Ein Simulationsansatz zur Bewertung paralleler Shared-Disk-Datenbanksysteme

Thomas Stöhr, Universität Leipzig; E-Mail: stoehr@informatik.uni-leipzig.de

1 Einleitung

Parallele Datenbanksysteme haben sich als einer der Schlüssel zur Hochleistungs-Transaktions- und Datenbankverarbeitung herauskristallisiert [1,2]. Sie nutzen die Kapazität einer Vielzahl gekoppelter Verarbeitungsknoten, um einerseits ein leistungsfähiges Online-Transaction-Processing (z.B. hohe Durchsatzraten für Kontenbuchungs-Transaktionen) zu gewährleisten, andererseits aber auch die Antwortzeit komplexer Adhoc-Anfragen, wie sie heute z.B. für Decision-Support-Systeme von großer Wichtigkeit sind, zu minimieren.
Parallele Datenbanksysteme basieren auf den drei Hauptarchitekturen Shared-Everything (SE), Shared-Nothing (SN) und Shared-Disk (SD) [1,3]. Diese drei Architketuren unterscheiden sich hauptsächlich in der Kopplung ihrer Ressourcen. In SE-Systemen teilen sich alle Prozessoren einen gemeinsamen Speicher und die Peripheriegeräte, während bei SN-Systemen jeder Knoten über getrennte Prozessoren-, Hauptspeicher- und Plattenkapazität verfügt. Somit bestimmt in SN-Systemen die Art der Verteilung der Daten zumindest für Basisoperatoren (z.B. relationale Scan-Anfragen) auch den Ort der Verarbeitung der (Teil-)Transaktionen, die diese Daten benötigen. In Systemen vom Typ SD besitzt jeder Knoten Zugriff zur gesamten Datenbank, d.h. auf alle Speichermedien, auf denen die Daten allokiert sind. Damit kann jede (Teil-)Transaktion auf jedem Knoten verarbeitet werden. Neben der größeren Skalierbarkeit besitzen SD-Systeme gegenüber SE-Systemen aufgrund der losen Kopplung aller Prozessoren ein größeres Maß an Fehlertoleranz. Gegenüber SN-Systemen bieten SD-Systeme speziell Freiheitsgrade im Bereich Lastbalancierung und Datenallokation, da der Ausführungsort von Transaktionen und Queries nicht durch die Datenverteilung festgelegt ist, sondern dynamisch gewählt und mit der Allokationsstrategie abgestimmt werden kann.
Bisher konzentrierten sich Forschungsaktivitäten weitestgehend auf SE- [4,5] bzw. SN-Systeme [1, 6, 7]. Aufgrund des aber offensichtlich vorhandenen großen Potentials paralleler SD-Systeme [8, 9] ist hier eine Lücke zu schließen. Zur ausführlichen Bewertung dieser Systeme wird daher an der Universität Leipzig ein umfangreiches Simulationssystem entwikkelt, welches für verschiedene Lasttypen eine ausführliche Bewertung des Systemverhaltens bei der Verarbeitung dieser Lasten vornehmen kann. Erste Simulationsergebnisse zeigen, daß durch Variation des Parallelitätsgrades für relationale Scan-Operationen (mit und ohne Indexunterstützung), wie bei SD-Systemen möglich, bessere Antwortzeiten für komplexe Queries bzw. ein höherer Durchsatz für OLTP-Lasten erzielt werden können [9]. Diese Untersuchungen zeigen auch, daß den E/A-Kosten in der DB-Verarbeitung eine besondere Rolle zukommt, da diese einen hohen Anteil an den Gesamtkosten der Transaktionsverarbeitung darstellen. Somit spielen bei der Verarbeitung großer Datenmengen Datenverteilungs- und -allokationsaspekte eine wichtige Rolle. In diesem Papier zeigen wir nun auf, inwiefern die Wahl der Datenallokationsstrategie die Performanz paralleler Scan-Operatoren beeinflußt.
In Kapitel 2 geben wir zunächst einen Überblick über das entwickelte Simulationssystem. Erste Simulationsergebnisse, die die Notwendigkeit einer mit der Anfrageverarbeitung abgestimmten Datenallokationsstrategie aufzeigen, liefert Kapitel 3.

