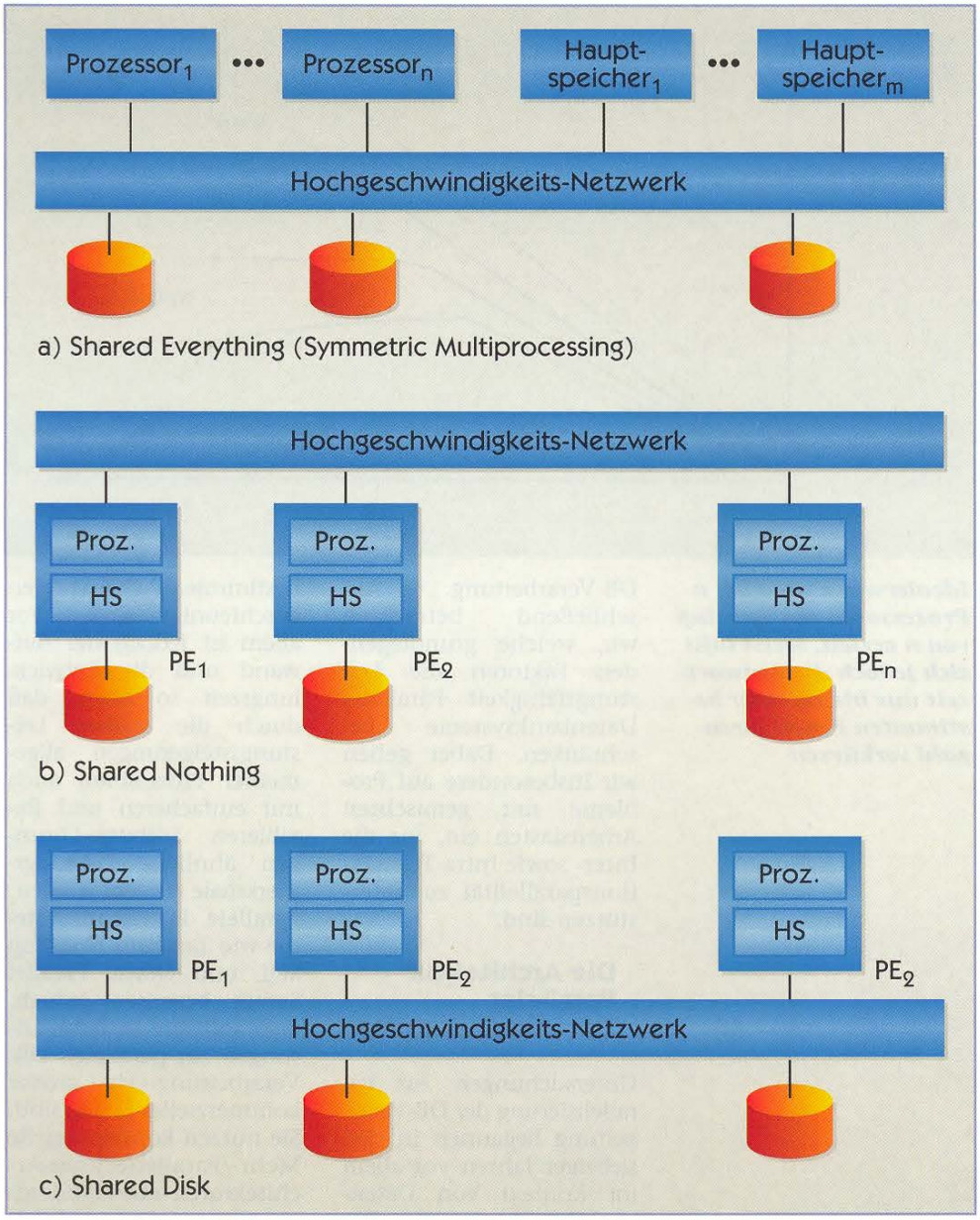
Parallele Datenbanksysteme

Die Zukunft kommerzieller Datenbanksysteme liegt in der Parallelverarbeitung. Parallele Datenbanksysteme nutzen die Verarbeitungskapazität zahlreicher Prozessoren – im Rahmen von Multiprozessoren oder Parallelrechnern – zur Leistungssteigerung. Hauptziel ist die Verkürzung der Bearbeitungszeit von Transaktionen und Anfragen (Queries), deren sequentielle Bearbeitung inakzeptable Antwortzeiten verursachen würde.

