Ein Kostenmodell der parallelen Anfragebearbeitung in Shared-Nothing-Datenbanksystemen

Robert Marek

University of Kaiserslautern
Dept. of Computer Science
67618 Kaiserslautern, GERMANY
Tel: +49 631 205 3272, FAX: +49 631 205 3558
E-mail: marek@informatik.uni-kl.de

Abstract:

Zunehmend komplexe und datenintensive Benutzeranfragen auf Datenbanken verlangen parallele Verarbeitungsansätze. Vor allem Datenbanksysteme der Architekturklasse Shared-Nothing bieten derzeit eine geeignete Basis für die parallele Anfragebearbeitung. Im Hinblick auf den interaktiven Charakter komplexer Datenbankanfragen ist eine Verkürzung der Antwortzeit das vorrangige Leistungsziel paralleler Datenbanksysteme. Im Falle der heute dominanten mengenorientierten relationalen Anfragesprachen erlaubt vor allem Intra-Operator-Parallelität eine effektive Antwortzeitverkürzung. Die Antwortzeit kann jedoch nicht durch zunehmende Parallelisierung beliebig verkürzt werden. Wird ein gewisser Parallelisierungsgrad überschritten, tritt vielmehr wieder eine Verschlechterung der Antwortzeit ein. Dieser Effekt liegt einerseits in einem beschränkten Parallelisierungspotential, andererseits in mit zunehmendem Parallelisierungsgrad steigenden Kooperations- und Kommunikationskosten begründet. Die Bestimmung des optimalen Parallelisierungsgrades ist daher von besonderer Bedeutung. Aus diesem Grunde haben wir ein analytisches Kostenmodell entwickelt, das die Antwortzeitentwicklung von Datenbankanfragen in Abhängigkeit vom Grad der Parallelisierung beschreibt. Anhand dieses Modells können wir grundsätzliche Trade-Offs der parallelen Anfragebearbeitung untersuchen. Weiterhin kann das Kostenmodell zur Unterstützung des Optimierers bei der Anfrageparallelisierung sowie zur Bestimmung einer geeigneten Datenverteilung genutzt werden. Das Kostenmodell wurde mit Hilfe begleitender Simulationsversuche zur parallelen Bearbeitung von Anfragen validiert.

Keywords: Parallele Datenbanksysteme; Shared-Nothing; komplexe Anfragen; Anfragebearbeitung; Intra-Operator-Parallelität; Kostenmodell; Simulation;

1 Einführung

Parallele Datenbanksysteme sind heute für eine leistungsfähige Transaktions- und Anfragebearbeitung obligatorisch [DeWitt und Gray 1992, Valduriez 1993]. Derartige Systeme nutzen die Kapazität mehrerer lokal verteilter Verarbeitungsknoten, die über ein Hochleistungsnetzwerk miteinander verbunden sind. Bevorzugt werden leistungsfähige und preiswerte Mikroprozessoren verwendet, da sie gegenüber herkömmlichen Mainframe-Lösungen eine weitaus bessere Kosteneffektivität (Kosten/MIPS) gewährleisten.

Große Bedeutung in der Datenbankverarbeitung gewinnen heute komplexe Ad-Hoc-Anfragen, was zum Teil durch die wachsende Verbreitung mächtiger Anfragesprachen und Benutzer-Tools bedingt ist. Doch auch die mächtigen Anfrage-, Manipulations- und Wartungsoperationen von DB-Anwendungen kommender Generationen wie Ingenieur-Anwendungen, VLSI-Entwurf, Multimedia-Anwendungen etc. vergrößern die Komplexität von DB-Anfragen deutlich [Silberschatz et al. 1991]. Für diesen Lasttyp steht eine Optimierung des Antwortzeitverhaltens im Vordergrund, um ein für den Dialogbetrieb akzeptables Antwortzeitverhalten gewährleisten zu können. Derartige Anfragen betreffen im allgemeinen große Datenvolumina und/oder führen aufwendige Berechnungen durch, so daß akzeptable Antwortzeiten nur durch den massiven Einsatz von Parallelität in der DB-Verarbeitung erzielt werden können [Pirahesh et al. 1990].