2 Überblick über das Simulationssystem

Bild 1 gibt einen Überblick über den Aufbau des Simulationssystems. Die einzelnen Module des derzeit existierenden Systems wurden mit Hilfe der diskreten, ereignisgesteuerten Simulationssprache DeNet [10] entwickelt. Diese basiert auf dem *Koroutinen-Konzept* von Modula-2, d.h. eine simulierte Verarbeitungseinheit (z.B. Transaktions-Manager, Sperrverwalter, Pufferverwalter, ...) wird jeweils als eine Koroutine implementiert, wobei die einzelnen Einheiten den Verarbeitungsfluß über das Auslösen / Empfangen von Ereignissen mit assoziierten Datenstrukturen realisieren. Handlungen, die durch solche Ereignisse ausgelöst werden (wie z.B. die Verarbeitung von CPU-Instruktionen) lassen die Simulationszeit fortschreiten, deren maximale Dauer für jeden Simulationslauf per Parameter festgelegt wird. Die Parameter der einzelnen Routinen sowie die Gesamttopologie des Systems werden in einer speziellen Beschreibungssprache formuliert und in Dateien abgelegt, die vom Laufzeitsystem vor Beginn der Simulation eingelesen werden. Am Ende eines Simulationslaufes werden umfangreiche Statistiken pro Modul ausgegeben, z.B. mittlere Antwortzeit der Transaktionen, mittlere Warteschlangenlänge der CPUs, mittlere Plattenauslastungen usw. Diese statistischen Daten können auch schon zur Laufzeit über ein Tool visualisiert werden.

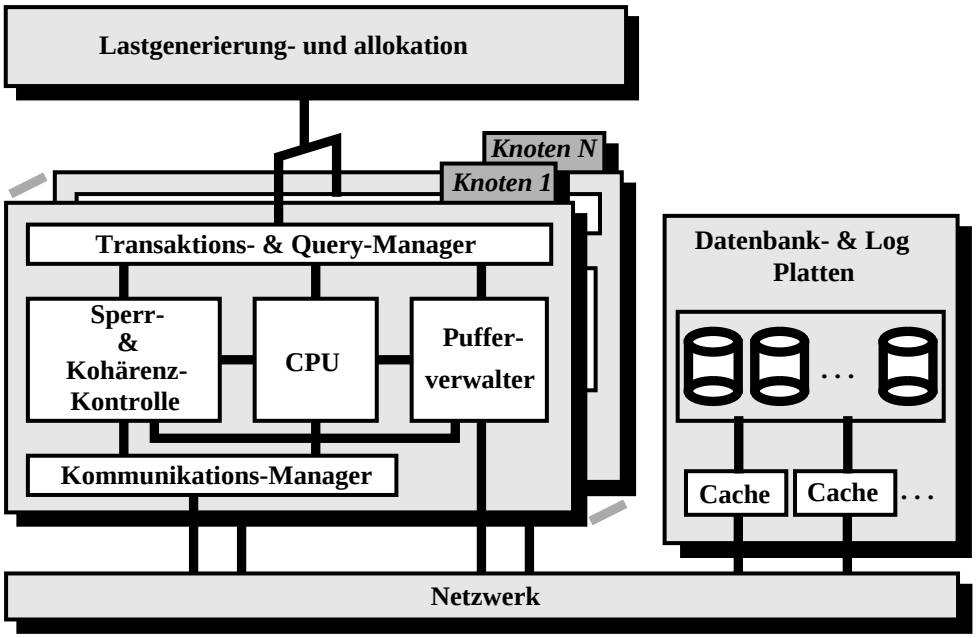


Bild 1 Überblick über das Simulationssystem

2.1 Verarbeitungsmodell

Vereinfacht dargestellt wird die Transaktionsverarbeitung unseres z.Z. auf die Shared-Disk-Architektur beschränkten Systems folgendermaßen nachgebildet:
Wir simulieren Transaktionen, die z.Z. genau einen Scan-Operator repräsentieren (s. Abschn.

2.2.) als eine durch eine Begin-of-Transaction-(BOT-)/End-of-Transaction-(EOT-)Durchführung eingeklammerten Baum von Seitenreferenzen, wobei ein ‘Ast’, also eine Liste von Seitenreferenzen, das Verarbeitungsvolumen einer Sub-Transaktion darstellt. Nach Durchführung von BOT und dem Allokieren der Sub-Transaktionen auf den vorgesehenen Rechnern (Verteilung / Parallelisierung wird über Matrizen parametrisch festgelegt) werden die Seitenreferenzen durchgeführt. Jede Seitenverarbeitung wird durch Anforderung einer Sperre, Bereitstellen der Seite im Puffer (wodurch ggf. eine E/A ausgelöst wird) und CPU-Zeit zur Verarbeitung der Seite im Puffer simuliert. Durch ein (ggf. leseoptimiertes) 2-Phasen-Commit-Protokoll wird die Transaktionsverarbeitung abgeschlossen.