ERHARD RAHM

Eine treibende Kraft für die zunehmende Notwendigkeit von Parallelen Datenbanksystemen sind die ständig wachsenden Datenvolumina, auf denen DB-Operationen und Dienstprogramme abzuwickeln sind. Die größten, kommerziell genutzten relationalen Datenbanken umfassen bereits heute mehrere Terabyte, wobei einzelne Relationen (Tabellen) über 100 Gigabyte belegen. Die sequen-



Drei Architekturen kommen zur Realisierung von Parallelen Datenbanksystemen in Betracht: Shared Everything, Shared Nothing sowie Shared Disk

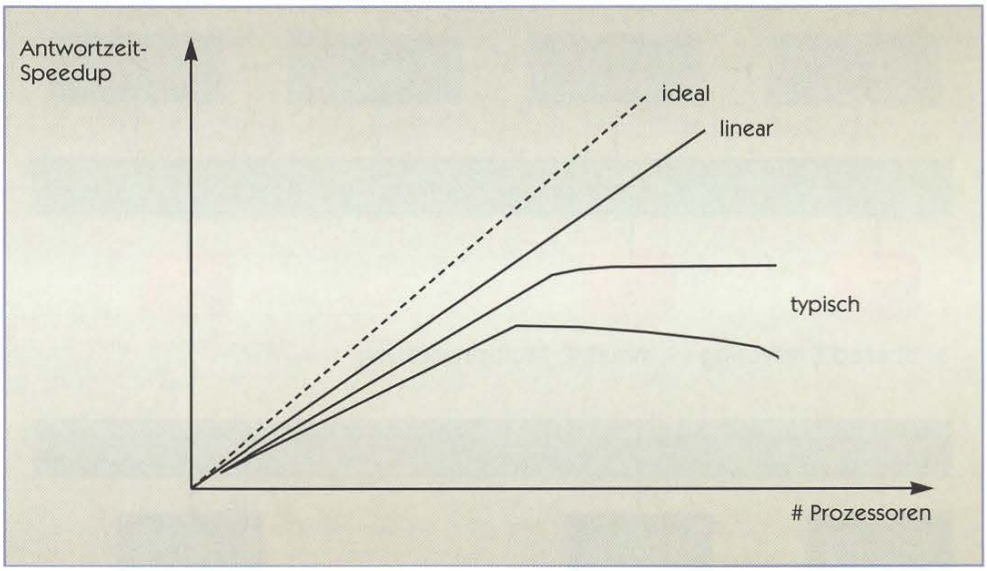
tielle Verarbeitung von DB-Operationen ist demgegenüber sehr langsam, trotz ständig verbesserter Hardware und somit erhöhten Geschwindigkeiten. So ist das sequentielle Einlesen der Daten von Magnetplatten typischerweise auf etwa 5 MByte/s beschränkt. Auch die Ausführungszeiten relationaler Operatoren auf hauptspeicherresidenten Daten ist relativ langsam. So sind für die einfachste Operation, die sequentielle Suche im Rahmen eines Relationen-Scans, etwa 1000 Instruktionen pro Satz anzusetzen. Damit ist die sequentielle Suche auf einem Prozessor von 100 MIPS auf 10 MByte/s beschränkt (Satzlänge: 100 Byte). Komplexere Operationen wie Sortieren oder Join-Bildung sind typischerweise um mindestens eine Größenordnung langsamer (weniger als 1 MByte/s). Die sequentielle Verarbeitung einer relationalen DB-Operation auf 1 TByte würde somit Bearbeitungszeiten von mehreren Tagen erfordern – und dies auch nur, wenn keine Behinderungen mit anderen Transaktionen auftreten (Einbenutzerbetrieb). Offensichtlich sind solche Bearbeitungszeiten in den allermeisten Fällen nicht tolerierbar. Eine Verschärfung der Problematik ergibt sich für neuere DB-Anwendungen, für die zunehmend objekt-orientierte Datenbanksysteme verwendet werden. Multimedia-Anwendungen verlangen so den Zugriff auf sehr große Datenmengen mit sehr restriktiven Antwortzeitvorgaben, zum Beispiel um digitalisierte Filme in hoher Qualität abzuspielen oder um Videoaufnahmen in Echtzeit digital zu speichern.

