps – der Möglichkeit einer Ortsverteilung – entgegenläuft. In einem ortsverteilten System könnte die nahe Kopplung nämlich nur innerhalb der einzelnen Knoten genutzt werden; eine solche Mischlösung führt jedoch wieder zusätzlichen Aufwand und Komplexität ein.

Die angesprochenen Einsatzformen einer nahen Kopplung werden nun genauer besprochen. Die für DB-Sharing aufgeführten Verwendungsformen wurden zum Teil schon in [Ra86a] diskutiert.

Synchronisation

Für DB-Sharing kann die Synchronisation und die Behandlung von Deadlocks quasi wie in einem zentralisierten DBVS erfolgen, wenn eine *globale Sperrtabelle* in einem gemeinsamen Halbleiterspeicher geführt wird. In jedem Rechner arbeitet ein Sperrverwalter mit der globalen Sperrtabelle, wobei z. B. über Semaphore die Zugriffe auf diese gemeinsame Datenstruktur geregelt werden. Diese Vorgehensweise setzt Instruktionsadressierbarkeit auf dem gemeinsamen Halbleiterspeicher voraus (Speichertyp 1), da sonst Seiten in den lokalen Hauptspeicher (und bei Änderungen wieder zurück) übertragen werden müßten. Zur Reduzierung von Zugriffskonflikten auf die Sperrtabelle (Engpaßgefahr) muß diese ggf. ausreichend partitioniert werden. Da ein Ausfall der Sperrtabelle Systemstillstand bedeuten würde, ist eine Kopie der globalen Sperrtabelle in einem unabhängigen Speichersegment zu führen, die jederzeit auf dem aktuellsten Stand ist.

Eine Alternative zur Verwendung einer globalen Sperrtabelle wäre die hardwaremäßige Synchronisierung der DB-Zugriffe über eine *Lock Box* (durch einen Spezialprozessor realisiert). Dieser Ansatz brächte jedoch nur Vorteile, wenn eine Lock Box-Interaktion mit der Geschwindigkeit eines Hauptspeicher-Zugriffs möglich

wäre und die Hardware intelligent genug wäre, variable Datenstrukturen zu verwalten und eine Deadlockerkennung vorzunehmen. Zudem sind für den Ausfall der Lock Box besondere Ausfallvorkehrungen erforderlich.

Für DB-Distribution ist eine Synchronisation über ein gemeinsames Speichersegment wenig sinnvoll, da jeder Rechner die Zugriffe auf seine Partition ohnehin fast wie in zentralen DBS synchronisieren kann. Lediglich die Abwicklung des Mehr(Zwei)-Phasen-Commit-Protokolls sowie die Erkennung globaler Deadlocks könnte durch einen gemeinsamen, instruktionsadressierbaren Halbleiterspeicher unterstützt werden.

Globaler Systempuffer

Der Einsatz eines gemeinsamen Halbleiterspeichers als globaler Systempuffer ist i.a. nur für DB-Sharing von Interesse. Für DB-Distribution bringt ein globaler Puffer keine Vorteile, wenn der Zugriff auf nicht-lokale Daten über das Verschicken von DML-Befehlen abgewickelt wird. Denn dann braucht jeder Rechner nur auf lokale Daten zuzugreifen, so daß die Verwendung lokaler Systempuffer ausreicht.

In einem DB-Sharing-System wird das Veralterungsproblem vermieden, wenn man nur einen globalen Systempuffer, jedoch keine lokalen Puffer mehr vorsieht. In diesem Fall ist für einen ausreichend schnellen Zugriff Instruktionsadressierbarkeit für den globalen Puffer erforderlich. Wegen der Zugriffshäufigkeit zum Systempuffer erhält man jedoch einen potentiellen Engpaß. Da ein Führen einer Kopie für den Systempuffer, der sehr groß sein müßte (z.B. > 100 MB), kaum realistisch erscheint, ist dieser Ansatz schon aus Verfügbarkeitsgründen abzulehnen.