2.2 Lastmodell

Unser Simulationssystem ist in der Lage, verschiedene Lastarten nachzubilden. So existieren z. Z. Lastgeneratoren für Kontenbuchungs-Transaktionen (Debit-Credit), die Simulation eines realen DB-Trace mit mehreren Lese- und Änderungs-Transaktionstypen sowie ein Generator für synthetische, relationale Scan-Lasten. Bei letzterem können mehrere Transaktions-Typen mit per Parameter festzulegender Ankunftsrate (offenes Warteschlangenmodell) simuliert werden, die über die unterschiedlichsten Zugriffscharakteristika (Lese-/Änderungszugriffshäufigkeit) verfügen. So können Relationen-Scans (Lesen einer kompletten Relation ohne Indexunterstützung) sowie Index-Scans (Lesen/Ändern einer Relation mit Indexunterstützung, wobei die Seiten physisch geclustert oder nicht-geclustert sein können) untersucht werden, wobei der Verteilungs- und Parallelisierungsgrad sowie der Allokationsort der (Teil-)Anfragen parametrisch eingestellt werden kann. Damit ist es möglich, verschiedenen Anfrageklassen verschiedene Knoten zuzuordnen, um z. B. den CPU-Wettbewerb zu vermindern. Jede Transaktion repräsentiert dabei genau einen der o. g. Operator-Typen. Es werden Inter-Transaktionsparallelität (Mehrbenutzerbetrieb) sowie Intra-Operatorparallelität (parallele Durchführung eines Operators auf mehreren Knoten) unterstützt.
Zur Abwicklung der notwendigen Kommunikationsvorgänge (Starten von Teilanfragen, Zugriff zum E/A-System etc.) simulieren wir ein Hochleistungsnetzwerk, um diesbezügliche Engpässe (zunächst) auszuschließen.

2.3 E/A- und Datenallokationsmodell

Das E/A-System ist als Plattenfarm organisiert, wobei die Kontroller als Server implementiert sind (jeder Kontroller kann jede Platten bedienen). Das System unterstützt das Lesen mehrerer physisch benachbarter Seiten bei nur einem Plattenzugriff (prefetching), was besonders für einen Relationen-Scan eine Einsparung von Positionierungs-Overhead durch Lesen mehrerer Blöcke in einem Lesevorgang und Speichern im Platten-Cache (zur Verbesserung der Trefferraten) bringt.
Sämtliche relevante Daten werden in *Relationen* organisiert, wobei Fragmente (*Partitionen*) das Allokationsgranulat auf einer Platte darstellen. Partitionen werden in *Seiten* unterteilt, die ihrerseits die einzelnen *Tupel* einer Relation enthalten. Dabei enthält jede Seite gleich viele, über einen Blockungsfaktor definierte Anzahl Tupel. Für jede Relation können ein geclusterter und/oder beliebig viele nicht-geclusterte Indizes definiert werden. Diese Indizes werden in eigenen Partitionen, getrennt von den eigentlichen Daten, ebenfalls verteilt gespeichert, um nicht einzelne Platten, die z.B. die Blattseiten eines Index enthalten würden, zum Engpaß werden zu lassen (Wurzel und innere Knoten sind zur Laufzeit i. d. R. im Platten-Cache oder im DB-Puffer vorhanden).

