Algorithmen zur effizienten Lastkontrolle in Mehrrechner-Datenbanksystemen

Erhard Rahm*

Stichworte: *Lastkontrolle, Mehrrechner-Datenbanksysteme, DB-Sharing, DB-Distribution, Transaktionssysteme*

Zusammenfassung: *In einem Mehrrechner-Datenbanksystem ist die Lastkontrolle für die Verteilung der aktuellen Transaktionslast auf die verfügbaren Rechner verantwortlich. Es sind dabei kurze Antwortzeiten einerseits sowie eine gleichmäßige Systemauslastung und hoher Durchsatz andererseits anzustreben. Nach einer Einführung wird die Funktion und Realisierung der Lastkontrolle in einem Mehrrechner-Datenbanksystem diskutiert. Im Hauptteil der Arbeit werden mehrere Verfahren zur Erstellung einer sogenannten Routing-Tabelle vorgestellt, mit deren Hilfe die Transaktionslast dynamisch verteilt wird.*

Algorithms for Efficient Load Control in Multiprocessor Database Systems

Key-words: *load control, multiprocessor database systems, DB-Sharing, DB-Distribution, transaction processing systems*

Abstract: *In multiprocessor database systems load control is responsible for distributing the current transaction load among the set of available processors. This transaction routing must ensure short response times as well as high throughput without overloading any processor. After an introduction we discuss the function and realization of load control in more detail. The main part of the paper describes several algorithms for constructing a so-called routing table required for a table-driven transaction routing.*

1 Einführung

An zukünftige Hochleistungs-Datenbanksysteme (HLDBS) werden hohe Anforderungen bezüglich Durchsatz, Antwortzeiten, Verfügbarkeit, modularem Wachstum und Handhabbarkeit gestellt [1]. Nach [2] werden große Anwender (z.B. Fluggesellschaften, Banken) in wenigen Jahren Transaktionsraten von bspw. 1000 Transaktionen (TA) pro Sekunde eines einfachen Typs (z.B. die KONTENBUCHUNG [3]) benötigen. Die Forderung nach modularem Wachstum verlangt ein lineares Durchsatzwachstum beim Hinzufügen eines neuen Rechners unter Beibehaltung der kurzen Antwortzeiten.

Diese hohen Anforderungen können von Datenbankverwaltungssystemen (DBVS) auf Monoprozessoren oder eng gekoppelten Rechnerverbunden sowohl aus Leistungs- als auch aus Verfügbarkeitsgründen nicht erfüllt werden. Wie in [1] ausführlich erläutert, kommen zur Realisierung von HLDBS im wesentlichen zwei Klassen von sogenannten Mehrrechner-Datenbanksystemen (MRDBS) in Frage, nämlich DB-Sharing und DB-Distribution.

DB-Sharing- und DB-Distribution-Systeme bestehen aus eine Menge von autonomen Rechnern, die lose oder nah gekoppelt sind. Jeder der Rechner besitzt einen eigenen Hauptspeicher sowie eine separate Kopie von Betriebssystem und DBVS. In lose gekoppelten Systemen erfolgt die Kommunikation zwischen den Rechnern ausschließlich über Nachrichten, während bei einer nahen Kopplung die Kommunikation für gewisse Teilaufgaben z.B. über ein gemeinsam benutztes Halbleiterspeichersegment erfolgen kann [4]. Der Unterschied zwischen DB-Sharing und DB-Distribution besteht in der Zuordnung der Externspeicher zu den Rechnern.

Bei *DB-Sharing* (Bild 1a) können alle Rechner (DBVS) direkt auf alle Externspeicher und damit auf alle Daten der DB zugreifen. Dies impliziert, daß alle Prozessoren lokal angeordnet (z.B. in einem Raum) sind, erlaubt jedoch auch eine leistungsfähige LAN-Kopplung der Rechner (z.B. 100 MB/Sec). Die von den Terminals gestarteten TA werden über einen TA-Verteiler zur Bearbeitung auf einen der Rechner geleitet (dieses sogenannte TA-Routing gehört zu den Aufgaben der Lastkontrolle; siehe Kap. 2). Dabei kann eine TA im Normalbetrieb stets vollständig innerhalb eines Rechners bearbeitet werden, da ja jeder Prozessor direkt auf die gesamte DB zugreifen kann. Neben der Realisierung der Lastkontrolle [5], gilt es, in solchen Systemen neue technische Lösungen für die Synchronisation der Rechner beim DB-Zugriff [6], [7], für die Behandlung des sogenannten Veralterungsproblems [8] sowie für Logging und Recovery zu finden.

DB-Sharing ist ein relativ neuer Ansatz. Eine erste Implementierung, die allerdings nur zwei Rechner zuläßt, stellt

* Dipl.-Inform. Erhard Rahm, Fachbereich Informatik, Universität Kaiserslautern, Postfach 3049, D-6750 Kaiserslautern

0013-5704/86/4 0161–09 $ 03.00/0

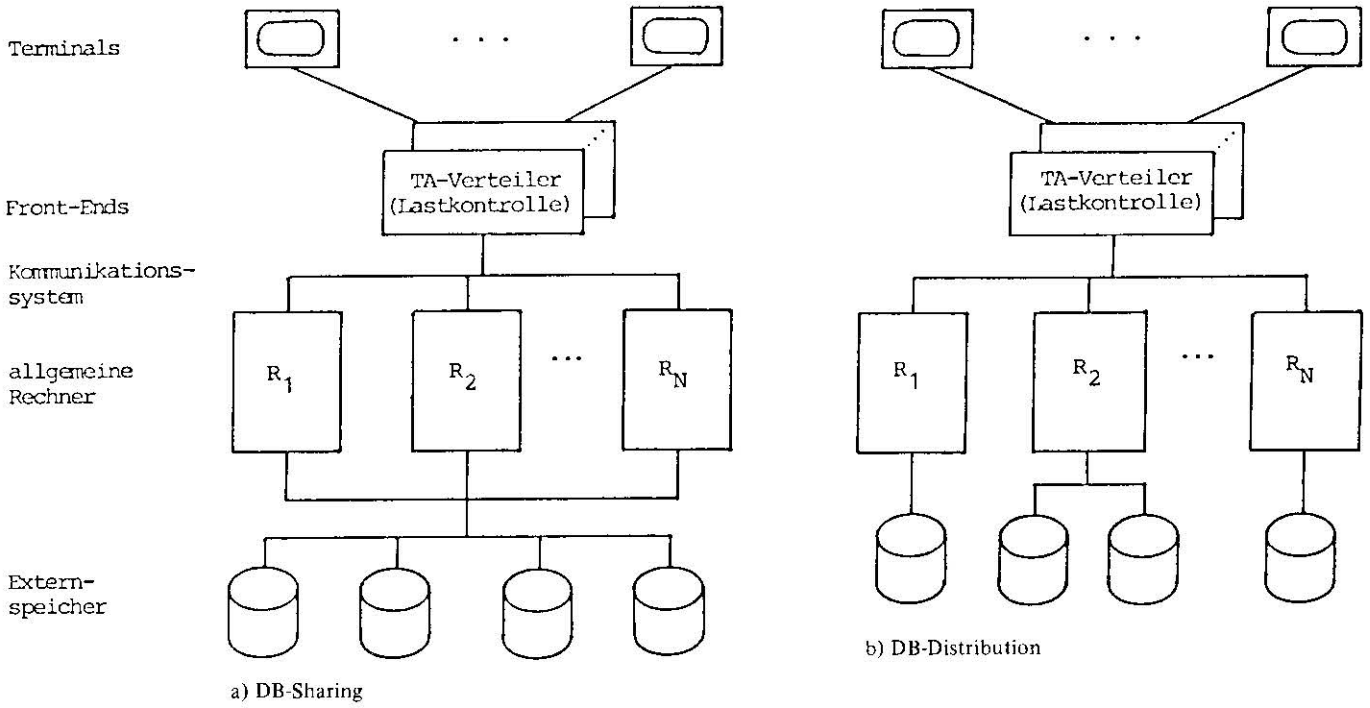

Bild 1 Grobarchitektur von DB-Sharing- und DB-Distribution-Systemen

das sogenannte ‚Data Sharing' für IMS dar. Die Prototyp-Erstellung eines leistungsfähigeren DB-Sharing-Systems wird im AMOEBA-Projekt angestrebt [9].