Parallele Prozesse für CAD/Multimedia

CAD- (Computer Aided Design) oder CASE-Anwendungen (Computer Aided Software Engineering) verursachen beispielsweise umfangreiche Operationen auf komplex strukturierten Objekten mit einem Berechnungsaufwand, der denen einfacher relationaler Operatoren um ein Vielfaches übersteigt. In diesen Bereichen ist der Einsatz einer parallelen Bearbeitung noch dringlicher, um für den Dialogbetrieb ausreichend kurze Bearbeitungszeiten zu ermöglichen. Die schnelle Bearbeitung komplexer DB-Anfragen verlangt den Einsatz von Intra-Transaktionsparallelität, also die parallele Verarbeitung innerhalb ei-

ner Transaktion oder Anfrage. Daneben müssen Parallele Datenbanksysteme in der Lage sein, für OLTP-Anwendungen durch Nutzung mehrerer Prozessoren hohe Transaktionsraten zu erzielen. OLTP (Online Transaction Processing) ist der kommerziell wichtigste Einsatzbereich heutiger Datenbanksysteme, etwa bei Reservierungssystemen oder Bankanwendungen (Rahm 1993). Typischerweise sind die Transaktionen hierbei sehr einfach, jedoch in großer Häufigkeit auszuführen. Die Durchsatzforderungen verlangen zwingend die gleichzeitige Bearbeitung zahlreicher unabhängiger Transaktionen, also einen Mehrbenutzerbetrieb beziehungsweise den Einsatz von Inter-Transaktionsparallelität. Eine besondere Herausforderung an Parallele Datenbanksysteme, die von derzeitigen Implementierungen noch nicht zufriedenstellend gelöst ist, besteht nun darin, zugleich Inter- und Intra-Transaktionsparallelität effizient zu unterstützen. Neben der OLTP-Last sollen also auf der gleichen Datenbank zeitgleich eine oder mehrere komplexe Anfragen bearbeitet werden, wobei für letztere durch Nutzung von Intra-Transaktionsparallelität eine kurze Antwortzeit erreicht werden soll. Parallele Datenbanksysteme müssen ferner eine hohe Skalierbarkeit, eine hohe Verfügbarkeit (24-Stunden-Betrieb) sowie eine gute Kosteneffektivität aufweisen.

Im folgenden diskutieren wir die Architektur Paralleler Datenbanksysteme und unterscheiden mehrere Arten der parallelen



Idealerweise wird bei n Prozessoren ein Speedup von n erzielt. Meist läßt sich jedoch die Antwortzeit nur bis zu einer bestimmten Prozessorenzahl verkürzen

DB-Verarbeitung. Anschließend betrachten wir, welche grundlegenden Faktoren die Leistungsfähigkeit Paralleler Datenbanksysteme beschränken. Dabei gehen wir insbesondere auf Probleme mit gemischten Arbeitslasten ein, für die Inter- sowie Intra-Transaktionsparallelität zu unterstützen sind.

Die Architektur Paralleler Datenbanksysteme

Untersuchungen zur Parallelisierung der DB-Verarbeitung begannen in den siebziger Jahren vor allem im Kontext von Datenbank-Maschinen. Die Mehrzahl der Vorschläge versuchte durch Nutzung von Spezial-Hardware bestimmte Operationen zu beschleunigen. Im Rückblick ist festzustellen, daß diese Ansätze weitgehend gescheitert sind (DeWitt/Gray 1992). Der Grund liegt vor allem darin, daß hardware-basierte Ansätze zur DB-Verarbeitung naturgemäß eine nur eingeschränkte Funktionalität unterstützen, so daß nur bestimmte Operationen beschleunigt werden. Vor allem ist jedoch der Aufwand und die Entwicklungszeit so hoch, daß durch die starken Leistungssteigerungen allgemeiner Prozessoren auch mit einfacheren und flexibleren Software-Lösungen ähnliche Leistungsmerkmale erreicht werden. Parallele Datenbanksysteme wie Tandems NonStop SQL und Oracle Parallel Server beweisen jedoch, daß software-basierte Lösungen zur parallelen DB-Verarbeitung ein großer kommerzieller Erfolg sind. Sie nutzen konventionelle Mehr-/Parallelrechner-Architekturen mit Standardprozessoren und -peripherie, wobei zur Zeit Installationen mit über 200 Prozessoren bestehen. Insbesondere bei Einsatz von Mikroprozessoren wird damit eine hohe Kosteneffektivität möglich. Zudem kann die rasante Leistungssteigerung bei Mikroprozessoren unmittelbar genutzt werden.