Die wichtigste Klasse paralleler Datenbanksysteme bilden derzeit *Shared-Nothing*-Architekturen [Stonebraker 1986, DeWitt und Gray 1992]. Zu den Parallelverarbeitung unterstützenden Shared-Nothing-Systemen gehören u.a. Produkte wie Tandem NonStop SQL [The Tandem Database Group 1989, Englert et al. 1990] und Teradata DBC/1012 [Neches 1986] sowie eine Reihe von Prototypen: Bubba [Boral et al. 1990], Gamma [DeWitt et al. 1990], EDS [Watson und Townsend 1991] und PRISMA/DB [Apers et al. 1992]. Shared-Nothing-Systeme bestehen aus mehreren funktional gleichwertigen Prozessorelementen (PE). Jedes PE verfügt über ein oder mehrere Prozessoren, lokalen Hauptspeicher sowie eigene Kopien der Anwendungs- und System-Software wie Betriebssystem und Datenbankverwaltungssystem (DBVS). Die Kommunikation zwischen den Prozessorknoten erfolgt in Shared-Nothing-Systemen nachrichtenbasiert - aus Leistungsargumenten i.d.R. über ein Hochgeschwindigkeitsnetzwerk. Die prägende Eigenschaft von Shared-Nothing-Systemen ist eine Auf- und Verteilung der Datenbank in sogenannte Partitionen derart, daß jedes PE eine eigene Partition der Datenbank “besitzt” (*Datenverteilung*). Transaktionen (Anfragen), die auf die Daten fremder PE zugreifen, starten auf den entsprechenden PE sogenannte Sub-Transaktionen, die den Datenzugriff stellvertretend für die eigentliche Transaktion durchführen.

Zur Verkürzung der Antwortzeit wird Intra-Transaktionsparallelität benötigt - in Form von *Inter*- oder *Intra-DML-Parallelität*. Inter-DML-Parallelität bezeichnet die konkurrente Ausführung verschiedener DML-Befehle (DB-Operationen) einer Transaktion. Aufgrund der im allgemeinen geringen Anzahl von DB-Operationen pro Transaktion sowie Vorgaben in der Ausführungsreihenfolge dieser Operationen ermöglicht diese Form der Parallelität i.d.R. nur eine eingeschränkte Parallelisierung.

Als weiterer Nachteil erweist sich, daß der Anwendungsprogrammierer Inter-DML-Parallelität mit Hilfe geeigneter Sprachmittel explizit darstellen muß. Aus diesen Gründen unterstützen bestehende Systeme Intra-Transaktionsparallelität lediglich in Form von Intra-DML-Parallelität[1]. Ermöglicht wird die Nutzung von Intra-DML-Parallelität v.a. durch relationale Datenbanksysteme mit ihren deskriptiven, mengenorientierten Anfragesprachen (z.B. SQL) [DeWitt und Gray 1992]. Implementiert wird Intra-DML-Parallelität durch den DBVS-Anfrageoptimierer - vollkommen transparent für Benutzer und Anwendungsprogrammierer. Für jede DB-Operation erstellt der Optimierer hierzu einen (parallelen) Ausführungsplan. Dieser spezifiziert, in welcher Weise die Basisoperatoren (z.B. Scan, Filter, Join, etc.) der DB-Operation abzuarbeiten sind. Intra-DML-Parallelität kann in zwei Formen angeboten werden: *Inter*- und *Intra-Operator-Parallelität*. Inter-Operator-Parallelität bezeichnet die konkurrente Bearbeitung verschiedener Operatoren, wohingegen bei Intra-Operatorparallelität eine Parallelisierung einzelner Operatoren erfolgt. In beiden Fällen ist die Parallelisierung entscheidend von der gewählten Datenverteilung abhängig. Die Datenbank sollte derart auf Prozessorknoten verteilt werden, daß Operatoren oder Sub-Operatoren auf disjunkten Datenpartitionen parallel von verschiedenen PE bearbeitet werden können. Typischerweise werden hierzu Relationen *horizontal*, d.h. tupelweise, auf mehrere PE aufgeteilt.

Die Antwortzeit von Transaktionen hängt in hohem Maße von der Anzahl der PE ab, die für deren Bearbeitung eingesetzt werden: zusätzliche PE verkürzen prinzipiell die Antwortzeit, wobei idealerweise ein linearer Zusammenhang zwischen PE-Anzahl und Antwortzeitverbesserung besteht. In der Praxis kann diese lineare Beziehung jedoch nur begrenzt erzielt werden, und die Antwortzeit kann durch zunehmende Parallelisierung nicht beliebig verkürzt werden. Wird ein gewisser Parallelisierungsgrad überschritten, tritt vielmehr wieder eine Verschlechterung der Antwortzeit ein. Dieser Effekt liegt einerseits in einem beschränkten Parallelisierungspotential, andererseits in mit zunehmendem Parallelisierungsgrad steigenden Kooperations- und Kommunikationskosten begründet. Die Bestimmung des *optimalen* Parallelisierungsgrades ist daher von besonderer Bedeutung.