Bei *DB-Distribution* (Bild 1b) 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 (VDBS) wie etwa R* verfolgt, eignet sich jedoch auch bei lokaler Anordnung der Rechner.

Ein wesentlicher Nachteil von DB-Distribution im Vergleich zu DB-Sharing liegt in der Notwendigkeit, die Datenbank(en) unter den Rechnern aufzuteilen. Denn da eine solche Datenverteilung nur relativ aufwendig änderbar ist (Verlegen von Kabeln, u.ä.), ist das System nur umständlich erweiterbar. Ebenso wie beim Hinzufügen eines Rechners muß auch nach einem Prozessorausfall die Datenverteilung angepaßt werden, da, um die Daten verfügbar zu halten, die Datenpartition des ausgefallenen Rechners durch einen der überlebenden Prozessoren zu übernehmen ist. In diesem Fall kann jedoch die Umverteilung der Daten vereinfacht werden, wenn der Rechner, der die Daten übernehmen soll, von vornherein festgelegt und mit den betroffenen Plattenlaufwerken verbunden wird.

Die relativ statische Datenverteilung erweist sich auch im Normalbetrieb als nachteilig. Denn die Zuordnung der abzuarbeitenden TA kann diese zwar berücksichtigen, jedoch läßt sich die Datenverteilung nur auf ein (durchschnittliches) TA-Profil hin erstellen. Größere Änderungen in der TA-Last, die mehrmals täglich vorkommen können, führen daher i.a. zu Leistungseinbußen.

Ein wesentlicher Vorteil von DB-Distribution liegt in der Tatsache, daß hierbei die Rechner sowohl lokal angeordnet als auch ortsverteilt sein können. Die Ortsverteilung, die bei DB-Sharing wegen der Externspeicheranbindung nicht möglich ist, wird v.a. von vielen Großunternehmen gefordert (Systemleistung vor Ort). Ein weiterer Grund zur Ortsverteilung liegt in der sogenannten Katastrophen-Recovery [2].

Die Vorteile für DB-Sharing ergeben sich weitgehend aus den genannten Nachteilen von DB-Distribution. Bei DB-Sharing ist es vergleichsweise einfach, neue Rechner ins System aufzunehmen oder den Ausfall eines Rechners zu behandeln, da hierbei die Systemstruktur nicht geändert werden muß. Weiterhin ist eine flexiblere Lastkontrolle möglich, und TA können vollständig innerhalb eines Rechners abgearbeitet werden. Schließlich braucht keine Datenverteilung vorgenommen zu werden, so daß Datenbanken aus bestehenden DB-Anwendungen beim Übergang auf ein DB-Sharing-System ohne Modifikation verwendet werden können. Für einen weitergehenden Vergleich zwischen DB-Sharing und DB-Distribution muß aus Platzgründen auf [1], [10] verwiesen werden.

Im nächsten Kapitel wird die Stellung der Lastkontrolle in einem MRDBS untersucht.

2 Lastkontrolle in Mehrrechner-Datenbanksystemen

Um eine effiziente TA-Verarbeitung in einem MRDBS zu erreichen, ist es wichtig, die Anzahl der Interprozessor-Kommunikationen gering zu halten. Für DB-Distribution wird Kommunikation notwendig, wenn zur Abarbeitung einer DB-Operation (DML-Befehl) Datenobjekte benötigt werden, die in der lokalen Datenpartition nicht vorliegen. In der Regel werden dabei nicht nur Daten, sondern gleich die DML-Befehle oder Teile davon (sowie die zugehörigen Antwortnachrichten) zwischen den Rechnern verschickt. Die Abarbeitung solcher DML-Befehle nicht-lokaler TA er-

folgt dann durch eigens erzeugte Sub-TA, die bei TA-Ende in ein rechnerübergreifendes Zweiphasen-Commit-Protokoll einzubeziehen sind, um die Atomizität der Gesamt-TA zu gewährleisten [11]. Bei DB-Sharing ist Kommunikation zwischen den Rechnern im wesentlichen zur Synchronisation der DB-Zugriffe notwendig; die Verarbeitung der DML-Befehle selbst kann in jedem Rechner durchgeführt werden, da alle Daten direkt erreichbar sind.

In verteilten DBS ist wegen der festen Zuordnung von Terminals zu Rechnern sowie der großen Entfernungen eine (dynamische) Lastkontrolle lediglich innerhalb der einzelnen Knoten (i.a. meist nur in Form der sogenannten Queryoptimierung [12]) möglich. Die weiteren Überlegungen konzentrieren sich daher auf *lokal gekoppelte MRDBS* (DB-Distribution- und DB-Sharing-Systeme), in denen eine weitaus flexiblere und leistungsfähigere Lastkontrolle möglich ist.

In lokal gekoppelten MRDBS kann davon ausgegangen werden, wie in Bild 1 angedeutet, daß TA von jedem Terminal aus über einen Kommunikationsrechner (Front-End) zu jedem der Verarbeitungsrechner (VAR) zur Ausführung geleitet werden können. Daher kann in solchen Systemen die Lastkontrolle zur Erfüllung der folgenden Aufgaben vorgesehen werden:

1) TA-Routing
   Von den Terminals gestartete TA sind unter Berücksichtigung der aktuellen Systemauslastung zur Bearbeitung derart auf die verfügbaren Rechner zu leiten, so daß sie möglichst effizient bearbeitet werden können. Damit wird die Lastkontrolle zu einer leistungsbestimmenden Komponente in einem lokal gekoppelten MRDBS.
2) Last-Balancierung
   Die verfügbaren Rechner sollen in etwa gleichmäßig ausgelastet sein; keiner der Rechner darf überlastet werden.
3) Systemüberwachung
   Die globale Lastkontrolle muß sowohl Änderungen in der Last (z.B. stark wechselndes Referenzverhalten) als in der Systemkonfiguration (z.B. Ausfall oder Hinzukommen eines Prozessors) erkennen und geeignet darauf reagieren können. Hierzu steht sie mit den VAR in Verbindung, um die Rechnerauslastung zu kontrollieren oder einen Prozessorausfall erkennen zu können. Nach Ausfall eines Rechners kann die Lastkontrolle auch zur Einleitung und Koordinierung der Recovery vorgesehen werden [13]. Weiterhin eignet sich die Lastkontrolle zur Entgegennahme von manuellen Eingriffen und Kommandos durch den Systemverwalter.