Eine weitere Anforderung an die Hardware-Architektur für Parallele Datenbanksysteme ist die lokale

Verteilung der Prozessoren, zum Beispiel im Rahmen eines Clusters in einem Maschinenraum. Denn dies gestattet den Einsatz eines skalierbaren Hochgeschwindigkeits-Netzwerkes, dessen Übertragungskapazität proportional zur Prozessoranzahl gesteigert werden kann.
Eine sehr schnelle Inter-Prozessor-Kommunikation ist für die effiziente Zerlegung einer Transaktion in zahlreiche Teiltransaktionen sowie für die Übertragung großer Datenmengen entscheidend. Weiterhin kann bei lokaler Verteilung am ehesten eine dynamische Lastbalancierung erreicht werden, wobei der aktuelle Systemzustand bei der Lastverteilung berücksichtigt wird. Dies betrifft sowohl die initiale Verteilung eintreffender Transaktionen und Anfragen unter den Verarbeitungsrechnern (Transaktions-Routing) als auch die Rechnerzuordnung von Teilaufträgen während der parallelisierten Abarbeitung.
Diese Anforderungen werden von drei allgemeinen Architekturen erfüllt, die zur Realisierung von Parallelen Datenbanksystemen in Betracht kommen: Shared Everything, Shared-Nothing sowie Shared-Disk. Die Ansätze können wie folgt charakterisiert werden:

- **Shared-Everything (Multiprozessor-Datenbanksysteme)**

Die DB-Verarbeitung erfolgt auf einem Multiprozessor. Hierbei liegt eine enge Kopplung vor, bei der sich alle Prozessoren einen gemeinsamen Hauptspeicher teilen, über den eine effiziente Kooperation möglich ist (Verwendung gemeinsamer Datenstrukturen). Lastbalancierung sind meist einfach (durch das Betriebssystem) durchzuführen, indem gemeinsame Auftragswarteschlangen im Hauptspeicher gehalten werden, auf die alle Prozessoren zugreifen können.
Allerdings bestehen Verfügbarkeitsprobleme, da der gemeinsame Hauptspeicher nur eine geringe Fehlerisolation bietet und Software wie das Betriebssystem oder das Datenbanksystem nur in einer Kopie vorliegt. Auch die Erweiterbarkeit ist in der Regel stark begrenzt, da mit wachsender Prozessoranzahl der Hauptspeicher beziehungsweise die für den Zugriff auf gemeinsame Hauptspeicherinhalte erforderliche Synchronisierung leicht zum Engpaß werden.
Die Bezeichnung „Shared Everything" resultiert daher, daß neben dem Hauptspeicher auch Terminals sowie Externspeicher von allen Prozessoren erreichbar sind.

- **Shared-Nothing**

Die DB-Verarbeitung erfolgt durch mehrere autonome Rechner oder Prozessor-Elemente (PE), die jeweils über eigene Hauptspeicher sowie eine eigene Instanz des Datenbanksystems sowie anderer Software-Komponenten verfügen.
Es liegt eine lose Kopplung vor, bei der sämtliche Kommunikationsvorgänge durch Nachrichtenaustausch (message passing) realisiert werden. Die Externspeicher sind unter den Rechnern partitioniert, so daß jede Datenbanksystem-Instanz nur auf Daten der lokalen Partition direkt zugreifen kann.

- **Shared-Disk (Database Sharing)**

Wie bei Shared-Nothing liegt eine Menge lose gekoppelter PE mit je einer Datenbanksystem-Instanz vor. Es besteht jedoch eine gemeinsame Externspeicherzuordnung, so daß jedes PE (jedes Datenbanksystem) direkten Zugriff auf die gesamte Datenbank hat. An das Kommunikationsnetzwerk sind hierbei besonders hohe Anforderungen gestellt, da die Plattenzugriffe aller Rechner darüber abzuwickeln sind (zahlreiche Seitentransfers).
Daneben sind auch hybride Architekturen denkbar, zum Beispiel Shared-Nothing- oder Shared-Disk-Systeme, bei denen jeder Knoten vom Typ Shared-Everything ist. Statt der hier verwendeten Architekturbezeichnungen findet man auch häufig die Begriffe SMP (Symmetric Multiprocessing) und MPP (Massive Parallel Processing). SMP-Systeme entsprechen Multiprozessoren und somit Shared-Everything-Konfigurationen. MPP-Systeme sind lose gekoppelte Parallelrechnersysteme, welche sowohl dem Shared-Nothing- oder Shared-Disk-Ansatz folgen können. Bisherige Implementierungen und Untersuchungen von Intra-Transaktionsparallelität erfolgten vor allem für Shared-Everything-(Multiprozessoren) und für Shared-Nothing-Systeme. Die Nutzung von Multiprozessoren zur Parallelverarbeitung kann dabei als erster Schritt aufgefaßt werden, da er meist auf relativ wenige Prozessoren begrenzt ist (meist unter zehn).