Zur Unterstützung paralleler Anfrageverarbeitung können Relationen und Indizes auf mehrere Platten verteilt werden. Sog. *physische* Verfahren arbeiten dabei auf (Mengen von) Blöcken, die in Round-Robin-Manier auf die Platten verteilt werden (*striping*); dabei ist ein physischer Block als Allokationsort für eine logische DB-Seite zu sehen. Die Allokation findet also außerhalb des DBS statt (z. B. in herkömmlichen Disk-Arrays [11]), so daß der Anfrageoptimierer keine Anhaltspunkte bzgl. Datenallokation zur Parallelisierung von Anfragen zur Verfügung hat, um Plattenwettbewerb zu vermeiden. Eine Verbesserung stellt das ebenfalls simulierte *manual striping* [12] dar, bei dem der DB-Administrator die Größe von Partitionen sowie die Platte, auf die diese allokiert werden soll, bestimmt. Beim Laden der Relation werden alle Tupel mit hochzählender Tupel-ID konsekutiv Partition für Partition gespeichert. Somit sind dem DBS Verteilungsgrad (Anzahl der Platten, auf denen die Relation allokiert ist) und -granulat (Größe eines Fragments auf einer Platte) sowie die Verteilung der Tupel-IDs einer Relation bekannt.
Bei sog. *logischen* Datenverteilungsstrategien werden die Tupel einer Relation nach einem oder mehreren Partitionierungsattributen entweder auf alle (full declustering) oder nur auf einen Teil (partial declustering) der Platten verteilt. Unser System unterstützt zur Zeit eine *Bereichspartitionierung*, also eine Aufteilung einer Relation nach einem Attribut in ebenso viele Partitionen, wie Platten existieren (Beschränkung der zu referenzierenden Platten speziell bei Punkt- oder Bereichsanfragen). Dabei kann auch die realistische evtl. Ungleichverteilung (Skew) eines Attributwertes berücksichtigt werden. Bei der Anwendung von logischen, also DBS-initiierten Datenverteilungs-Strategien ist somit der Allokationsort eines Tupels bzgl. des Wertes des Partitionierungsattributes (bzw. der Bereiche von Tupeln) dem DBS bekannt.

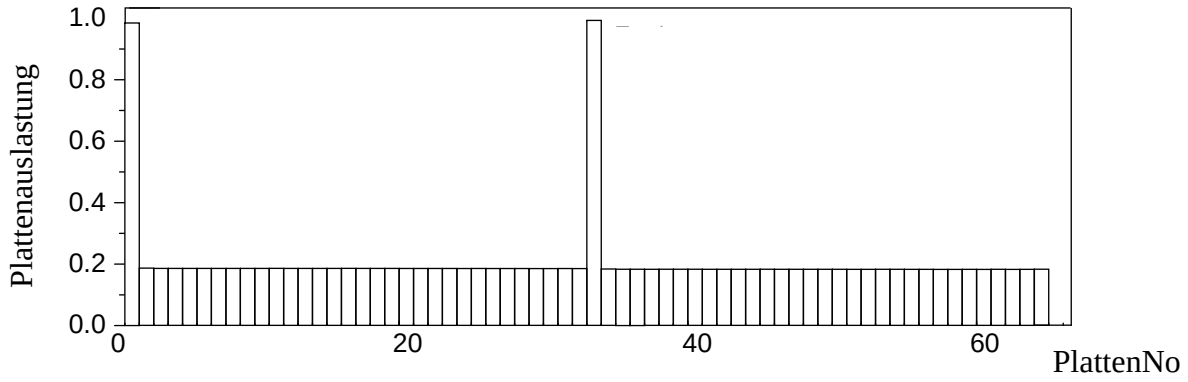
2.4 Sperrverwaltung

Als Sperrverfahren ist ein verteilter, hierarchischer Primary-Copy-Locking-Algorithmus implementiert. Bei diesem Verfahren übernimmt jeder der vorhandenen Knoten die Verantwortung über einen Teil der zu sperrenden Objekte (Relationen, Seiten, Tupel), wobei alle Sperranforderungen bzgl. eines Objektes an den jeweiligen Koordinator gehen müssen. Zur Kohärenzkontrolle ist ein Verfahren implementiert, welches die Überprüfung der Gültigkeit von Daten- und Indexseiten sowie die Bereitstellung der aktuellsten Version einer Seite im anfordernden Rechner in das Sperrverfahren integriert und somit den Nachrichtenaufwand minimiert.