Um eine TA möglichst schnell bearbeiten zu können, sollte das TA-Routing sicherstellen, daß eine TA demjenigen der (nicht überlasteten) Rechner zugeordnet wird, bei dem sie die meisten der voraussichtlich benötigten Betriebsmittel (z.B. Daten, Sperren) vorfindet. Um diese Entscheidung treffen zu können, braucht die Lastkontrolle offenbar folgende Informationen:

1) eine Vorhersage des (voraussichtlichen) Referenzverhaltens einer TA
2) aktuelle Betriebsmittel-Situation und Auslastung der VAR.

zu 1): Diese Vorhersage wird dadurch erleichtert, daß in kommerziellen Transaktionssystemen fast ausschließlich vordefinierte TA-Typen (z.B. Buchungen) ausgeführt werden [14]. Daher kann man das wahrscheinliche Referenzverhalten einer TA i.a. aus den Informationen bestimmen, die bei Ankunft einer TA bekannt sind. Dazu gehören: TA-Typ, Subschema, Terminal-Identifikation und möglicherweise Eingabedaten. Hiermit kann dann – zumindest für die hier interessierenden einfachen TA-Typen – i.a. abgeschätzt werden, welche Datenbank-Bereiche, Satztypen usw. mit welchem maximalen Zugriffsmodus referenziert werden.

zu 2): Das beim TA-Routing anzustrebende Optimierungskriterium ist die Bestimmung des Rechners, der nicht überlastet ist und der eine weitgehend lokale – d.h. mit minimaler Interprozessor-Kommunikation verbundene – TA-Bearbeitung zuläßt. Bei DB-Distribution ist dies derjenige Rechner, zu dessen Datenpartition die meisten der benötigten Daten zählen; bei DB-Sharing derjenige Rechner, der eine weitgehend lokale Synchronisation zuläßt (ein sekundärer Aspekt ist, in welchem Ausmaß benötigte Daten bereits im Systempuffer vorliegen). Wenn das Referenzverhalten einer TA in etwa bekannt ist, dann ist bei DB-Distribution die Auswahl eines geeigneten Rechners vergleichsweise einfach, da die aktuelle Datenverteilung der Lastkontrolle natürlich bekannt ist. Bei DB-Sharing dagegen ist die Entscheidung wesentlich schwieriger und hängt stark vom gewählten Synchronisationsverfahren ab. Jedoch ist allgemein eine Routing-Strategie anzustreben, die TA mit gleichem Referenzverhalten demselben Rechner zuordnet. Die dadurch innerhalb eines Rechners entstehende Lokalität unterstützt dann i.a. sowohl eine lokale Synchronisation als auch eine Reduktion von physischen E/A-Vorgängen, da viele der benötigten Objekte bereits im lokalen Systempuffer vorhanden sind.

Die Auslastung der Rechner ist der Lastkontrolle wenigstens grob bekannt, da sie weiß, welche und wieviele TA sie auf die einzelnen VAR geleitet hat. Weitergehende Informationen können etwa durch periodisches Nachfragen bei den VAR besorgt werden.

Änderungen in der TA-Last können relativ einfach durch Überwachung der Ankunftsraten von TA-Typen erkannt werden, wenn TA-Typen als ‚stabil' angesehen werden (d.h. homogenes Referenzverhalten innerhalb eines TA-Typs). Dies kann durch einen X^2-Test geeignet durchgeführt werden [5], wobei allerdings darauf zu achten ist, daß die sogenannten Sampling-Intervalle geeignet gewählt werden. Ein zu kleines Zeitintervall kann dazu führen, daß auf kleine Änderungen überreagiert wird, während bei einem zu groben Zeitraster selbst deutliche Laständerungen möglicherweise nicht registriert werden.

Da in einem System von beispielsweise 1000 TA/sec die Lastkontrolle sehr häufig das TA-Routing durchführen muß, ist es wichtig, daß die Zuordnung einer TA zu einem Rechner sehr schnell geschehen kann. Denn zum einen könnte sonst die Lastkontrolle leicht zu einem Engpaß werden, und zum anderen schlägt sich ja die für das TA-Routing benötigte Zeit unmittelbar in der Antwortzeit nieder. In [5] wird daher ein tabellengesteuertes TA-Routing mittels einer sogenannten *Routing-Tabelle* vorgeschlagen. In einer solchen Tabelle werden zu jedem TA-Typ ein oder mehrere Rechner genannt, in denen TA dieses Typs ausgeführt werden sollen. Welcher Rechner ausgewählt

wird, hängt dann von der aktuellen Auslastung bei Eintreffen der TA ab.

Bei Erstellung einer solchen Routing-Tabelle wird eine feste Anzahl von Rechnern sowie ein bestimmtes Lastprofil vorausgesetzt. Wenn die einzelnen TA-Typen als homogen angesehen werden, ist das Lastprofil im wesentlichen durch die Ankunftsraten für die TA-Typen festgelegt. Solange sich dieses Lastverhalten oder die Rechnerkonfiguration nicht ändert, kann die berechnete Routing-Tabelle zum TA-Routing verwendet werden. Erst wenn ein neuer Rechner hinzukommt oder der Ausfall eines Rechners oder eine signifikante Änderung im Lastprofil festgestellt wird, ist eine neue Routing-Tabelle zu bestimmen. Da aber erfahrungsgemäß das Referenzverhalten in TA-Systemen über längere Zeiträume (Stunden) homogen ist, braucht die Routing-Tabelle i.a. nur relativ selten geändert zu werden.

Im nächsten Kapitel werden drei Verfahren zur Erstellung einer Routing-Tabelle angegeben.

3 Algorithmen zur Berechnung der Routing-Tabelle

Wie in Kapitel 2 erwähnt, muß die Routing-Tabelle eine Aufteilung der TA-Last auf eine feste Anzahl N von Rechnern festlegen, so daß:

1) alle Rechner in etwa gleich ausgelastet sind und
2) eine weitgehend lokale TA-Verarbeitung gewährleistet ist.

Eine für die Erstellung einer Routing-Tabelle geeignete Beschreibung der TA-Last stellt die sogenannte *Referenzmatrix* dar. In einer solchen Matrix wird für jeden TA-Typ angegeben, wieviele Objektreferenzen (bzw. Sperranforderungen) für jede der in Betracht kommenden DB-Partitionen während eines charakteristischen Zeitraumes aufgetreten sind. Bild 2 zeigt ein Beispiel einer derartigen Referenzmatrix. In dem zugrundeliegenden Zeitraum gingen z.B. 4000 der insgesamt 7000 Objektreferenzen (Sperranforderungen) von TA des TA-Typs T4 auf die DB-Partition P4.

Um eine gleichmäßige Auslastung der Rechner erreichen zu können, muß für jeden TA-Typ bekannt sein, was die Ausführung einer TA dieses Typs im Mittel ‚kostet'. Als Kostenmaß kann man z.B. die mittlere Pfadlänge einer TA oder die mittlere Anzahl referenzierter DB-Objekte verwenden. In dieser Arbeit wird durchweg nach der letztgenannten Möglichkeit verfahren, und zwar derart, daß zur Berechnung der Routing-Tabelle neben der Referenzmatrix ein Vektor als Eingabe zur Verfügung steht, der für jeden TA-Typ die Anzahl der referenzierten Objekte angibt, die während des bei der Referenzmatrix zugrundeliegenden Zeitraums vorgenommen wurden. Durch dieses Kostenmaß wird jedoch nur die reine TA-Verarbeitung berücksichtigt, nicht aber ein möglicherweise anfallender Kommunikationsoverhead oder Last-Umverteilungen durch Verschicken von Aufträgen an andere Rechner.