IN DER PRAXIS KOMMEN SELTEN MEHR ALS ZEHN PROZESSOREN ZUM EINSATZ

Informix nutzt Multiprozessoren

Die Verwendung eines gemeinsamen Hauptspeichers erleichtert dabei die Parallelisierung, da eine effiziente Kommunikation zwischen Prozessen einer Transaktion und eine einfachere Lastbalancierung unter den Prozessoren möglich wird. Kommerzielle Datenbanksysteme nutzen zunehmend Multiprozessoren für Intra-Transaktionsparallelität, zum Beispiel DB2 oder Informix.
Shared-Nothing- und Shared-Disk-Systeme ermöglichen eine größere Anzahl von Prozessoren als Shared-Everything, so daß

eine weitergehende Parallelisierung erreichbar wird. Shared-Nothing erfordert, die Datenbank so unter den Prozessorknoten aufzuteilen, daß alle Rechner gut für die parallele DB-Verarbeitung genutzt werden können. Für relationale Datenbanksysteme erfolgt hierzu im allgemeinen eine horizontale (zeilenweise) Partitionierung von Relationen, die durch Spezifikation einer Verteilungsfunktion (meist Hash-Funktion oder Wertebereichsunterteilung) auf einem ausgezeichneten Verteilungsattribut (meist dem Primärschlüssel) definiert wird. In einer Bankanwendung könnte so eine Konto-Relation durch eine Verteilungsfunktion auf dem Attribut Kontonummer unter n Rechnern aufgeteilt werden. Anfragen auf der Kontorelation (zum Beispiel Berechnung der Summe aller Kontostände) können dann parallel von allen n Knoten bearbeitet werden. Problematisch bei dieser Vorgehensweise ist jedoch, daß der Parallelitätsgrad von Anfragen sowie die ausführenden Rechner durch die Datenbankverteilung schon statisch festgelegt sind.

Optimierung der Anfragen nötig

Erweiterungen gegenüber zentralisierten Datenbanksystemen, die von Shared-Nothing-Datenbanksystemen zu unterstützen sind, betreffen neben der Definition einer Datenbankverteilung vor allem die Anfrageoptimierung, um zu effizient ausführbaren, parallelen Ablaufplänen zu gelangen. Hierbei sind mehrere Arten der Parallelisierung zu unterstützen, welche im folgenden angesprochen werden. Weiterhin ist ein verteiltes Commit-Protokoll zu unterstützen, um bei verteilter Transaktionsausführung die Atomarität von Transaktionen zu gewährleisten. Intra-Transaktionsparallelität wird bereits seit mehreren Jahren in den beiden kommerziellen Shared-Nothing-Systemen Teradata DBC/1024 und Tandem NonStop-SQL unterstützt.

In Shared-Disk-Systemen ist aufgrund der gemeinsamen Externspeicherzuordnung keine physische Datenaufteilung unter den Rechnern vorzunehmen. Insbesondere kann somit eine DB-Operation auch von jedem Rechner gleichermaßen abgearbeitet werden, wodurch eine hohe Flexibilität zur dynamischen Lastbalancierung entsteht. Wenn keine Intra-Transaktionsparallelität genutzt werden soll, kann eine Transaktion daher auch stets an einem Rechner bearbeitet werden (keine Notwendigkeit für verteilte Ausführungspläne sowie ein verteiltes Commit-Protokoll). Zur internen Parallelisierung komplexer Anfragen bestehen daneben weit mehr Freiheitsgrade als bei Shared-Nothing, da weder der Parallelitätsgrad noch die Rechner, welche die Teiloperationen ausführen sollen, vorab durch die Datenverteilung festgelegt sind. In Shared-Disk-Datenbanksystemen wird Kommunikation zwischen den Rechnern vor allem zur globalen Synchronisation der DB-Zugriffe notwendig, wozu oft rechnerübergreifende Sperrverfahren verwendet werden. Ein weiteres Problem stellt die Behandlung sogenannter Pufferinvalidierungen dar, die sich dadurch ergeben, daß jedes Datenbanksystem Seiten der gemeinsamen Datenbank in seinem lokalen Hauptspeicherpuffer hält. Die dadurch erforderliche Kohärenzkontrolle läßt sich jedoch gut ins Sperrprotokoll integrieren, da vor jedem Objektzugriff eine Sperre anzufordern ist, so daß die Aktualität einer gepufferten Objektkopie bei der Sperrvergabe geprüft werden kann (Rahm 1994).