Sinnvoller ist der Einsatz eines globalen Systempuffers unter Beibehaltung der lokalen Puffer, wobei nun der gemeinsame Halbleiterspeicher seitenadressierbar sein kann (Speichertyp 2 oder 3). Die damit vorgenommene Erweiterung der Speicherhierarchie kann zur Reduzierung von Plattenzugriffen und damit zur Beschleunigung der TA-Verarbeitung genutzt werden. Wenn eine geänderte Seite aus einem lokalen Puffer ausgeschrieben werden soll, muß bei einem flüchtigen globalen Puffer weiterhin ein Schreiben auf Platte erfolgen, während bei Nichtflüchtigkeit des globalen Puffers auch solche Schreibvorgänge stark beschleunigt werden. Die Nichtflüchtigkeit erlaubt zudem, daß Änderungen systemweit im globalen Puffer akkumuliert werden können, bis irgendwann einmal ein Verdrängen aus dem globalen Puffer die Kopie auf Platte aktualisiert. Für DB-Sharing muß das Veralterungsproblem zwar immer noch gelöst werden, jedoch erlaubt der globale Systempuffer einen Austausch geänderter Seiten, was i.a. wesentlich schneller gehen dürfte als über Leitungen oder gar über Platte.

Wenn der globale Systempuffer als Erweiterung der Speicherhierarchie dienen soll, erfordert jede im lokalen Puffer nicht erfolgreiche Objektreferenz einen Zugriff auf den globalen Puffer. Um einen Engpaß zu vermeiden, ist es daher notwendig, den globalen Puffer in eine ausreichend große Anzahl von Teilbereichen zu unterteilen, die jeweils von einem eigenen Controller verwaltet werden. Jeder Teilbereich dient dabei zur Pufferung der Daten aus einer festgelegten Teilmenge der Plattenperipherie. Durch diese Zuordnung ist jedem Rechner bekannt, auf welchen Teilbereich des globalen Systempuffers zuzugreifen ist, um ein bestimmtes Datum einzulesen bzw. auszuschreiben.

Globales Logging

Dieses Einsatzgebiet betrifft nur DB-Sharing und bezieht sich auf die beschleunigte Erstellung der globalen Log-Datei durch Verwendung eines *globalen Log-Puffers*, für den Seitenadressierbarkeit ausreicht. Ein nichtflüchtiger Halbleiterspeicher ist zwar sinnvoll, jedoch nicht zwingend notwendig, da die Log-Daten in den lokalen Log-Dateien gesichert sind. Ein Ausschreiben auf den globalen Log-Puffer ist wegen des schnellen Zugriffs parallel zum lokalen Logging möglich, so daß das Mischen der lokalen Log-Daten sehr einfach werden dürfte. Es muß jedoch dafür Sorge getragen werden, daß die Log-Daten nur kurz im Log-Puffer verbleiben, um einem Überlauf vorzubeugen, der bei dem zu erwartenden Log-Umfang leicht möglich ist [Ra86a]. Insbesondere ist zur Reduzierung des Log-Volumens eintragsweises Logging und Gruppen-Commit vorzusehen. Zur Vermeidung von Engpaßsituationen, ist u.U. für den globalen Log-Puffer (ähnlich wie bei einem globalen Systempuffer) eine Partitionierung in Teilbereiche erforderlich, insbesondere bei einem hohen Anteil von Änderungs-TA.

Lastkontrolle

Hier wäre der Einsatz eines gemeinsamen Halbleiterspeichers sinnvoll, wenn die Lastkontrolle von mehreren Rechnern (z.B. Front-Ends) durchgeführt wird. Er könnte zur Ablage gemeinsam benutzter Datenstrukturen verwendet werden, die etwa Informationen für TA-Routing oder zur Auslastung der Rechner enthalten. Auch hier ist für einen schnellen Zugriff Instruktionsadressierbarkeit erforderlich. Weiterhin könnten über einen gemeinsamen Speicherbereich sehr schnell Ein- und Ausgabenachrichten zwischen den Front-Ends und den Verarbeitungsrechnern ausgetauscht werden. Diese Einsatzformen einer nahen Kopplung sind sowohl für DB-Sharing als auch für DB-Distribution möglich.