Im Einbenutzerbetrieb ist die Frage nach dem optimalen Parallelisierungsgrad äquivalent zu der Frage nach demjenigen Parallelisierungsgrad, der die geringste Antwortzeit bietet. Die Optimierungsentscheidung hängt in diesem Fall vorwiegend von statischen Parametern wie Datenverteilung, Relationengrößen, Zugriffsmethode und Schärfe von Selektionsprädikaten ab. Während im Einbenutzerbetrieb alle Betriebsmittel des DBS (CPU, Hauptspeicher, Datenobjekte,...) jeweils einer Anfrage exklusiv zur Verfügung stehen, zeichnet sich Mehrbenutzerbetrieb dadurch aus, daß die begrenzten Betriebsmittel geeignet zwischen konkurrenten Anfragen aufgeteilt werden müssen. Infolge dessen verlangt Mehrbenutzerbetrieb eine andere Definition des Optimalitätsbegriffs. Dort gilt es das Verhältnis zwischen Nutzen (Antwortzeitverkürzung) und Kosten (Kooperations- und Kommunikations-Overhead) der Parallelisierung in Abhängigkeit von der Systemauslastung zu optimieren mit

1. Im Falle einer Ad-Hoc-Anfrage beinhaltet eine Transaktion lediglich einen einzigen DML-Befehl. Intra-Transaktionsparallelität ist hier gleichbedeutend mit Intra-DML-Parallelität.

dem Ziel, globale Antwortzeit- und Durchsatzvorgaben möglichst optimal zu erfüllen. Der Parallelisierungsgrad im Mehrbenutzerbetrieb liegt - je nach Systemauslastung - zum Teil deutlich unter dem des Einbenutzerbetriebes [Marek und Rahm 1993, Rahm und Marek 1993].
Unser Ziel ist es, eine geeignete Unterstützung des Anfrageoptimierers bei der Bestimmung des optimalen Parallelisierungsgrades zu finden. Für den Einbenutzerbetrieb haben wir ein Kostenmodell entwickelt, das die Antwortzeitentwicklung in Abhängigkeit vom Parallelisierungsgrad beschreibt. Anhand dieses Modells können wir grundsätzliche Trade-Offs der parallelen Anfragebearbeitung (z.B. Kommunikations-Overhead versus Parallelisierungsgewinn) untersuchen, ohne aufwendige Versuche basierend auf Simulationsmodellen oder Prototypimplementierungen paralleler DBS durchführen zu müssen. Mit Hilfe des Kostenmodells können wir in einfacher Weise den Einfluß signifikanter Systemparameter auf die Effektivität der Anfrageparallelisierung untersuchen. Das Kostenmodell kann ferner als Entscheidungshilfe zur Bestimmung einer geeigneten Datenverteilung genutzt werden.
Im folgenden Kapitel stellen wir ein abstraktes Modell der parallelen Anfragebearbeitung vor, aus dem wir das analytische Kostenmodell ableiten. Anhand einer detaillierten Betrachtung der parallelen Bearbeitung von Scan- und Join-Anfragen, schätzen wir in Kapitel 3 die Koeffizienten des analytischen Kostenmodells ab. Daran anschließend wollen wir das Kostemodell einschließlich der errechneten Koeffizienten anhand der parallelen Bearbeitung von Scan- und Join-Anfragen in Shared-Nothing-Systemen validieren. Dazu haben wir die parallele Bearbeitung von Anfragen mit Hilfe eines Simulationsmodelles untersucht. In Kapitel 4 stellen wir unser Simulationsmodell vor, und präsentieren schließlich in Kapitel 5 die durchgeführten Versuche und bewerten das analytische Kostenmodell.

2 Ein abstraktes Modell der parallelen Anfragebearbeitung

Zur Herleitung unseres Kostenmodells wollen wir uns auf ein einfaches Modell der parallelen Anfragebearbeitung stützen, das weitestmöglich von physischen Aspekten der Anfragebearbeitung abstrahiert. Wir unterstellen eine Shared-Nothing-Hardware bestehend aus n Knoten (PE), die über ein Kommunikationsnetzwerk miteinander verbunden sind. Das Parallelisierungsmodell besteht aus einer Verwaltungseinheit und einer Anzahl von Ausführungseinheiten. Die Ausführungseinheiten beschreiben diejenigen Teilaufgaben, die durch die Zerlegung einer parallel zu verarbeitenden (komplexeren) Aufgabe entstehen. Eine derartige Aufgabe kann beispielsweise die Bearbeitung eines DML-Befehles oder eines Operators (Scan, Join etc.) beinhalten. Die Verwaltungseinheit fungiert als Koordinator dieser Ausführungseinheiten. Sowohl die Verwaltungseinheit als auch die Ausführungseinheiten werden jeweils durch einen eigenen Prozeß repräsentiert und genau einem PE zugeordnet. Die Bearbeitung der gegebenen Aufgabe beginnt mit der Initialisierung der Ausführungseinheiten durch die Verwaltungseinheit. Dazu sendet der Verwalter Aktivierungsnachrichten an diejenigen PE, denen eine Ausführungseinheit zugeordnet werden soll. Wir unterstellen, daß diese Nachrichten sequentiell verschickt werden.