	P1	P2	P3	P4	P5	P6	Summe
T1	5000	3000		2000			10000
T2	4000	4000				1000	9000
T3		1000	4000		2000	1000	8000
T4			2000	4000		1000	7000
T5	1000			1000	3000	1000	6000
Summe	10000	8000	6000	7000	5000	4000	40000

Bild 2 Fiktive Referenzmatrix

Eine gleichmäßige Rechnerauslastung zu erreichen, bedeutet durch die noch vorzustellenden Algorithmen, die TA-Typen aus der Referenzmatrix so auf die Rechner aufzuteilen, daß jeder Rechner in etwa gleich viele Objektreferenzen zu verarbeiten hat. Die Schwierigkeit entsteht dadurch, daß dabei noch eine weitgehend lokale TA-Verarbeitung erreicht werden soll. Bei DB-Distribution muß hierzu die Datenverteilung beachtet werden; bei DB-Sharing ist eine Abstimmung mit dem verwendeten Synchronisationsverfahren erforderlich.

Für DB-Sharing wird hier nur ein einziges Synchronisationsverfahren betrachtet, nämlich das sogenannte *Primary-Copy-Sperrverfahren* [6], da es eine effektive Kooperation mit der Lastkontrolle erlaubt. Bei diesem Verfahren wird – ähnlich wie bei DB-Distribution – eine Partitionierung der Datenbank vorgenommen, allerdings nur eine logische. Jeder Rechner bekommt hier die Synchronisationsverantwortung für eine Teilmenge dieser Partitionen; man sagt ein Rechner besitzt die Primary Copy Authority (PCA) für seine Partitionen. Eine Sperranforderung für ein Objekt O muß bei diesem Verfahren nun stets an denjenigen Rechner übermittelt werden, der die PCA für O besitzt. Ziel der Lastkontrolle muß es also sein, möglichst viele der Sperranforderungen lokal bearbeitbar zu machen, d.h. eine TA zu dem Rechner zu leiten, der für die meisten der benötigten Objekte die PCA besitzt.

Die Erstellung einer Routing-Tabelle bei diesem Sperrverfahren kann also praktisch wie bei DB-Distribution vorgenommen werden. Der einzige Unterschied, der jedoch für die Algorithmen ohne Bedeutung ist, besteht darin, daß bei Primary Copy Locking (PCL) Sperranforderungen zwischen den Rechnern verschickt werden; bei DB-Distribution dagegen Teile von DML-Befehlen.

Im folgenden werden drei verschiedene Verfahren zur Bestimmung der Routing-Tabelle angegeben. Verfahren 1 berechnet die Routing-Tabelle für DB-Distribution bei vorgegebener Datenverteilung. Verfahren 2 und 3 bestimmen für DB-Distribution bzw. DB-Sharing mit PCL sowohl die Routing-Tabelle als auch die Datenverteilung bzw. die PCA-Verteilung.

3.1 Verfahren 1

Hier soll die Erstellung einer Routing-Tabelle für ein DB-Distribution-System betrachtet werden, für das die Datenverteilung vorgegeben ist. Das Verfahren eignet sich zwar auch für ein DB-Sharing-System mit Primary Copy Locking und vorgegebener PCA-Verteilung, aber da die PCA-Verteilung vergleichsweise einfach umgestellt werden kann [13], wird sie i.a. mit Erstellung einer neuen Routing-Tabelle angepaßt (Verfahren 2 oder 3).

Für DB-Distribution ist jedoch die Routing-Tabelle erheblich einfacher zu ändern als die Datenverteilung. Daher kann man durch Anpassung der Routing-Tabelle zumindest teilweise die fehlende Flexibilität bei der Datenverteilung kompensieren. Eine Neubestimmung der Routing-Tabelle kann z.B. bei stark geändertem Referenzverhalten vorge-

Eingabegrößen:

P Anzahl der Datenpartitionen

N Anzahl der Rechner

M Anzahl der TA-Typen

T Referenzmatrix der Dimension M * P. Dabei gibt T (i, j) die Anzahl der Teil-DML's (Sperranforderungen) an, die von TA-Typ i auf Partition j entfallen

L Vektor der Dimension M, der für jeden TA-Typ die Anzahl der Objektreferenzen angibt, die im Beobachtungszeitraum auf diesen Typ entfielen

D Daten-/PCA-Verteilung (nur in Verfahren 1 Eingabeparameter). Vektor der Dimension P, der für jede DB-Partition angibt, welchem Rechner sie zugeordnet ist

MAX-OR maximale Anzahl der Objektreferenzen, die einem Rechner zugeordnet werden sollen

Ausgabegrößen:

R Routing-Tabelle der Dimension N * M, mit $0 \leq R(i,j) \leq 1$. R (i, j) gibt an, welcher Anteil des TA-Typs j in Rechner i abgearbeitet werden soll. Die Spaltensummen in dieser Matrix müssen den Wert 1 haben.

TL Vektor der Dimension N, der zu jedem Rechner angibt, wieviele der Teil-DML's (aus T) lokal bearbeitbar sind

D Daten-/PCA-Verteilung (nur in Verfahren 2 und 3 Ausgabegröße)

Hilfsstrukturen:

#OR Vektor der Dimension N, der für jeden Rechner die Anzahl der bereits zugeteilten Objektreferenzen enthält.

NEXT-TT Funktion, die den TA-Typ J ermittelt, für den noch die meisten Objektreferenzen aufzuteilen sind

Bild 3 Allgemein verwendete Variablen und Funktionen

nommen werden, vor allem jedoch nach einem Rechnerausfall. Denn da i.a. ein Rechner die Datenpartitionen des ausgefallenen Rechners übernimmt, muß die Routing-Tabelle geändert werden, um eine Überlastung dieses Rechners zu verhindern. Zudem dürfen dem ausgefallenen Rechner ja keine TA mehr zugeordnet werden.

Bei Erstellung einer Routing-Tabelle für DB-Distribution wird eine Referenzmatrix T benutzt, die zu jedem TA-Typ und jeder Partition die Anzahl der Teil-DML-Befehle angibt, die während des Beobachtungszeitraums von TA dieses Typs auf die betreffende DB-Partition entfielen. Es wird dabei der Einfachheit halber angenommen, daß Teil-DML-Befehle stets vollständig innerhalb einer Partition abgewickelt werden können.

Bei Erstellung der Routing-Tabelle werden die TA-Typen ihrer Größe nach, beginnend bei dem größten TA-Typ (gemäß der Anzahl von Objektreferenzen), auf die N Rechner aufgeteilt. Ein TA-Typ, der wegen seiner Größe von mehr als einem Rechner bearbeitet werden soll, wird in mehrere *Pseudo-TA-Typen* unterteilt. Dabei werden für jeden dieser Pseudo-TA-Typen die gleichen Zugriffscharakteristika wie für den gesamten TA-Typ unterstellt.

Um eine in etwa gleiche Auslastung der Rechner zu erreichen, wird die Variable MAX-OR geführt (Bild 3), die angibt, wieviele Objektreferenzen einem Rechner maximal zugeordnet werden sollen. Ein vernünftiger Wert für MAX-OR liegt etwas (z.B. 5 %) über der Gesamtanzahl der Objektreferenzen des Beobachtungszeitraumes dividiert durch die Rechneranzahl N.