TA-Bearbeitung

Dieser Aspekt betrifft nur DB-Distribution, wo ein gemeinsames Speichersegment zum beschleunigten Austausch von DML-Operationen und/oder Daten zwischen den Rechnern benutzt werden könnte. Hierzu müßte Instruktionsadressierbarkeit des gemeinsamen Speichers vorliegen.

Diskussion

Ob und in welchem Umfang gemeinsame Halbleiterspeicher in einem Mehrrechner-DBS eingesetzt werden sollten, ist u.a. abhängig von den geforderten Verfüg-

barkeitsanforderungen, den Kosten sowie der Anzahl der Rechner, die an den gemeinsamen Halbleiterspeicher anschließbar sind (Erweiterbarkeit!). Ein instruktionsadressierbarer, flüchtiger Halbleiterspeicher läßt sich nahezu immer zu Beschleunigung von Interprozessor-Kommunikationen einsetzen. Dieser Speichertyp birgt jedoch ähnliche Gefahren bezüglich der Verfügbarkeit wie der gemeinsame Hauptspeicher in eng gekoppelten Mehrrechnersystemen in sich. Daher muß durch geeignete Vorkehrungen und Protokolle dafür gesorgt werden, daß ein Rechnerausfall keine „unerwünschten" Spuren in dem gemeinsamen Speicherbereich hinterläßt. Insbesondere muß durch gegenseitige Überwachung verhindert werden, daß kritische Abschnitte endlos gesperrt bleiben. Wegen der stärkeren Entkopplung dürften seitenadressierbare, gemeinsame Speicherbereiche, die zur Realisierung eines globalen Systempuffers bzw. eines globalen Log-Puffers einsetzbar sind, bessere Verfügbarkeitsmerkmale aufweisen.

Eine hohe Verfügbarkeit ist umso schwieriger zu realisieren, je mehr Funktionen zentralisiert werden, v. a. bei Verwendung von flüchtigen Halbleiterspeichern. Wenn z. B. bei einem gemeinsam benutzten flüchtigen Halbleiterspeicher der Ausfall eines Rechners den Ausfall dieses Speichers bewirkt, besitzt dieser gemeinsame Speicher eine sehr hohe Ausfallwahrscheinlichkeit. Aus diesen Überlegungen heraus scheint es sinnvoll, auf den Einsatz einer nahen Kopplung über instruktionsadressierbare Halbleiterspeichersegmente möglichst zu verzichten.

Da alle vorgestellten Einsatzmöglichkeiten einer nahen Kopplung für DB-Distribution (Lastkontrolle, Synchronisation, TA-Bearbeitung) diesen Speichertyp erfordern, ist *für DB-Distribution eine nahe Kopplung* schon aus Verfügbarkeitsgründen *nicht empfehlenswert.* Außerdem würde durch eine nahe Kopplung die ‚shared nothing'-Eigenschaft aufgegeben, die Voraussetzung für eine Ortsverteilung der Rechner ist. Für DB-Sharing dagegen verbleiben zwei vielversprechende Verwendungsmöglichkeiten einer gemeinsamen Speicherpartition, nämlich zur Realisierung eines globalen Systempuffers und eines globalen Log-Puffers. Inwieweit bei DB-Sharing eine Hardwareunterstützung bei der Synchronisation möglich bzw. sinnvoll ist, kann zur Zeit noch nicht abgeschätzt werden.