Mit Hilfe dieses Modells können wir sowohl Parallelität innerhalb einzelner Aufgaben als auch zwischen verschiedenen Aufgaben darstellen. Im letzteren Fall werden der Verwaltungseinheit mehrere Mengen von Ausführungseinheiten zugeordnet (Bild 1).

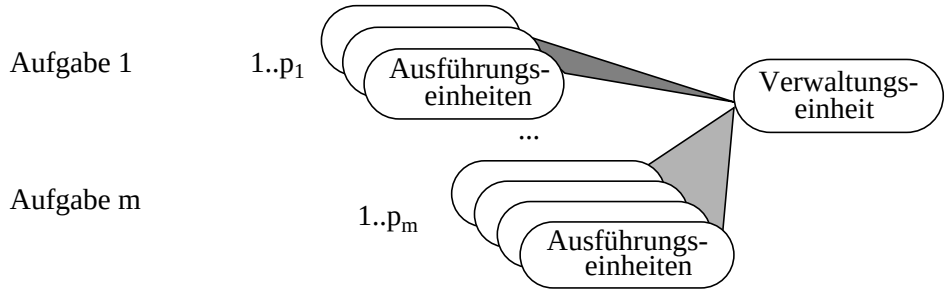


Bild 1: Ein abstraktes Modell der parallelen Anfragebearbeitung.

Wir wollen nun im folgenden untersuchen, wie sich die Bearbeitungszeit von Anfragen in Abhängigkeit von der Anzahl der Ausführungseinheiten verhält. Wir identifizieren dabei drei verschiedene Kostenanteile: ein konstanter (von der Anzahl der Ausführungseinheiten unabhängiger) Anteil, ein mit zunehmender Anzahl von Ausführungseinheiten abnehmender Anteil, sowie mit ein mit zunehmender Anzahl von Ausführungseinheiten steigender Kostenanteil. Im folgenden stellen wir eine allgemeine Betrachtung dieser Kostenterme vor und präzisieren den letztgenannten Kostenanteil (Parallelisierungsgewinn) in Abhängigkeit vom Kostenverlauf der Operatoren (lineare bzw. logarithmische Verarbeitungskosten) und Speichermedium (Hauptspeicher bzw. Platte) der Operanden. Die Koeffizienten der Kostenformel werden wir auf analytischem Wege in Kapitel 3 herleiten[2].
An dieser Stelle wollen wir hervorheben, daß das Kostenmodell auf einer weitreichenden Gleichverteilungsannahme basiert. Im Sinne einer praktikablen Kostenabschätzung berücksichtigen wir keine Streuung (*Skew* [Walton et al. 1991]) in den Verarbeitungskosten paralleler Ausführungseinheiten. Wir nehmen im folgenden an, daß Operanden zu parallelisierender Operatoren gleichmäßig auf alle Ausführungseinheiten verteilt werden und daß unter den parallelen Ausführungseinheiten keine Streuung in den Verarbeitungskosten pro Operand auftritt. Diese Annahmen implizieren beispielsweise eine gleichmäßige Datenverteilung der Basisrelationen auf Rechnerknoten, gleichmäßige E/A-Kosten pro Ausführungseinheit sowie einheitlich selektive Prädikate verteilter Ausführungseinheiten von Selektionsbefehlen. Sicherlich sollten weiterführende Arbeiten den Einfluß von Skew auf die Kostenentwicklung der Operatorparallelisierung berücksichtigen.

2. In [Wilschut et al. 1992] wurde ein ähnliches Kostenmodell vorgestellt. Im Gegensatz zu unserem Ansatz, der die Antwortzeit ganzer Anfragen (u.U. aus mehreren Operatoren bestehend) modelliert, beschränkt sich [Wilschut et al. 1992] auf einzelne Operatoren. Darüber hinaus werden in [Wilschut et al. 1992] ausschließlich Operatoren mit linearen Verarbeitungskosten zugrunde gelegt und der Einfluß des Speichermediums wird nicht explizit modelliert. Die Berechnung der Koeffizienten erfolgt in [Wilschut et al. 1992] auf Basis von Messungen an der Prototypimplementierung eines Hauptspeicher-DBS (*PRISMA/DB*). Wir wählen einen analytischen Ansatz (vgl. Kapitel 3).

Konstanter Antwortzeitkostenanteil:

In der Antwortzeit sind im allgemeinen Kostenanteile enthalten, die von Parallelisierungsmaßnahmen unabhängig sind. Hierzu gehören vor allem Kosten für die Initialisierung der Anfrage sowie von der Parallelisierung nicht betroffene Operatoren. Beispielsweise erfordert die verteilte Berechnung einer Verbundoperation in der Regel das Mischen (*Merge*) der Treffertupeln in einer zentralen Instanz. Der Aufwand für die Merge-Operation hängt von der Anzahl der Treffertupeln, nicht aber vom Parallelisierungsgrad des Verbundoperators ab. Derartige Kosten wollen wir mit einem konstanten Anteil a abschätzen.