BEI DATENBANKEN IST DIE PARALLELISIERUNG ERFOLGREICH

Die Parallelverarbeitung für DB-Anwendungen kann auf unterschiedliche Arten erfolgen, die in Rahm (1994) klassifiziert wurden.

Arten der Parallelverarbeitung

Insbesondere können verschiedene Arten von Verarbeitungsparallelität und E/A-Parallelität (Eingabe/-Ausgabe) unterschieden werden, die beide benötigt werden. Bei der Verarbeitungsparallelität, welche die Nutzung mehrerer Prozessoren verlangt, wurde schon zwischen Inter-Transaktionsparallelität (Mehrbenutzerbetrieb) und Intra-Transaktionsparallelität unterschieden. Besondere Bedeutung in Parallelen Datenbanksystemen kommt dabei der Intra-Transaktionsparallelität zu. In kommerziellen Systemen wird dazu die interne Parallelisierung von SQL-Anfragen unterstützt (Intra-Query-Parallelität).

Dies wird durch das relationale Datenmodell sowie deskriptive und mengenorientierte Abfragesprachen wie SQL ermöglicht, da in einer Abfrage große Datenmengen bearbeitet und umfangreiche Berechnungen vorgenommen werden können, so daß ein hohes Parallelisierungspotential besteht. Ein großer Vorteil dabei ist weiterhin, daß die Parallelisierung vollkommen automatisch im Datenbanksystem (durch den Datenbanksystem-Optimierer) und somit transparent für den Programmierer und DB-Benutzer möglich ist. Dies ist ein Hauptgrund für den Erfolg paralleler DB-Verarbeitung und stellt einen wesentlichen Unterschied zur Parallelisierung in an-

deren Anwendungsbereichen dar, die in vielen Fällen eine sehr schwierige Programmierung verlangen.
Eine effektive Intra-Query-Parallelität basiert zumeist auf Datenparallelität, welche eine Partitionierung der Daten erfordert, so daß verschiedene Teilabfragen auf disjunkten Datenpartitionen arbeiten. So können zum Beispiel Suchabfragen auf verschiedenen Datenpartitionen parallel ausgeführt werden. Ein großer Vorteil dabei ist, daß der Parallelitätsgrad proportional zur Datenmenge (Relationengröße) erhöht werden kann.

Parallele Algorithmen

Für komplexere Operationen wie Join-Berechnung und Sortierung existieren zahlreiche parallele Algorithmen, die zum Teil die Eingabedaten dynamisch zwischen den Rechnern umverteilen. Der Einsatz von Funktions- oder Pipeline-Parallelität ist für komplexere Operationen auch möglich, jedoch meist weniger wirkungsvoll als Datenparallelität (DeWitt/Gray 1992, Rahm 1994).
Neben Verarbeitungsparallelität ist für Parallele Datenbanksysteme die Unterstützung von E/A-Parallelität für Externspeicherzugriffe obligatorisch. Denn bei einer Sequentialisierung der Plattenzugriffe würden ansonsten sämtliche Parallelitätsgewinne bei den CPU-bezogenen Verarbeitungsanteilen wieder zunichte gemacht. E/A-Parallelität erfordert, daß Daten einer Relation über mehrere Platten verteilt sind. Dadurch kann das Lesen einer ganzen Relation parallel von mehreren Platten erfolgen, so daß eine entsprechende Verbesserung der Zugriffszeit erreicht wird (Unterstützung von Datenparallelität innerhalb einer Anfrage). Daneben können kleinere E/A-Aufträge verschiedener Transaktionen parallel verarbeitet werden, sofern sie unterschiedliche Platten betreffen (Unterstützung von Inter-Transaktionsparallelität). Eine Möglichkeit, E/A-Parallelität zu nutzen, besteht im Einsatz sogenannter Disk-Arrays, welche in der jüngsten Vergangenheit starke Bedeutung erreicht haben (Rahm1993). Sie bestehen intern aus mehreren Platten, können jedoch logisch wie eine Platte angesprochen werden, so daß ihr Einsatz prinzipiell keine Änderungen in der nutzenden Software erfordert. Ihr Einsatz zur parallelen DB-Verarbeitung kann jedoch Leistungsprobleme hervorrufen, da die Datenverteilung innerhalb des Disk-Arrays dem Datenbanksysteme nicht bekannt ist. Daher kann die vom Datenbanksystem vorgenommene Parallelisierung einer Abfrage dazu führen, daß parallele Teiloperationen auf dieselben Platten zugreifen und somit E/A-Engpässe erzeugen.