Der Algorithmus in Bild 4 ermittelt zu jedem aufzuteilenden TA-Typ J unter Benutzung der Referenzmatrix und der

```
Eingabe:  P, N, M, D, T, L, MAX-OR     (siehe Bild 3)
Ausgabe:  R, TL                        (s. Bild 3)
Hilfsstrukturen: #OR, NEXT-TT          (s. Bild 3), zusätzlich
   #TL  Vektor der Dimension N, der benutzt wird um zu
        einem TA-Typ den Rechner zu bestimmen, auf dem
        bei der vorgegebenen Datenverteilung die meisten
        Teil-DML's lokal bearbeitbar sind.

Algorithmus:
  R, #OR, TL : = 0; /* Initialisierung */
  DO WHILE NOT (alle TA-Typen vollständig aufgeteilt);
     J: = NEXT-TT; /* nächster aufzuteilender TA-Typ */
     L1 : = Anzahl der Objektreferenzen, die für TA-Typ J noch
          aufzuteilen sind;

     /* Bestimmung des geeigneten Rechners */
     DO I = 1 TO N;
        L2 : = max (MAX-OR - #OR (I), L1);
             /* max. Anzahl von Objektreferenzen, die von
                TA-Typ J in Rechner I noch bearbeitet
                werden können*/
        #TL (I) : = 0;
        DO K = 1 TO P;
           IF D (K) = I THEN
              #TL (I) := #TL (I) + T (J,K) * L2 / L (J);
           /* Wenn L2 < L1 kann Rechner I den TA-Typ J
              nicht mehr vollständig aufnehmen. L2 / L(J)
              gibt den Anteil des TA-Typs an, der noch zuge-
              ordnet werden kann */
        END; /* K */
  END; /* I */
  I : = Rechner-Nr, für die #TL maximal ist;

  TL (I) : = TL (I) + #TL (I);
  L2 : = max (MAX-OR - #OR (I), L1);
  #OR (I) = #OR (I) + L2;
  R (I, J) = R (I,J) + L2 / L (J);
END;
```

Bild 4 Algorithmus zu Verfahren 1

Datenverteilung denjenigen Rechner, in dem J weitgehend lokal bearbeitbar ist. Neben der Routing-Tabelle R wird auch berechnet, in welchem Ausmaß eine lokale Bearbeitung zu erwarten ist (Variable TL).

Will man den Algorithmus auf die Referenzmatrix aus Bild 2 anwenden, so muß zuvor noch eine Datenverteilung vorgenommen werden. Wir betrachten hier nur den einfachsten Fall N = 2, wobei Rechner 1 die Partitionen P1, P5 und P6, Rechner 2 die Partitionen P2, P3 und P4 verwalten soll. Es wird weiterhin vereinfachend angenommen, daß jedes Teil-DML der Referenzmatrix nur eine Objektreferenz umfaßt, d.h. der Vektor L ist ebenfalls durch die Referenzmatrix gegeben. Setzt man noch MAX-OR = 21000, dann erhält man folgende Routing-Tabelle:

$$R(1,1) = R(1,2) = 1,\ R(1,3) = R(1,4) = R(1,5) = 0$$
$$R(2,1) = R(2,2) = 0,\ R(2,3) = R(2,4) = R(2,5) = 1.$$

Das heißt also, daß man für die vorgegebene Datenverteilung die TA-Typen T1 und T2 in Rechner 1 und T3, T4 sowie T5 in Rechner 2 abarbeiten soll. Damit sind 10000 der 19000 Objektreferenzen, die Rechner 1 zugeordnet wurden, lokal bearbeitbar. In Rechner 2 können 12000 der 21000 Objektreferenzen lokal bearbeitet werden. Insgesamt erhält man also einen Anteil von 55 %, für den eine lokale Verarbeitung möglich wird.

Es ist natürlich klar, daß durch die Vorgabe einer Datenverteilung nur ein relativ geringes Optimierungspotential bei Erstellung der Routing-Tabelle vorliegt. Dies ist jedoch für DB-Distribution durchaus der Regelfall, da die Datenverteilung, wie bereits mehrfach erwähnt, nur schwer änderbar ist. Mehr Freiheitsgrade erhält man, wenn die Daten- bzw. PCA-Verteilung noch nicht vorgegeben ist.

3.2 Verfahren 2

In diesem sowie dem nächsten Verfahren wird eine Berechnungsmethode für die Routing-Tabelle eines DB-Distribution- bzw. eines DB-Sharing-Systems mit PCL angegeben, bei dem die Datenverteilung (bzw. die PCA-Verteilung) noch nicht vorgegeben ist, sondern ebenfalls berechnet wird. Das bedeutet, daß v.a. aus der Referenzmatrix T und der Rechneranzahl N folgende Angaben zu bestimmen sind:

1) die Routing-Tabelle R,
2) die Datenverteilung (bzw. PCA-Verteilung) D
3) der Vektor TL, der angibt, wieviele Teil-DML's (bzw. Sperranforderungen) lokal bearbeitbar sind.

Hierzu wurde bereits in [5] eine Heuristik vorgeschlagen, die eine Problemlösung in zwei aufeinander aufbauenden Schritten vorsieht. Im ersten Schnritt wird dabei zunächst die Routing-Tabelle erstellt, die dann als Eingabe für Schritt 2 dient, in dem dann die Datenverteilung (PCA-Verteilung) D ermittelt wird. Diese Vorgehensweise wird auch im Algorithmus in Bild 5 verwendet.

```
Eingabe:  P, M, N, T, L, MAX-OR                (s. Bild 3)
Ausgabe:  R, D, TL  (s. Bild 3)
Hilfsstrukturen: #OR, NEXT-TT (s. Bild 3), zusätzlich
   #TL Hilfsvektor der Dimension N zur Bestimmung des
       Rechners, für den zu einer DB-Partition die meisten
       Teil-DML's (Sperranforderungen) lokal bearbeitbar
       sind.

Algorithmus:

 /* Bestimmung der Routing-Tabelle R */
 R, #OR : = 0; /* Initialisierung */
 DO WHILE NOT (alle TA-Typen vollständig aufgeteilt);
    J : = NEXT-TT;
    I : = Rechner-Nr, für die #OR minimal ist;
    L1 : =  Anzahl Objektreferenzen, die für TA-Typ J noch
            aufzuteilen sind;
    L2 : =  max (MAX-OR – #OR (I), L1);
            /* max. Anzahl von Objektreferenzen, die von
            TA-Typ J in Rechner I noch bearbeitet werden
            können */
    #OR (I) : = #OR (I) + L2;
    R (I, J)  : = R (I, J) + L2 / L(J);
 END

 /* Bestimmung der Daten-/PCA-Verteilung D */
 TL : = 0;
 DO K = 1 TO P;
    #TL : = 0;
    DO J = 1 TO M;
       DO I = 1 TO N;
          #TL (I) : = #TL (I) + R (I, J) * T (J, K);
       END;
    END;
    I : = Rechner-Nr, für die #TL (I) maximal;
    TL (I) : = TL (I) + #TL (I);
    D (K) : = I;
 END;
```

Bild 5 Algorithmus zu Verfahren 2

In diesem Algorithmus wird bei Berechnung der Routing-Tabelle ein TA-Typ immer (weitestgehend) demjenigen Rechner zugeordnet, dem bis dahin die wenigsten Objektreferenzen zugeteilt wurden. Bei Bestimmung der Daten- bzw. PCA-Verteilung wird eine Partition demjenigen Rechner zuerkannt, für dessen TA-Last (gemäß der zuvor ermittelten Routing-Tabelle) die meisten Teil-DML's (Sperranforderungen) lokal bearbeitbar sind.