6 Zusammenfassende Bewertung

Nach der Diskussion allgemeinerer Aspekte, der Realisierung einzelner Systemkomponenten sowie von Optimierungsmöglichkeiten über eine nahe Rechnerkopplung sollen abschließend die angesprochenen Vor- und Nachteile in einer einfachen Bewertung zusammengefaßt werden. Dazu werden in der folgenden Tabelle für die drei Realisierungsformen von Mehrrechner-DBS (eng gekoppelt, DB-Sharing, DB-Distribution) und den angesprochenen Unterscheidungskriterien Noten zwischen 1 (Bestwert) und 3 vergeben. Hierzu wurden die einzelnen Vergleichskriterien in drei Gruppen eingeteilt. Im ersten Teil der Tabelle wird zunächst die Tauglichkeit der Ansätze zur Erfüllung der wichtigsten Anforderungen an Hochleistungs-DBS bewertet. Bei den nächsten vier Vergleichskriterien handelt es sich ebenfalls um allgemeinere Aspekte (die jedoch nicht als Anforderungen anzusehen sind), inklusive der Optimierungsmöglichkeiten über eine nahe Rechnerkopplung. Schließlich erfolgt im dritten Teil eine Bewertung des Realisierungsaufwandes der einzelnen Systemkomponenten im Vergleich zu zentralen DBS. Hier wurde für eng gekoppelte Mehrrechner-DBS stets die Note ‚1' vergeben. Durch die Unterschiedlichkeit der aufgenommenen Aspekte sowie deren vom Anwendungsfall abhängigen Relevanz ist klar, daß sich die ‚beste' Realisierungsart nicht durch Ermittlung des Notendurchschnitts bestimmen läßt.

Kriterien	eng gekoppelte Mehrrechner-DBS	DB-Sharing	DB-Distribution
Hohe Verfügbarkeit	3	1	2
Erweiterbarkeit	3	1	2
Handhabbarkeit	1	2	3
Ortsverteilung	3	3	1
Hohe TA-Raten	3	?	?
Kurze Antwortzeiten	1	?	?
DB-Entwurf	1	1	3
TA-Bearbeitung	1	1	2
Verfügbare Lösungen	1	3	2
Nahe Kopplung	–	1	2
Systempufferverwaltung	1	2	1
Synchronisation	1	3	2
Lastkontrolle	1	3	2
Logging	1	2	1
Recovery	1	3	2

Die Tabelle zeigt, daß eng gekoppelte Mehrrechner-DBS für wesentliche Anforderungen die schlechteste Lösung darstellen, so daß die eigentliche Alternative DB-Sharing oder DB-Distribution lautet. Für diese beiden Ansätze wurden bezüglich den Performance-Aspekten bewußt keine Benotungen vorgenommen, da hierzu noch aussagekräftige quantitative Analysen fehlen und eine starke Abhängigkeit zu der jeweiligen TA-Last sowie der Realisierung einzelner Systemfunktionen besteht.

Die wichtigsten Vorteile von DB-Distribution im Vergleich zu DB-Sharing liegen in der Möglichkeit zur Ortsverteilung, daß viele Systemfunktionen einfacher zu realisieren sind und daß dafür bereits eine Fülle von Lösungsvorschlägen bekannt sind, wenngleich diese nicht immer Hochleistungsanforderungen genügen (z. B. bei der Lastkontrolle). Andererseits bewirkt die Notwendigkeit der Datenverteilung nicht nur einen aufwendigen und schwierigen DB-Entwurf, sondern schränkt die Flexibilität bezüglich Änderungen in der Systemkonfiguration erheblich ein, da eine Neupartitionierung der Daten nur sehr umständlich durchführbar ist. Die statische Partitionierung der Daten kann auch im Normalbetrieb zu Leistungseinbußen führen, da sie kaum Möglichkeiten zur Lastbalancierung oder zur Minimierung des Nachrichtenaufkommens erlaubt; insbe-

sondere ist kein kurzfristiges Anpassen der Datenverteilung an (größere) Schwankungen im TA-Profil, wie sie mehrmals täglich vorkommen können, möglich. Zudem sind mit DB-Distribution Hochleistungs-DBS praktisch nur mit relationalen Systemen möglich, da nur damit Minimalanforderungen bezüglich der Änderbarkeit der Datenverteilung erfüllbar sind.