Kooperations- und Kommunikationskosten:

Neben dem konstanten Kostenanteil ist in der Antwortzeit insbesondere auch der Aufwand für das Starten (und Beenden) der parallelen Ausführungseinheiten enthalten. Die hierbei anfallenden Kosten umfassen u.a. Aktivierungsnachrichten bzw. Quittungsmeldungen sowie Initialisierungskosten verteilter Ausführungseinheiten und sind proportional zur Anzahl der Ausführungseinheiten *m*:

(i) $$b \times m$$

Verkürzung der Bearbeitungszeit durch Parallelverarbeitung:

Im Falle relationaler Operatoren sind in der Regel Tupelmengen gegeben, auf die die Operatoren anzuwenden sind. Betragen die Verarbeitungskosten eines Operators pro Tupel c Zeiteinheiten und wachsen die Verarbeitungskosten linear mit der Tupelanzahl, so nimmt die sequentielle Verarbeitung aller M Tupeln insgesamt

(ii) $$c \times M$$

Zeiteinheiten in Anspruch. Die Verarbeitung auf *m* Prozessorelementen dauert idealerweise lediglich

(iii) $$\frac{c \times M}{m}$$

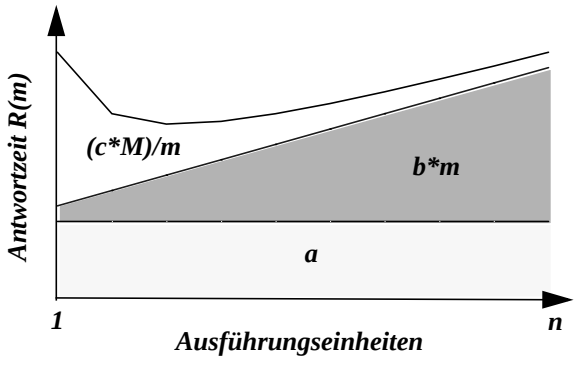
Zeiteinheiten, wobei dieser Quotient gleichzeitig die Bearbeitungszeit einer jeden Ausführungseinheit angibt. Man spricht in diesem Falle von linearem *Speedup.* Der Speedup mißt die Verbesserung der Bearbeitungszeit einer Aufgabe durch Parallelisierung in *m* Ausführungseinheiten (bzw. PE) [Englert et al. 1990]. Dabei ist der Speedup definiert als Quotient der Bearbeitungszeit auf 1 PE und der Zeit, die für die Bearbeitung auf *m* PE benötigt wird (siehe unten).

Mit Hilfe der beschriebenen Kostenterme können wir nun abschätzen, wie sich die Parallelisierung linearer Operatoren auf die Query-Antwortzeit auswirkt. Für ein unireletionales SELECT beträgt beispielsweise die Anfrageantwortzeit *R* in Abhängigkeit vom Parallelisisierungsgrad *m* des Scan-Operators

$$R(m) = a + b \times m + \frac{c \times M}{m}$$

Zeiteinheiten, wobei eine hauptspeicherresidente Speicherung der Eingaberelation angenommen ist.

Wir können bei der parallelen Anfragebearbeitung eine Antwortzeitentwicklung beobachten, wie sie in Bild 2 dargestellt ist. Denjenigen Parallelisierungsgrad m_{opt}, der die geringste Antwortzeit erzielt,



$$R(m) = a + b \times m + \frac{c \times M}{m}$$

Bild 2: Einfluß der parallelen Anfragebearbeitung auf die Antwortzeit.

können wir ermitteln, indem wir die Ableitung der Kostenformel gleich Null setzen:

$$R'(m) = b - \frac{c \times M}{m^2} \qquad \text{und} \qquad R'(m_{opt}) = 0 \qquad \Rightarrow \qquad m_{opt} = \sqrt{\frac{c \times M}{b}}$$

Wir erkennen, daß dieser Parallelisierungsgrad unabhängig vom konstanten Kostenanteil a der Antwortzeit ist. Sind die Anzahl der von der Parallelisierung betroffen Tupeln *M*, die Verarbeitungskosten *c* pro Tupel und die Initialisierungskosten *b* pro Ausführungseinheit bekannt, können wir nun mit Hilfe der Formel den optimalen Parallelisierungsgrad errechnen. Wir erkennen, daß dieser Parallelitätsgrad generell abhängig ist von dem Verhältnis zwischen Nutzarbeit *c*M* und dem Parallelisierungs-Overhead *b*.
Die Effektivität der Operatorparallelisierung kann mit Hilfe des Antwortzeit-Speedups bewertet werden (siehe oben). In unserem Beispiel beträgt der Antwortzeit-Speedup:

$$\frac{R(1)}{R(m)} = \frac{a + b + c \times M}{a + b \times m + \frac{c \times M}{m}}$$