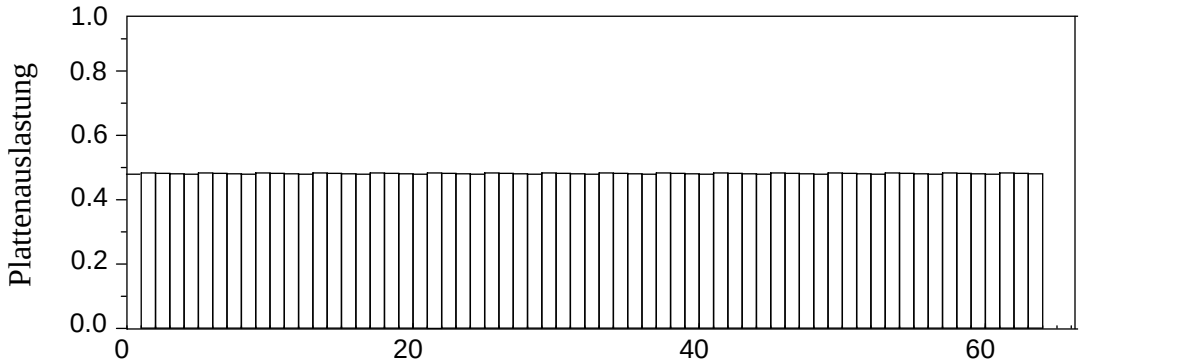
3 Erste Ergebnisse

Gegenstand der ersten Untersuchungen sind relationale Scan-Lasten mit und ohne Indexunterstützung. In dem nun beschriebenen Experiment betrachten wir die Verarbeitung von Relationen-Scans auf einer gemeinsamen Relation im Mehrbenutzerbetrieb, um für diese E/A-intensiven Lasten die Abhängigkeit der Antwortzeit von der Allokationsstrategie zu beobachten. Wir gehen von einem Verarbeitungsparallelitätsgrad (*VPG*) von 16 und einer gleichmäßigen Verteilung der ca. 1,2 Mio. Tupel (entspricht *anzsei* = ca. 120.000 Seiten) der betrachteten Relation auf 64 Platten (*anzplt*) aus. Jede Teiltransaktion verarbeitet die gleiche Anzahl Seiten $\frac{anzsei}{VPG}$. Betrachtet werden drei verschiedene Verteilungsgranulate (*VG*): 8, 32, *max* ($=\frac{anzsei}{anzplt}$) Blöcke. Zunächst wird *manual striping* als Allokationsstrategie unterstellt. Es werden bei jedem E/A-Vorgang 8 physisch benachbarte Blöcke gelesen (prefetching), um Positionierungs-

Overhead zu sparen und bessere Cache-Trefferraten zu erzielen. Die Ankunftsrate beträgt 1.12 TPS, welches eine mittlere Plattenauslastung von ca. 50 % erzeugt.
Bild 2 zeigt die Plattenauslastungsverteilung bei Wahl von *VG*=8 bzw. *VG*=32. Man erkennt



(i) *VG* = 8 Seiten



(ii) *VG* = 32 Seiten

Bild 2 Plattenauslastungen für verschiedene Verteilungsgranulate

für *VG*=8 **(i)** eine starke Überlastung zweier Platten und eine ausgeglichene, allerdings sehr niedrige Auslastung der übrigen Platten (ca. 20 %). Im Gegensatz dazu liegen bei einem *VG* von 32 **(ii)** die Plattenauslastungen für alle Platten bei ca. 50 %. Die Größe $\frac{anzsei}{VG \bullet VPG}$ bestimmt bei konsekutiver Abspeicherung der Seiten den “Abstand” zwischen denjenigen zwei Platten, die durch zwei Teilanfragen, deren zu verarbeitenden Bereiche konsekutiv aufeinanderfolgen, als erstes referenziert werden. Daraus ergeben sich für alle Teiltransaktionen derselben Transaktion eine unterschiedliche Menge von verschiedenen ”E/A-Einstiegsplatten”, je nach Wahl von *VPG* und *VG*. In unserem Beispiel ergeben sich für ein *VG* von 8 gerade 2, für ein *VG* von 32 aber 8 solcher Platten. Dadurch wird für *VG*=32 die E/A-Last bei weitem besser balanciert. Dies führt dazu, daß im beobachteten Simulationszeitraum nahezu alle Transaktionen mit vertretbarer Antwortzeit beendet werden konnten, während bei einer Wahl *VG*=8 praktisch keine Transaktionen zum Ende kamen. In diesem Fall spiegelt sich der Skew in der Plattenauslastung