DB-Sharing dagegen kann als natürliche Weiterentwicklung von zentralisierten DBS auf Mehrrechner-DBS angesehen werden, da beim Übergang auf ein DB-Sharing-System bestehende Anwendungsprogramme und Datenbanken nicht geändert zu werden brauchen. Die DB-Sharing-Architektur erlaubt es auch – wegen der fehlenden Datenverteilung – in einfacher Weise das System zu erweitern oder den Ausfall eines Rechners zu verkraften, ohne daß die Verfügbarkeit der Daten beeinträchtigt wird. Allerdings ergibt sich eine schwierige Realisierung insbesondere für die Synchronisation, die Lastkontrolle und die Recovery; die Lösungen sind zudem wegen der starken wechselseitigen Abhängigkeiten geeignet aufeinander abzustimmen. Die bei geeigneten Lösungen ereichbare Flexibilität v.a. bei der Lastkontrolle ergibt aber auch weitaus größere Möglichkeiten zur Lastbalancierung, zur Reduzierung von Außenbeziehungen und zu einem wirkungsvolleren Reagieren auf Lastschwankungen als bei DB-Distribution. Weitere Leistungsvorteile dürften sich durch die besseren Einsatzmöglichkeiten einer nahen Rechnerkopplung ergeben. Schließlich ist die Realisierung eines Hochleistungs-DBS nicht nur auf relationale Systeme beschränkt, sondern auch mit netzwerkartigen und hierarchischen Datenmodellen möglich.

Literatur

[Ba86] *Bayer, R.:* Consistency of Transactions and Random Batch, in: ACM Trans. on Database Systems, Vol. 11, No. 4, 1986, pp. 397–404.

[BG81] *Bernstein, P. A., Goodman, N.:* Concurrency Control in Distributed Database Systems, in: ACM Comp. Surveys, Vol. 13, No. 2, Juni 1981, pp. 195–221.

[CDY86] *Cornell, D. W., Dias, D. M., Yu, P. S.:* On Multisystem Coupling through Function Request Shipping, in: IEEE Trans. on Software Engineering, Vol. SE-12, No. 10, 1986, pp. 1006–1017.

[Da81] *Dadam, P.:* Synchronisation in verteilten Datenbanken: Ein Überblick, in: Informatik Spektrum 4, 1981, S. 175–184 und S. 261–270.

[DIY86] *Dias, D. M., Iyer, B. R., Yu, P. S.:* On Coupling Many Small Systems for Transaction Processing, in: Proc. 13th Annual Int. Symp. on Comp. Architecture, 1986, pp. 104–110.

[Ga85] *Gawlick, D.:* Processing ‚Hot Spots' in High Performance Systems, in: Proc. IEEE Spring CompCon, 1985, pp. 249–251.

[Ga87] *Gawlick, D.:* Persönliche Kommunikation, Febr. 1987.

[GA85] *Gray, J., Anderson, M.:* Distributed Databases – Four Case Studies, Tandem TR 85.5, Juni 1985.

[Gr85] *Gray, J., Good, B., Gawlick, D., Homan, P., Sammer, H.:* One Thousand Transactions per Second, in: Proc. IEEE Spring CompCon, 1985, pp. 96–101.

[Gr86] *Gray, J.:* Why Do Computers Stop and What can be Done About it, in: Proc. 5th Symp. on Reliability in Distr. Software and Database Systems, 1986, pp. 3–12.

[Hä86] *Härder, T.:* DB-Sharing vs. DB-Distribution – die Frage nach dem Systemkonzept zukünftiger DB/DC-Systeme, in: Tagungsband zur 9. NTG/GI-Fachtagung über Architektur und Betrieb von Rechensystemen, VDE-Verlag, NTG-Fachberichte 92, 1986, S. 151–165.

[He85] *Helland, P.:* High Transaction Rates in a Distributed System, Int. Workshop on High Performance Transaction Systems, Asilomar, Sep. 1985.

[HM86a] *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.

[HM86b] *Härder, T., Meyer-Wegener, K.:* Die Zusammenarbeit von TP-Monitoren und Datenbanksystemen in DB/DC-Systemen – Existierende Systeme und zukünftige Entwicklungen, in: Informatik – Forschung und Entwicklung, Vol. 1, No. 3, 1986, S. 101–122.