Grenzen der Parallelität

Eine wesentliche Forderung an Parallele Datenbanksysteme ist Skalierbarkeit, insbesondere in Form eines mit der Prozessoranzahl linear zunehmenden Antwortzeit-Speedups (= Antwortzeitverhältnis zwischen sequentieller und paralleler Bearbeitung). Zudem soll, vor allem für OLTP-Anwendungen, der Durchsatz proportional mit der Rechneranzahl gesteigert werden. Idealerweise wird bei n Prozessoren ein Speedup-Wert von n erzielt. Bei einem linearen Antwortzeit-Speedup kann die Antwortzeit auch proportional zur Prozessoranzahl verbessert werden, jedoch oft auf einem geringeren Niveau. Typischerweise läßt sich jedoch die Antwortzeit nur bis zu einer bestimmten Prozessoranzahl verkürzen. Eine weitere Erhöhung der Prozessorzahl führt dann ggf. zu einer Reduzierung des Speedups, also einer Zunahme der Antwortzeit.
Die suboptimale Speedup-Entwicklung basiert auf mehreren Ursachen. Zunächst ist der maximal mögliche Speedup inhärent begrenzt durch den Anteil einer Transaktion bzw. Operation, der überhaupt parallelisierbar ist (Amdahls Gesetz). Besteht zum Beispiel die Antwortzeit einer Transaktion nur zu fünf Prozent aus nicht-parallelisierbaren (sequentiellen) Verarbeitungsanteilen, so ist der maximal mögliche Speedup auf 20 Prozent beschränkt, unabhängig davon, wieviel Prozessoren eingesetzt werden. Desweiteren sind es vor allem folgende Faktoren, die den Antwortzeit-Speedup und damit die Skalierbarkeit einer Anwendung beeinträchtigen können (DeWitt/Gray 1992):

- **Startup- und Terminierungskosten**
  Das Starten und Beenden mehrerer Teiloperationen

Über den Autor

Prof. Dr. Erhard Rahm ist Lehrstuhlinhaber für Datenbanken am Institut für Informatik der Universität Leipzig. Er weist langjährige Forschungs- und Lehrerfahrung auf dem Gebiet der Parallelen Datenbanksysteme auf und ist Autor von Büchern sowie zahlreichen Veröffentlichungen in internationalen Fachzeitschriften und Tagungsbänden.

LEIDER LÄSST SICH DAS PRINZIP DER PARALLELITÄT NICHT UNBEGRENZT ANWENDEN

in verschiedenen Prozessen/ Rechnern verursacht einen Overhead, der mit dem Parallelitätsgrad zunimmt. Da umgekehrt die pro Teiloperation zu verrichtende Nutzarbeit (zum Beispiel Anzahl zu verarbeitender Sätze) sinkt, vermindert sich der relative Gewinn einer Parallelisierung mit wachsendem Parallelitätsgrad.

◆ **Interferenz**

Die Erhöhung der Prozeßanzahl führt zu verstärkten Wartezeiten auf gemeinsam benutzten Systemressourcen. Vor allem der durch die Parallelisierung eingeführte Kommunikations-Overhead kann sich negativ bemerkbar machen, insbesondere im Mehrbenutzerbetrieb (Inter-Transaktionsparallelität). Neben Wartezeiten auf physische Ressourcen (CPU, Hauptspeicher, Platten, etc.) kann es auch verstärkt zu Sperrkonflikten zwischen unabhängigen Transaktionen kommen.