in einem Skew in der Antwortzeit der einzelnen Teiltransaktionen wieder.
Um diesen Skew möglichst gering zu halten, sollte zur Abstimmung zwischen *VG* und *VPG* ein *VG* pro Relation gewählt werden, welcher die maximale Anzahl "Einstiegsplatten" gewährleistet, um eine ausgewogenere Balancierung der E/A-Last zu erreichen. Genauso ist die Wahl eines entsprechend niedrigeren oder höheren *VPG* denkbar bzw. die Nutzung einer logischen (für den DB-Administrator allerdings komplexer zu bestimmenden) Datenverteilungsstrategie.
Weitere Simulationsläufe, deren Ergebnisse nicht graphisch dargestellt sind, simulierten die Verwendung von *VG=max* ($=\frac{anzsei}{anzplt}$) bzw. eine dem DBS nicht bekannte Verteilung, welche durch Random-Plattenzugriffe simuliert wurden. *VG=max* (dies entspräche bei dieser homogenen Last vom Ergebnis her auch einer Bereichsfragmentierung mit Gleichverteilung der Attributwerte) liefert ein dem Fall *VG*=32 vergleichbar gleichmäßiges Plattenauslastungsverhalten und führt in diesem Beispiel auch zu den besten Antwortzeiten (wg. gänzlich fehlender Platten-Wartebeziehungen zwischen Teiltransaktionen derselben Transaktion), während die "übliche" Strategie, die Datenverteilung durch das (parallele) File-System, also außerhalb des DBS, durchführen zu lassen, zwar gegenüber *VG*=8 gleichmäßigere, aber recht hohe Plattenauslastungen mit wesentlich schlechteren Antwortzeiten als für *VG*=32 bzw. *VG=max* bringt. Durch den quasi-zufallsgesteuerten Plattenzugriff kommt es zu großer E/A-Konkurrenz innerhalb einer Transaktion.
Wählt man beim in unseren Experimenten eigentlich optimalen *VG=max* eine höhere Ankunftsrate (**Bild 3**), so ist auch für diesen Fall eine (zeitweilige) Ungleichverteilung der

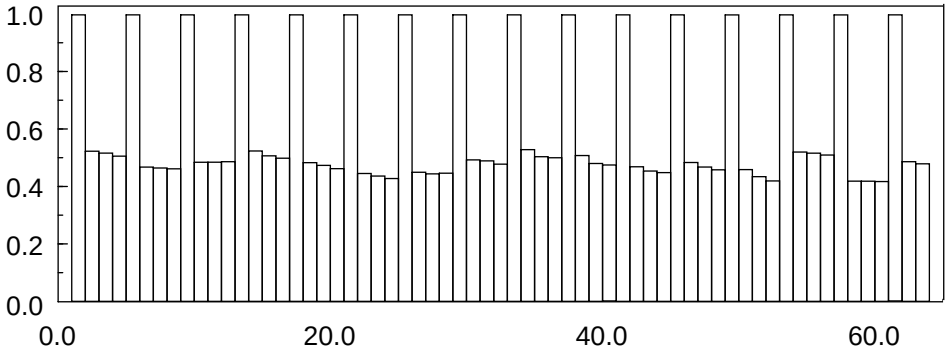


Bild 3 Plattenauslastungen für *VG = max* bei höherer Ankunftsrate

Plattenauslastung innerhalb der Teilmenge der für eine Teiltransaktion zu referenzierenden Platten zu beobachten. Dies ist damit zu begründen, daß durch das große Allokationsgranulat für einen relativ langen Zeitraum zunächst nur 16 (entsprechend dem angenommenen Parallelitätsgrad) der 64 Platten referenziert werden, dann die nächsten 16 usw. Gerade im Mehrbenutzerbetrieb kann dies zu Überlastungen einzelner Platten führen, falls während solcher ungleichen Plattenauslastungen neue Transaktionen eintreffen, die wiederum diese Platten zuerst referenzieren usw. Somit wäre ein kleineres *VG* anzustreben, um eine bessere Balancierung zu erreichen.