[HMMS87] *Härder, T., Meyer-Wegener, K., Mitschang, B. Sikeler, A.:* PRIMA – a DBMS Prototype Supporting Engineering Applications, SFB-Bericht 22/87, FB Informatik, Univ. Kaiserlautern, 1987.

[HR83] *Härder, T., Reuter, A.:* Principles of Transaction-Oriented Database Recovery, in: ACM Computing Surveys, Vol. 15, No. 4, 1983, pp. 287–317.

[HR86] *Härder, T., Rahm, E.:* Mehrrechner-Datenbanksysteme für Transaktionssysteme hoher Leistungsfähigkeit, in: Informationstechnik, Vol. 28, No. 4, 1986, S. 214–225.

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

[KLS86] *Kronenberg, N. P., Levy, H. M., Strecker, W. D.:* VAX clusters: A Closely-Coupled Distributed System, in: ACM Trans. on Comp. Systems, Vol. 4, No. 2, 1986, pp. 130–146.

[KR81] *Kung, H. T., Robinson, J. T.:* On Optimistic Methods for Concurrency Control, in: ACM Trans. on Database Systems, Vol. 6, No. 2, 1981, pp. 213–226.

[ON86] *O'Neil, P. E.:* The Escrow Transactional Method, in: ACM Trans. on Database Systems, Vol. 11, No. 4, 1986, pp. 405–430.

[Ra86a] *Rahm, E.:* Nah gekoppelte Rechnerarchitekturen für ein DB-Sharing-System, in: Proc. 9. NTG/GI-Fachtagung über Architektur und Betrieb von Rechensystemen, VDE-Verlag, NTG-Fachberichte 92, 1986, S. 166–180.

[Ra86b] *Rahm, E.:* Concurrency Control in DB-Sharing Systems, in: Proc. 16. GI-Jahrestagung, Springer-Verlag, Informatik Fachberichte 126, 1986, pp. 617–632.

[Ra86c] *Rahm, E.:* Primary Copy Synchronization for DB-Sharing, in: Information Systems, Vol. 11, No. 4, 1986, pp. 275–286.

[Ra87a] *Rahm, E.:* Performance Analysis of Primary Copy Synchronization in Database Sharing Systems, Interner Bericht 165/87, FB Informatik, Univ. Kaiserslautern, 1987.

[Ra87b] *Rahm, E.:* Optimistische Synchronisationsverfahren in Datenbanksystemen: Ein Überblick, Interner Bericht 166/87, FB Informatik, Univ. Kaiserslautern, 1987.

[Re82] *Reuter, A.:* Concurrency on High-Traffic Data Elements, in: Proc. ACM Symp. on Principles of Database Systems, 1982, pp. 83–92.

[Re86a] *Reuter, A.:* Load Control and Load Balancing in a Shared Database Management System, in: Proc. 2nd Data Eng. Conf., 1986, pp. 188–197.

[Re86b] *Reuter, A.:* Mehrrechner-Architekturen für Datenbanksysteme, in: Proc. 9. NTG/GI-Fachtagung über Architektur und Betrieb von Rechensystemen, VDE-Verlag, NTG-Fachberichte 92, 1986, S. 141–150.

[RDPSZ86] *Reuter, A., Duppel, N., Peinl, P., Schiele, G., Zeller, H.:* An Outlook on PROSPECT, Institut für Informatik, Universität Stuttgart, 1986.

[RS84] *Reuter, A., Shoens, K.:* Synchronization in a Data Sharing Evironment, IBM San Jose Research Laboratory, vorläufige Fassung, 1984.

[Se84] *Sekino, A. et al.:* The DCS – A New Approach to Multisystem Data Sharing, in: Proc. National Comp. conf., 1984, pp. 59–68.

[Sh85] *Shoens, K. et al.:* The AMOEBA Project, in: Proc. IEEE Spring CompCon, 1985, pp. 102–105.