◆ **Skew (Varianz der Ausführungszeiten)**

Die langsamste Teiloperation bestimmt die Bearbeitungszeit einer parallelisierten Operation. Varianzen in den Ausführungszeiten, zum Beispiel aufgrund ungleichmäßiger Daten- oder Lastverteilung oder Sperrkonflikten, führen daher zu Speedup-Einbußen. Das Skew-Problem nimmt meist mit wachsendem Parallelitätsgrad (Rechneranzahl) zu und beschränkt daher die Skalierbarkeit.

Ein noch weitgehend ungelöstes Problem ist die effektive Parallelisierung komplexer DB-Abfragen für heterogene oder gemischte Lasten, wenn parallel zu einer Abfrage andere Abfragen oder Online Transaction Processing (OLTP) im System auszuführen sind.

Die Unterstützung solcher Arbeitslasten ist bereits für zentralisierte Datenbanksysteme ein großes Problem, da es zwischen den Lasttypen zu Interferenzen kommt, welche das Leistungsverhalten empfindlich beeinträchtigen können. Behinderungen zeigen sich sowohl für Datenobjekte im Rahmen der Synchronisation (*data contention*) als auch für allgemeine Betriebsmittel wie CPU, Platten-, Hauptspeicher, um die konkurriert wird (*resource contention*). So sind etwa zahlreiche Sperrkonflikte vorprogrammiert, wenn komplexe Abfragen lange Lesesperren halten, die parallel vorzunehmende Änderungen blockieren und somit den OLTP-Durchsatz reduzieren.

Eine Abhilfe hierfür besteht in der Nutzung eines sogenannten Mehrversionen-Sperrverfahrens, bei dem für geänderte Objekte gegebenenfalls mehrere Objektversionen geführt werden. So wird den parallelen Lesetransaktionen stets ohne Sperrkonflikte (Wartezeiten) eine gültige Version bereitgestellt (Rahm 1994). Einige kommerzielle Datenbanksysteme unterstützen bereits einen solchen Ansatz.

Problematischer ist die Behandlung von Resource-Contention aufgrund des hohen Ressourcenbedarfs komplexer Abfragen, der zu starken Behinderungen gleichzeitig aktiver Transaktionen führen kann. Hier besteht im wesentlichen nur die Möglichkeit, durch geeignete Scheduling-Verfahren die Behinderungen zu kontrollieren (zum Beispiel durch Vergabe von Transaktionsprioritäten).

In Parallelen Datenbanksystemen liegt eine weitere Schwierigkeit darin, eine Datenverteilung zu finden, die sowohl eine effektive Parallelverarbeitung für komplexe Anfragen als auch eine möglichst rechnerlokale Bearbeitung von OLTP-Aktionen zur Begrenzung des Kommunikationsaufwandes ermöglicht.

Dynamische Parallelisierung

Zur Reduzierung von Resource-Contention stellt sich darüber hinaus die Notwendigkeit dynamischer Parallelisierungsstrategien, welche den Parallelitätsgrad vom aktuellen Systemzustand (beispielsweise CPU-Auslastung, Hauptspeicherverfügbarkeit) abhängig machen. Weiterhin ist eine dynamische Lastbalancierung erforderlich, um eine möglichst gleichmäßige Rechnerauslastung zu erreichen. Dazu sollten Transaktionen oder einzelne Teilanfragen möglichst Rechnern mit geringer Auslastung zugewiesen werden. Für solche Kontrollaufgaben bestehen - in Abhängigkeit der jeweiligen Architektur (Shared-Nothing, Shared-Disk oder Shared-Everything) – zahlreiche Verfahrensweisen, die Gegenstand der Forschung sind. Diese Fragestellungen werden unter anderem auch an der Universität Leipzig unter Leitung des Autors im Rahmen eines Forschungsprojektes untersucht.

Literatur

DeWitt, D.J., Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems. Comm. ACM 35 (6), 85-98, 1992

Rahm, E.: Hochleistungs-Transaktionssysteme. Vieweg-Verlag, 1993

Rahm, E.: Mehrrechner-Datenbanksysteme – Grundlagen der verteilten und parallelen Datenbankverarbeitung. Addison-Wesley-Verlag, 1994

VERMEIDUNG VON SPERRKONFLIKTEN UND DYNAMISCHE LASTVERTEILUNG