Im Gegensatz zum optimalen Parallelisierungsgrad ist die Effektivität der Anfrageparallelisierung vom konstanten Kostenanteil *a* abhängig. Ist dieser Anteil im Verhältnis zu den übrigen Kostenanteilen hoch, so hat die Operatorparallelisierung wenig Einfluß auf die Antwortzeitverbesserung der gesamten Anfrage (*die Operatorparallelisierung ist wenig effektiv*). Es sei bemerkt, daß linearer Speedup im Falle von *a*=0 und *b*=0 erzielt wird, d.h. wenn keinerlei Initialisierungskosten und Verzögerungen durch Nachrichten auftreten, so daß alle Ausführungseinheiten gleichzeitig beginnen.

Parallelisierung logarithmischer Operatoren:

Wie bereits erwähnt, liegt obigem Zusammenhang die Annahme linearer Verarbeitungskosten zugrunde, d.h., die Bearbeitungszeit verhält sich umgekehrt proportional zur Anzahl der eingesetzten PE. Wenngleich diese Annahme für einen Großteil der Implementierungen relationaler Operatoren erfüllt ist (z.B. unirelationale Selektion, Projektion ohne Duplikateleminierung, hash-basierte Join-Algorithmen etc.), unterliegen einige Algorithmen einer logarithmischen Kostenfunktion. So erfordern beispielsweise Sortieralgorithmen, wie sie u.a. für die Implementierung von Sort-Merge-Joins benötigt werden,

(iv) $$c \times M \times \log(M)$$

Zeiteinheiten. Hier kann die Ausführung auf *m* PE mit

(v) $$c \times \frac{M}{m} \times \log\left(\frac{M}{m}\right)$$

Zeiteinheiten einen superlinearen Speedup ermöglichen (Bild 3).

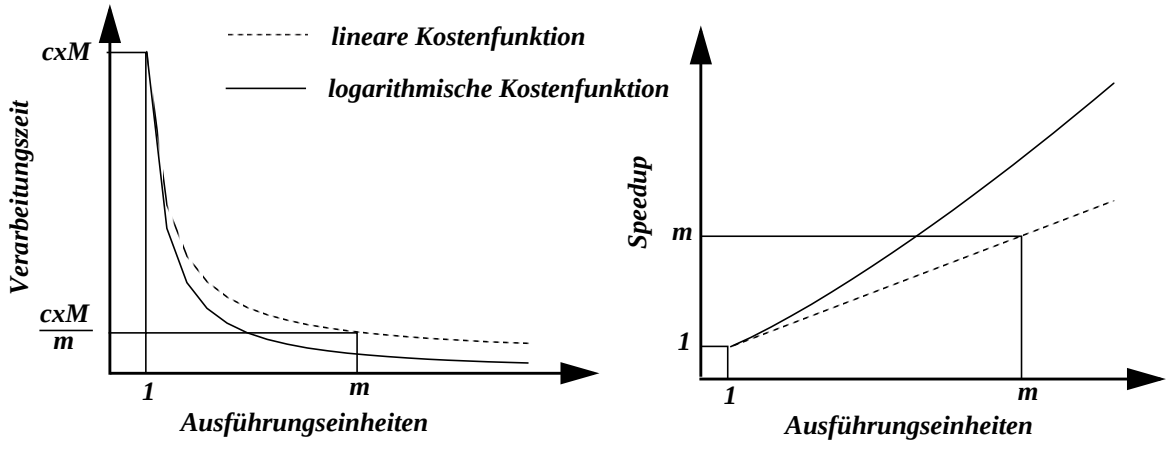


Bild 3: Antwortzeitverbesserung linearer und logarithmischer Kostenfunktionen durch Parallelisierung.

Für einen lokalen Sort-Merge-Join zwischen zwei Relationen mit je M_1 bzw. M_2 Tupeln können wir die Verarbeitungsdauer unter Berücksichtigung der Kostenterme (ii) und (iv) mit folgender Formel abschätzen:

(vi) $$c_1 \times M_1 + c_2 \times M_2 + d_1 \times M_1 \times \log(M_1) + d_2 \times M_2 \times \log(M_2)$$

Dabei ist ein 1:n-Join angenommen[3]. Die logarithmischen Terme schätzen den Sortieraufwand der

3. Bei einem 1:n-Join existiert zu jedem Tupel der zweiten Verbundrelation maximal ein Tupel der ersten Relation mit demselben Verbundattributwert. Zu jedem Tupel der ersten Relation kann es hingegen n > 0 Verbundpartner in der zweiten Relation geben.