Wendet man den Algorithmus wie bei Verfahren 1 auf die Referenzmatrix in Bild 2 an, so ergibt sich für N = 2 folgende Routing-Tabelle:

R (1,1) = R (1,4) = 1, R (1,5) = 0.33, R (1,2) = R (1,3) = 0
R (2,1) = R (2,4) = 0, R (2,5) = 0.67, R (2,2) = R (2,3) = 1.

Man sieht, daß hier TA-Typ T5 in zwei Pseudo-TA-Typen gesplittet werden mußte, da er nicht mehr vollständig Rechner 2 zugeordnet werden konnte, ohne das Limit MAX-OR zu überschreiten. Die Zuordnung sieht also so aus, daß ein Drittel der TA vom Typ T5 in Rechner 1 und zwei Drittel dieser TA in Rechner 2 bearbeitet werden sollen.

Diese Routing-Tabelle bewirkte eine Daten-/PCA-Verteilung bei der Rechner 1 für die Partitionen P1 und P4, Rechner 2 für P2, P3, P5 und P6 verantwortlich ist. Hiermit erhält man dann, daß von den 19000 (21000) Teil-DML-Befehlen (Sperranforderungen), die Rechner 1 (2) zugeordnet wurden, 11667 (15667) lokal bearbeitet werden können. Dies ergibt zusammen einen Anteil von 68.3 % lokaler Verarbeitung, also erheblich mehr als bei der für Verfahren 1 vorgegebenen Datenverteilung.

3.3 Verfahren 3

Verfahren 2 hat den Nachteil, daß die Erstellung der Routing-Tabelle völlig entkoppelt von der Bestimmung der Datenverteilung (PCA-Verteilung) abläuft. Insbesondere kann sich die Strategie zur Erstellung der Routing Tabelle, bei der ein TA-Typ immer dem Rechner zugeordnet wird, dem zu diesem Zeitpunkt die wenigsten Objektreferenzen zugeteilt sind, als nachteilig erweisen. Für die durch Bild 2 beschriebene TA-Last wird beispielsweise mit Verfahren 2 und N = 2 zunächst TA-Typ T1 vollständig Rechner 1 zugeordnet. TA-Typ T2 wird anschließend Rechner 2 zugeteilt, da diesem zu diesem Zeitpunkt weniger (nämlich gar keine) Objektreferenzen als Rechner 1 zugeordnet sind. Sinnvoller wäre jedoch gewesen, beide TA-Typen demselben Rechner zuzuordnen, da sie beide die meisten Zugriffe auf die Partitionen P1 und P2 vornehmen. Die Nichtausnutzung dieses Wissens führt dann in Verfahren 2 dazu, daß ein hoher Anteil der Referenzen auf diesen beiden Partitionen nicht lokal erledigt werden kann.

Hier soll nun ein Verfahren angegeben werden, daß die Routing-Tabelle und die Datenverteilung (PCA-Verteilung) nicht unabhängig voneinander, sondern koordiniert erstellt. Dazu wird bei jeder Zuordnung eines TA-Typs J (bzw. einem Teil davon) nicht nur die Routing-Tabelle, sondern auch die Datenverteilung (PCA-Verteilung) fortentwickelt. Ein TA-Typ wird jetzt nicht mehr einfach dem Rechner zugeordnet, dem bisher am wenigsten zugeteilt wurde; vielmehr wird die Zuordnung für jeden

Rechner ‚durchgespielt'. Hierbei wird für jeden Rechner I die ‚optimale' Weiterentwicklung der Datenverteilung (PCA-Verteilung) bestimmt für den Fall, daß der TA-Typ J dem Rechner I zugewiesen wird. Danach wird ermittelt, für welchen Rechner I die Zuordnung von J die meisten Teil-DML's (Sperranforderungen) lokal bearbeitbar läßt. Diese Zuordnung wird dann in die Routing-Tabelle übernommen; gleichzeitig wird die für den Rechner I weiterentwickelte Datenverteilung (PCA-Verteilung) als neuer Zwischenstand festgehalten, auf dem bei der weiteren Verteilung der TA-Typen aufgesetzt wird.

Der Algorithmus wird der Überschaubarkeit wegen in zwei Teilen dargestellt. Zunächst folgt die Darstellung des Hauptprogramms (Bild 6), danach die der Funktionsprozedur BEST-R (Bild 7), mit der zu einem TA-Typ der geeignete Rechner bestimmt wird.

Wie in Verfahren 1 und 2 werden auch durch den Algorithmus in Bild 6 die TA-Typen ihrer Größe nach geordnet auf die Rechner aufgeteilt (wird durch die Funktion NEXT-TT gewährleistet). Die neu eingeführten Datenstrukturen X, XL und XN werden bei jeder Neuzuordnung eines TA-

Eingabe und Ausgabe sind wie in Verfahren 2.
Zusätzlich werden zur Berechnung folgende Datenstrukturen benutzt:

X Vektor der Dimension P. X gibt die zu einem Zeitpunkt gültige Datenverteilung (PCA-Verteilung) an, die bei jeder Zuordnung eines TA-Typs weiterentwickelt wird. Zu Beginn gilt X = 0 (keine Zuordnung); nach Zuordnung aller TA-Typen stellt X die zu berechnende Datenverteilung (PCA-Verteilung) D dar.

XL, XN zwei Matrizen der Dimension N * P, die zusammen mit X weiterentwickelt werden. XL (i, j) gibt an, wieviele der bereits aufgeteilten Teil-DML's (Sperranrorderungen) von T (i, j) mit dem aktuellen Stand der Routing-Tabelle R und der aktuell gültigen Datenverteilung X lokal bearbeitet werden können.
XN (i, j) gibt an, wieviel nicht lokal bearbeitet werden können.

Y Matrix der Dimension N * P. Y (I,*) entspricht der Weiterentwicklung von X für Rechner I.

YL, YN Matrizen der Dimension N * N * P. YL (I, *, *) entspricht der Weiterentwicklung von XL für Rechner I. YN analog für XN.

Hauptprogramm:

```
R, #OR, X, XL, XN : = 0;
DO  WHILE NOT (alle TA-Typen vollständig aufgeteilt);
      J : = NEXT-TT;
      L1 : = Anzahl der Objektreferenzen, die für TA-Typ J
             noch aufzuteilen sind;
      I : =  BEST-R (...); /* Bestimmung des geeigneten
             Rechners durch Aufruf der Funktionsprozedur
             BEST-R (Bild 7) */
             X (*) : = Y (I, *);
             XL (*,*)  : = YL (I, *, *);
             XN (*,*)  : = YN (I, *, *);
             L2 : = max (MAX-OR – #OR (I), L1);
             #OR (I) = #OR (I) + L2;
             R (I, J) = R (I, J) + L2 / L (J);
         END;
         D : = X;
         /* Bestimmung von TL */
         TL : = 0;
         DO I = 1 TO N;
            DO  K = 1 TO P;
                TL (I) : = TL (I) + XL (I, K);
            END;
         END;
```

Bild 6 Hauptprogramm zu Verfahren 3

Zusätzlich benutzte Datenstrukturen:

#TL Vektor der Dimension N. Wird benutzt bei der Auswahl des Rechners, dem ein TA-Typ zugeordnet wird. Gibt für jeden Rechner I an, wieviele Teil-DML's (Sperranforderungen) lokal bearbeitbar sind, wenn der TA-Typ J Rechner I zugeteilt wird und die damit verbundene Weiterentwicklung von X übernommen wird. Die Zuordnung geschieht zu dem Rechner, für den #TL maximal ist.