4 Zusammenfassung und Ausblick

Die Anbindung aller Knoten an die vollständige Datenbank in einem parallelen SD-Datenbanksystem bietet eine große Flexibilität in der Allokation von (Teil-)Anfragen und -Transaktionen auf vorhandenen Knoten. Allerdings birgt die Nutzung von Intra-Operatorparallelität im Falle nicht abgestimmter Last- oder Datenallokationsstrategien die Gefahr von Plattenengpässen. Somit müssen zur Ausnutzung dieser Möglichkeiten Datenallokations- und Lastbalancierungsalgorithmen entwickelt werden, die aufeinander abgestimmt sind, um optimales Antwortzeit- und Durchsatzverhalten zu erreichen.
Es werden Datenallokationsstrategien in das Simulationssystem eingebracht, die die Ergebnisse eines z. Z. in Entwicklung befindlichen analytischen Modells zur parallelen Anfrageverarbeitung im SD-Kontext umsetzt. Damit sollen das Antwortzeit- und Durchsatzverhalten paralleler, relationaler Scan- (und später auch Join-) Lasten in Abhängigkeit von Parallelisierungsgrad, Datenverteilungsgrad und -granulat untersucht werden. Ziel ist es, ein Allokations-Tool zu entwickeln, das einem Datenbank-Administrator mittels weniger Parameter erlaubt, eine auf die zu erwartende Datenbank-Last abgestimmte Datenallokationsstrategie zu entwerfen, die einerseits dynamische Änderungen des Verarbeitungsparallelitätsgrades (abhängig von der Lastsituation) unterstützt und andererseits bei einem sich stark ändernden Lastprofil kostengünstige Umverteilungsstrategien anbietet. Das für diese Untersuchungen entwickelte Simulationssystem befindet sich im Prozeß ständiger Erweiterung und Verfeinerung. Laufende und geplante Arbeiten beschäftigen sich mit der Konvertierung des Programmpakets in die prozeß-orientierte Simulations-Library csim, Version 18 [13]. Dabei werden wir die Funktionalität an vielen Stellen erweitern, um es z.B. zu ermöglichen, verschiedene DBS-Architekturen auf unterschiedlichen Hardware-Architekturen zu simulieren, z.B. ein SD-DBS auf einer SN-Hardware bzw. hybride Architekturen (z.B. SE-Knoten in einem SD-Cluster) oder neue Architketurformen (z.B. NUMA) zu untersuchen.
In weiteren Arbeiten werden wir neue Funktionalitäten integrieren: einen Lastgenerator für relationale Join-Anfragen, einen TPC-D-Benchmark-Lastgenerator, eine Hardware-Schicht, die Disk-Array-Technologie simuliert und nicht zuletzt eine Abbildungsschicht, die es uns ermöglicht, die Verarbeitung paralleler DBS auf unterschiedlichen Hardware-Architekturen zu untersuchen. Ein weiterer Punkt wird die Verbesserung einzelner Teilmodelle sein (exaktere Platten- und Netzwerkmodellierung). Außerdem wird z. Z. an der Erstellung einer graphischen Benutzeroberfläche zur Generierung, Überwachung und automatischen Auswertung der verteilten Simulationsläufe gearbeitet.

5 Literatur

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

[2] Valduriez, P.: Parallel Database Systems: Open Problems and New Issues. Distributed and Parallel Databases 1(2), 137-165, 1993

[3] Bhide, A.: An Analysis of Three Transaction Processing Architectures. Proc. 14th Int. Conf. on Very Large Databases, Long Beach, CA, 339-350, 1988

[4] Stonebraker, M.; Katz, R.; Patterson, D.; Ousterhout, J.: The Design of XPRS. Proc. 14th

Int. Conf. on Very Large Databases, Long Beach, CA, 318-330, 1988
[5] Graefe, G.: Encapsulation of Parallelism in the Volcano Query Processing System. Proc. ACM SIGMOD Conf., 102-111, 1990
[6] Özsu, M.T.; Valduriez, P.: Principles of Distributed Database Systems. Prentice Hall, 1991
[7] Stonebraker, M.: The Case for Shared Nothing. IEEE Database Engineering 9 (1), 4-9, 1986
[8] Rahm, E.: Parallel Query Processing in Shared Disk Database Systems. Proc. 5th Int. Workshop on High Performance Transaction Systems (HPTS-5), Asilomar, Sep. 1993 (Extended Abstract: ACM SIGMOD Record 22 (4), Dec. 1993)
[9] Rahm, E.; Stöhr, T.: Analysis of Parallel Scan Processing in Shared Disk Database Systems. Proc. EURO-PAR, Stockholm, Springer-Verlag, LNCS, 1995
[10] Livny, M.: DeNet Users's Guide, Version 1.5, Computer Science Department, University of Wisconsin, Madison, 1989.
[11] Patterson, D. A., Gibson, G., Katz, R. H.: A Case for Redundant Arrays of Inexpensive Disks (RAID). Proc. ACM SIGMOD Conf., 109-116, 1988
[12] Oracle 7 Parallel Server: Concepts & Administration, Release 7.3, Oracle Corporation, 1996
[13] Schwetman, H.: Csim18 Simulation Engine: User's Guide. Mesquite Software Inc., Austin, TX, 1996