Eingaberelationen ab; die linearen Terme beschreiben das Referenzieren der Tupeln in den sortierten temporären Eingabeströmen des Joins. Unter Berücksichtigung der Kostenterme (iii) und (v) errechnen sich die Verarbeitungskosten der verteilten Bearbeitung des Sort-Merge-Joins in *m* Ausführungseinheiten wie folgt:

(vii) $$c_1 \times \frac{M_1}{m} + c_2 \times \frac{M_2}{m} + d_1 \times \frac{M_1}{m} \times \log\left(\frac{M_1}{m}\right) + d_2 \times \frac{M_2}{m} \times \log\left(\frac{M_2}{m}\right)$$

Dabei sei der Übersicht halber angenommen, daß die Operanden lokal in den *m* Ausführungseinheiten vorliegen. In der Regel sind die Operanden vorher entsprechend (in Abhängigkeit von ihrer Herkunft und der Zuordnung der Ausführungseinheiten zu Prozessorelementen) umzuverteilen. Grundsätzlich lassen sich die Umverteilungskosten den bisher aufgezählten Kostentermen zurechnen. Eine detailliertere Betrachtung dieser Kosten werden wir in Kapitel 3 vorstellen.

Berücksichtigung von Platten-E/A:

Die bisher beschriebenen Kostenterme beinhalten diejenigen Operatorkosten, die beim Hauptspeicherzugriff der Operatoren auf deren Eingabetupeln anfallen. Liegen die Eingabetupeln nicht im Hauptspeicher vor, so müssen die betreffenden Datenobjekte vom Sekundärspeicher (i.d.R. Platte) nachgeladen werden. Referenziert ein Operator während seiner Ausführung *P* Datenbankseiten, betragen ferner die E/A-Kosten pro Seite *io* Zeiteinheiten und beträgt die Wahrscheinlichkeit, daß sich eine referenzierte Seite nicht im Hauptspeicher befindet *f* (0<=*f*<=1), so sind pro Ausführungseinheit zusätzlich zu den Hauptspeicherzugriffen

(viii) $$io \times \frac{P}{m} \times f$$

Zeiteinheiten für die Platten-E/A notwendig. Zu beachten ist hier, daß die Wahrscheinlichkeit für eine Fehlseitenbedingung, d.h. einen Plattenzugriff, von der Größe des verfügbaren Datenbankpuffers abhängig ist. Da der Datenbankpuffer wiederum mit der Anzahl *m* der Ausführungseinheiten zunimmt, verhält sich die Wahrscheinlichkeit für eine Fehlseitenbedingung umgekehrt proportional zur PE-Anzahl. Beträgt die Puffergröße eines PE *B* Seiten und umfaßt die referenzierte Relation *D* physische Seiten, so können wir (unter der Annahme, daß die Zugriffe auf alle DB-Seiten wahlfrei erfolgen und gleichverteilt sind) die Wahrscheinlichkeit *f* einer Fehlseitenbedingung mit folgender Annäherung abschätzen[4]:

(ix) $$1 - MIN\left(1, \frac{m \times B}{D}\right)$$

Unter Berücksichtigung der Terme (viii) und (ix) erkennen wir, daß mit zunehmender PE-Anzahl ei-

4. Dabei unterstellen wir Lokalität im Referenzierungsverhalten derart, daß Datenbankseiten durch mehrere Benutzer bzw. Queries gemeinsam benutzt werden (*inter transaction locality* [Härder 1987]).

nerseits die Anzahl der Seitenreferenzen pro Ausführungseinheit abnimmt, andererseits aber auch die Fehlseitenwahrscheinlichkeit aufgrund wachsender Puffergröße abnimmt. Aus diesem Grunde können wir im Falle plattenallokierter Relationen im allgemeinen eine superlineare Antwortzeitverbesserung durch Anfrageparallelisierung erwarten [Marek und Rahm 1992].

In analoger Weise zum haupspeicherbasierten, linearen Fall kann man nun die Parallelisierung logarithmischer Operatoren beschreiben und die Plattenallokation von Relationen berücksichtigen. Hierzu werden wir in Kapitel 5 Versuche durchführen. Zunächst werden wir jedoch versuchen, die Koeffizienten der Kostenformel auf analytischem Wege zu ermitteln. Zu diesem Zweck wollen wir im folgenden Kapitel unsere Sicht der parallelen Anfrageverarbeitung präzisieren.