MOEGL Anzahl der Teil-DML's (Sperranforderungen), die Rechner I für eine Partition K vom TA-Typ J übernehmen kann.

Algorithmus für BEST-R:

```
#TL : = 0;
DO I = 1 TO N;
 L2 : = max (MAX-OR – #OR (I), L1); /* max. Anzahl von
    Objektreferenzen, die von TA-Typ J in Rechner noch
    bearbeitbar sind */
 /* Bestimmung von Y, YL und YN */
 IF  L2 > 0 THEN DO;
    /* Vorbesetzung */
    Y (I,*) : = X (*);
    XL (I, *,*) : = XL (*, *);
    YN (I, *, *) : = XN (*, *);

    DO  K = 1 TO P;
        IF T (J, K) > 0 THEN DO;
           MOEGL = T (J, K) * L2 / L(J);
           /*  Bestimmung von Y (I, K) */
           IF   X(K) = 0   OR
              (X (K) NOT = I AND  (XN (I,K) + MOEGL) >
              XL (X (K), K))
           THEN Y (I, K) : = I;
           /*  Weiterentwicklung von YL und YN */
           IF  Y (I, K) = I THEN DO;
              YL (I, I, K) : = XL (I, K) + XN (I, K)
                                         + MOEGL;
              YN (I, I, K) : = 0;
              IF X (K) > 0  AND (Y (I, K) NOT = X (K))
              THEN DO;
                    /* Partition K ist in Y (I) im Gegensatz zu X dem Rechner I zugeordnet */
                    YL (I, X (K), K) : = 0;
                    YN (I, X (K), K) : = XL (X (K), K);
              END;
           END;
           ELSE DO; /* I ist nicht ‚Besitzer' von Partition
                       K */
             YL (I, I, K) : = 0;
             YN (I, I, K) : = XN (I, K) + MOEGL;
           END;
        END; /* IF T (J, K) > 0 */
    END; /* K */
    /* Bestimmung von #TL (I) */
    DO  I2 = 1 TO N;
        DO K = 1 TO P;
           #TL (I) : = #TL (I) + YL (I, I2, K);
        END;
    END;
 END; /* L2 > 0 */
END; /* I * /
I : = Rechner-Nr. für die #TL maximal ist;
RETURN (I);
```

Bild 7 Funktionsprozedur BEST-R (Verfahren 3)

Typs weiterentwickelt und zwar in Abhängigkeit von dem Rechner, der von der Funktion BEST-R zur Übernahme des TA-Typs bestimmt wurde. Die Berechnung der Weiterentwicklungen erfolgt in BEST-R und wird in den Variablen Y, YL bzw. YN festgehalten. Wenn alle TA aufgeteilt sind, stellt X die berechnete Daten-/PCA-Verteilung dar, und TL läßt sich aus XL bestimmen.

Bei der Darstellung von BEST-R (Bild 7) wird auf die Angabe von Parametern verzichtet, da sie nichts zum Verständnis beitragen. Vielmehr werden die bereits eingeführten Variablen (J, L1, X, XL, ...) auch hier benutzt.

Bei der Bestimmung des geeigneten Rechners zur Übernahme eines TA-Typs J berücksichtigt BEST-R alle Rechner I, denen noch Objektreferenzen zugeordnet werden können (L2 > 0). Für jeden dieser Rechner werden dabei Y, YL und YN mit den aktuellen Werten von X, XL bzw. XN vorbesetzt. Danach wird zu jeder Partition K die Weiterentwicklung von Y, YL und YN bestimmt unter der Voraussetzung, der TA-Typ wird dem Rechner zugeordnet. Bei der Bestimmung von Y, d.h. der Weiterentwicklung Verfahren 3 soll nun auch für N = 2 auf die Referenzmatrix aus Bild 2 angewendet werden:

MAX-OR sei 21000. Ordnet man zunächst TA-Typ T1 Rechner 1 zu, dann erhält dieser die Verantwortung für die Partitionen P1, P2 und P4. TA-Typ T2, der nur die Partitionen P1 und P2 referenziert, wird danach (im Gegensatz zu Verfahren 2) ebenfalls Rechner 1 zugeordnet, da dort beide TA-Typen dann vollständig lokal bearbeitbar sind. Die weitere Abarbeitung des Algorithmus ergibt, daß die verbleibenden TA-Typen auf Rechner 2 abzuarbeiten sind. Damit sind insgesamt 19000 der Objektreferenzen Rechner 1 und 21000 Rechner 2 zugeordnet. Als weiteres Ergebnis erhält man:

D = X = (1, 1, 2, 2, 2, 2) d.h. P1 und P2 sind Rechner 1, P3 bis P6 Rechner 2 zugeordnet

sowie

XL (1) = (9000, 7000, 0, 0, 0, 0),
XN (1) = (0, 0, 0, 2000, 0, 1000),
XL (2) = (0, 0, 6000, 5000, 5000, 3000),
XN (2) = (1000, 1000, 0, 0, 0, 0).

Aus XL bekommt man TL (1) = 16000 und TL (2) = 19000.

Durch den Algorithmus ergibt sich also, daß für die Referenzmatrix aus Bild 2 und N = 2 etwa 87.5 % der Teil-DML's (Sperranforderungen) lokal verarbeitbar sind. Dies sind fast 20 Prozentprodukte mehr als die mit Verfahren 2 erzielten 68 %.

3.4 Beobachtungen

Mit dem Beispiel der einfachen Referenzmatrix aus Bild 2 konnten bereits die unterschiedlichen Charakteristika der vorgestellten drei Algorithmen gezeigt werden. Bei einer bereits festgelegten Datenverteilung, was für DB-Distribution der Normalfall ist, liegt nur ein geringes Optimierungspotential bei der Erstellung einer Routing-Tabelle vor (Verfahren 1). Das heißt, durch die Datenverteilung ist das der Daten-/PCA-Verteilung, erfolgt ein ‚Besitzrechtwechsel' im Vergleich zu X, wenn eine Partition K bisher noch keinem Rechner zugeordnet war (X (K) = 0) oder wenn der XL-Wert des bisherigen Besitzers kleiner ist als die Summe von XN und MOEGL für den betrachteten Rechner I.

Ausmaß einer lokalen Verarbeitung bereits weitgehend festgelegt.

Mehr Freiheitsgrade (und in der Regel einen höheren Anteil lokaler Verarbeitung) erhält man, wenn die Daten- bzw. die PCA-Verteilung zusammen mit der Routing-Tabelle bestimmt wird. Von den hierzu vorgeschlagenen zwei Algorithmen ist Verfahren 3 offentsichtlich klar überlegen, da hierbei die Erstellung der Routing-Tabelle und der Daten-/PCA-Verteilung schrittweise koordiniert wird. Dies wurde auch durch die Anwendung der beiden Verfahren 2 und 3 auf eine Reihe von logischen Seitenreferenzstrings bestätigt [15]. In einem Referenzstring sind dabei alle DB-Seitenzugriffe während einer kommerziellen DB-Anwendung protokolliert. Daher konnten hieraus die für die Algorithmen benötigten Referenzmatrizen ermittelt werden.

Bei diesen Untersuchungen ergaben sich noch weitere wesentliche Beobachtungen. So ist der erfolgreiche Einsatz von DB-Distribution und DB-Sharing mit Primary Copy Locking (PCL) nur möglich, wenn die folgenden beiden Voraussetzungen erfüllt sind.