[Sh86] *Shoens, K.:* Data Sharing vs. Partitioning for Capacity and Availability, in: IEEE Quarterly Bulletin Database Engineering, Vol. 9, No. 1, 1986, pp. 10–16.

[St86] *Stonebraker, M.:* The Case for Shared Nothing, in: IEEE Quarterly Bulletin Database Engineering, Vol. 9, No. 1, 1986, pp. 4–9.

[Tr83] *Traiger, I.:* Trends in Systems Aspects of Database Management, in: Proc. 2nd Int. Conf. on Databases (ICOD-2), 1983, pp. 1–20.

[We83] *West, J. C. et al.:* PERPOS Fault-Tolerant Transaction Processing, in: Proc. 3rd Symp. on Reliability in Distr. Software and Database Systems, 1983, pp. 189–194.

[YCDT86] *Yu, P. S., Cornell, D. W., Dias, D. M., Thomasian, A.:* On Coupling Partitioned Data Systems, in: Proc. 6th Int. Conf. on Distr. Comp. Systems, 1986, pp. 148–157.

Diese Arbeit wurde teilweise von der SIEMENS AG finanziell unterstützt.

Prof. Dr. *Theo Härder* (41), 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 im IBM Research Laboratory, 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, Juli 1986–Feb. 1987 Visiting Scientist am IBM Almaden Research Center, San Jose, Calif. Besondere Forschungsinteressen: Leistungsanalyse von DB- und DB/DC-Systemen, Mehrrechner-Datenbanksysteme, Non-Standard-Anwendungen für DBS (CAD/CAM, XPS), Datenbanken in Netzen.

Dipl.-Inform. *Erhard Rahm* (28), Studium der Informatik (1979–1984) an der Universität Kaiserlautern 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, FB Informatik, Postfach 3049, D-6750 Kaiserslautern.

Datenbankzugriff in Rechnernetzen

Database access in networks

Wolfgang Effelsberg, IBM Heidelberg

Die Einführung von preiswerten Arbeitsplatzrechnern hat die Systemumgebung für Datenbanksysteme beträchtlich verändert. Neben einfachen Terminals, die mit einem Zentralrechner verbunden sind, setzt sich in zunehmendem Maße dezentrale Rechnerleistung bis hin zu kleinen Arbeitsplatzrechnern durch, die untereinander vernetzt sind und für spezielle Dienste auf zentrale Rechner (Server) im Netz zugreifen können. Solche Dienststationen sind bereits für Dateisysteme und Drucker gebräuchlich. Schwieriger ist dagegen die Bereitstellung eines Datenbankdienstes im Netz.

Der Aufsatz klassifiziert zunächst die Vielfalt der Datenbank-Zugriffstechniken in die drei Grundmodelle Terminalnetz, Netz aus autonomen Rechnern und verteiltes Datenbanksystem. Auf jedes dieser drei Grundmodelle wird dann näher eingegangen. Die Verwendung von standardisierten Kommunikationsprotokollen erlaubt den Fernzugriff auch in heterogenen Netzen, in denen Rechner unterschiedlicher Architektur (Hard- und Software) zusammenarbeiten.

The availability of inexpensive workstations has changed the hardware and software environment for database systems considerably. In addition to terminals connected to a mainframe, interconnected workstations with access to servers in a network are now becoming more and more popular. File servers and print servers in such networks are already widely used. Providing a database server in a network is more difficult.

This paper classifies database access techniques into three basic models: Terminal networks, networks of autonomous computers, and distributed database management systems. These three models are then discussed in detail. Standardized communication protocols allow remote database access in heterogeneous networks, connecting computers of different architectures (hardware and software).

1 Einführung

In den letzten Jahren haben Rechnernetze zunehmend an Bedeutung gewonnen. Die schnelle Verbreitung von preisgünstigen Arbeitsplatzcomputern (PCs) macht autonome Rechnerleistung an einer steigenden Zahl von Arbeitsplätzen verfügbar. Sehr schnell zeigt sich dabei,