1) Es muß eine ausreichende Menge ‚relevanter' DB-Partitionen vorliegen, d.h., es dürfen beispielsweise nicht 90 % aller DB-Zugriffe auf eine Partition entfallen.

   Diese Voraussetzung läßt sich für PCL immer erreichen, da hier nur eine logische Partitionierung vorzunehmen ist. Daher kann man die DB auch in nahezu beliebig kleine Partitionen unterteilen. Bei DB-Distribution ist dies jedoch i.a. nicht möglich, da die Partitionen den Rechnern physisch zugeordnet werden müssen. Daher liegt hier meist ein relativ grobes Verteilgranulat vor (z.B. Dateien).

2) Alle TA-Typen müssen weitgehend auf einem Rechner bearbeitbar sein, ohne diesen zu überlasten. Wenn nämlich ein TA-Typ ‚gesplittet' werden muß, d.h. von mehreren Rechnern zu bearbeiten ist, dann kann für diesen TA-Typ in höchstens einem der Rechner Lokalität genutzt werden.

   Auch hier ist u.U. eine Entschärfung des Problems möglich, nämlich durch Einführung von *Sub-TA-Typen* (z.B. unter Berücksichtigung aktueller Eingabeparameter). In Bankanwendungen wäre es beispielsweise sinnvoll, den dominierenden TA-Typ KONTENBUCHUNG [3] über Wertebereiche bezüglich des aktuellen Parameters KONTO-NR in Sub-TA-Typen zu unterteilen.

Wenn diese beiden Voraussetzungen erfüllt sind (was umso schwieriger ist, je mehr Rechner eingesetzt werden sollen), dann hängt die erreichbare Performance im wesentlichen noch davon ab, inwieweit die TA-Typen auf disjunkten DB-Partitionen operieren. Für PCL sind auch noch gute Ergebnisse zu erwarten (d.h. wenig Interprozessor-Kommunikationen), falls auf Partitionen von rechnerübergreifendem Interesse vorwiegend lesend zugegriffen wird [6].

4 Resümee

Nach einer Einführung wurde die Stellung der Lastkontrolle in einem lokal gekoppelten Mehrrechner-Datenbanksystem untersucht. Hier kann die Lastkontrolle die TA-Last unter Berücksichtigung der aktuellen Rechnerauslastung sowie der Betriebsmittelzuordnung auf die verfügbaren Rechner verteilen. Dieses TA-Routing kann durch Verwendung einer sogenannten Routing-Tabelle besonders effizient durchgeführt werden.

Im Hauptteil der Arbeit wurden drei Verfahren zur Berechnung einer Routing-Tabelle angegeben, wobei in zwei Fällen zusätzlich eine Datenverteilung für DB-Distribution bzw. eine PCA-Verteilung für DB-Sharing mit Primary Copy Locking (PCL) erstellt wird. Die Anwendung der Verfahren auf mehrere Seitenreferenzstrings zeigte, daß der als Verfahren 3 bezeichnete Algorithmus die besten Ergebnisse liefert. In ihm wird eine Heuristik verwendet, die die Erstellung der Routing-Tabelle und der Daten-/PCA-Verteilung schrittweise koordiniert.

Bei den Untersuchungen zeigte sich auch die vergleichsweise geringe Flexibilität, die DB-Distribution der Lastkontrolle erlaubt, da die Datenverteilung meist festgelegt ist. Selbst wenn neben der Routing-Tabelle auch die Datenverteilung bestimmt wird, können für die Verteilung der Daten nur relativ grobe Granulate (z.B. Dateien) berücksichtigt werden. Bei PCL dagegen können die Partitionen beliebig fein gewählt werden, und die PCA-Verteilung kann stets mit der Routing-Tabelle zusammen optimiert werden, da sie relativ einfach angepaßt werden kann. Weiterhin erlaubt PCL – im Gegensatz zu DB-Distribution – auch Kommunikationseinsparungen, falls auf Partitionen von rechnerübergreifenden Interesse vorwiegend lesend zugegriffen wird.

In den Algorithmen zur Berechnung der Routing-Tabellen werden die Kosten der Transaktionsverarbeitung vergleichsweise grob abgeschätzt, so daß eine gleichmäßige Rechnerauslastung durch die Routing-Tabelle allein nicht immer gewährleistet ist. In weiterer Forschungsarbeit gilt es daher Verfahren zu entwickeln, die den anfallenden Kommunikationsoverhead sowie die Rechnerbelastungen durch externe Aufträge besser berücksichtigen.

Literatur

[1] *Härder, T., Rahm, E.:* Klassifikation von Mehrrechner-Datenbanksystemen – Anforderungen, Entwurfsprinzipien, Realisierungskonzepte. Interner Bericht 152/85, FB Informatik, Univ. Kaiserslautern (1985)

[2] *Gray, J.* et al.: One Thousand Transactions per Second. In: Proc. IEEE Spring CompCon, San Francisco (1985), S. 96–101

[3] *Anon.* et al.: A Measure of Transaction Processing Power. In: Datamation (April 1985)

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

[5] *Reuter, A.:* Load Control and Load Balancing in a Shared Database Management System. Interner Bericht 129/85, FB Informatik, Univ. Kaiserslautern (1985)

[6] *Rahm, E.:* Primary Copy Synchronization for DB-Sharing. Erscheint in: Information Systems, Vol. 11, No. 4 (1986)

[7] *Rahm, E.:* Concurrency Control in DB-Sharing Systems. FB Informatik, Univ. Kaiserslautern (1986)

[8] *Rahm, E.:* Buffer Invalidation Problem in DB-Sharing Systems. Interner Bericht 154/86, FB Informatik, Univ. Kaiserslautern (1986)

[9] *Shoens, K.* et al.: The AMOEBA Project. In: Proc. IEEE Spring CompCon, San Francisco (1985), S. 102–105

[10] *Härder, T.:* DB-Sharing vs. DB-Distribution – die Frage nach dem Systemkonzept zukünftiger DB/DC-Systeme. In: Proc. 9. NTG/GI-Fachtagung ‚Architektur und Betrieb von Rechensystemen', Stuttgart, NTG-Fachberichte 92, S. 151–165, VDE-Verlag 1986

[11] *Gray, J.:* Notes on Database Operating Systems. In: Bayer, R., Graham R. M., Seegmüller, G. (Hrsg.): Operating Systems – An Advanced Course. Lecture Notes in Computer Science, Vol. 60, S. 393–481. Berlin, Heidelberg, New York, Tokyo: Springer 1978

[12] *Bayer, R., Elhardt, K., Kießling, W., Killar, D.:* Verteilte Datenbanksysteme – Eine Übersicht über den heutigen Entwicklungsstand. In: Informatik-Spektrum 7 (1984), S. 1–19

[13] *Rahm, E.:* A Reliable and Efficient Synchronization Protocol for DB-Sharing. Interner Bericht 139/85, FB Informatik, Univ. Kaiserslautern (1985)

[14] *Härder, T., Meyer-Wegener, K.:* Transaktionssysteme und TP-Monitore – Eine Systematik ihrer Aufgabenstellung und Implementierung. In: Informatik – Forschung und Entwicklung 1 (1986), S. 3–25

[15] *Rahm, E.:* Analyse von logischen Seitenreferenzstrings zur optimierten Lastkontrolle bei Simulationen von Mehrrechner-Datenbanksystemen. Technischer Bericht, FB Informatik, Univ. Kaiserslautern (1985)

Diese Arbeit wurde von der SIEMENS AG finanziell unterstützt.