3 Die Verfeinerung des Kostenmodells für Scan- und Join-Anfragen

Um nun die soeben hergeleitete Kostenformel effektiv nutzen zu können, müssen wir natürlich die Werte der bis jetzt noch unbekannten Koeffizienten der Formel kennen. Um zu einer adäquaten Abschätzung zu gelangen, werden wir unsere Sicht der parallelen Anfragebearbeitung präzisieren. Wir werden hierzu die Verarbeitungsschritte der parallelen Bearbeitung von Scan- und Join-Anfragen detailliert nachvollziehen und daraus eine Abschätzung der gesuchten Koeffizienten für Scan- und Join-Operatoren treffen.

3.1 Lastprofil

Die Anfragen, die wir im folgenden zugrundelegen, verwenden drei relationale Basisopertoren: *Scan*, *Sort* und *Join*. Der Scan, ausgestattet mit einem Selektionsprädikat P, auf einer Relation A generiert einen relationalen Datenstrom als Ausgabe. Dazu liest der Scan alle Tupel der Eingaberelation, wendet das Prädikat P auf jedes Tupel an und fügt das Tupel zu der Ausgabemenge hinzu, falls es das Prädikat P erfüllt. Das Lesen aller Tupel der Eingaberelation (sog. Relationen-Scan) kann vermieden werden, wenn eine Indexstruktur (z.B. B^*-Baum) den Zugriff nach dem Selektionsprädikat unterstützt. In diesem Fall werden, neben der Index-Information, nur die das Selektionsprädikat erfüllenden Tupel gelesen. Der Sort-Operator sortiert den Eingabestrom seiner Tupel entsprechend eines auf ein Attribut anzuwendendes Sortierkriteriums. Der Join-Operator verknüpft zwei Eingaberelationen A und B anhand eines Join-Attributes und erzeugt eine neue Relation. Für jedes Tupel t_a in A werden alle Tupel t_b in B gesucht, deren Join-Attributwerte dem von t_a entspricht[5]. Für jedes auf diese Weise gefundene Tupelpaar generiert der Join-Operator durch Konkatenation des Paares ein neues Tupel und fügt es dem Ausgabestrom hinzu[6]. Mit diesen Basisoperatoren können nun rechnerlokale Anfragen beschrieben werden. Im Falle von Scan-Anfragen benötigen wir lediglich einen Scan-Operator; im Falle einer Join-Anfrage generiert jeweils ein Scan-Operator einen Eingabestrom des Join-Operators.

5. Wir wollen uns hier auf Equi-Joins beschränken.
6. Durch Verknüpfung von n-1 (n > 1) dieser sogenannten 2-Wege-Join-Operatoren erhält man n-Wege-Joins.

Für die verteilte Anfragebearbeitung benötigen wir zwei weitere Operatoren: *Merge* und *Split*. Der Merge-Operator “mischt” mehrere (sortierte) parallele Datenströme in einen sequentiellen Strom. Der Split-Operator teilt einen sequentiellen Datenstrom in mehrere einzelne Datenströme auf. Mit Hilfe dieser Operatoren können wir nun die Scan- und Join-Operatoren parallelisieren.

Für die Ausführung des parallelen Joins unterstellen wir eine hash-basierte Strategie. Mit Hilfe einer Hash-Funktion, angewendet auf das Join-Attribut, verteilt der Split-Operator die Ausgabeströme der Scan-Ausführungseinheiten auf die Join-Ausführungseinheiten (dynamische Datenumverteilung). Auf diese Weise wird garantiert, daß Tupeln mit demselben Join-Attributwert auch derselben Join-Ausführungseinheit und damit demselben PE zugewiesen werden.
Zur Verdeutlichung der Arbeitsweise des Split-Operators sei in Anlehnung an [DeWitt und Gray 1992] das folgende Beispiel herangezogen: es sei ein Join angenomen, der die Relationen *A* und *B* über ein Verbundattribut *VA* miteinander verknüpft. Relation *A* sei in drei Fragmente A_0, A_1 und A_2 und Relation *B* in zwei Fragmente B_0 und B_1 partitioniert. Die Fragmente seien disjunkten Prozessorelementen zugeordnet. Der Scan auf Relation *A* wird parallel auf den Fragmenten A_0, A_1 und A_2, der Scan auf *B* analog parallel auf den Fragmenten B_0 und B_1 ausgeführt. Die Ausgabeströme eines jeden lokalen Scans werden mit Hilfe des Split-Operators, entsprechend der Anzahl der Join-Ausführungseinheiten, in separate Tupelströme aufgeteilt. Im folgenden sei angenommenen, daß der Join in drei Ausführungseinheiten bearbeitet wird, die den Prozessorelementen PE_i, PE_j und PE_k zugeordnet seien. Einfache, auf Wertebereichen des Verbundattributes basierende, Split-Operatoren[7] für die Relationen *A* und *B* können wir nun mit der folgenden Tabelle 1 beschreiben. Auf jede der drei Partitionen

Relation *A*	Scan Split-Operator	Relation *B*	Scan Split-Operator
VA-Werte	Bestimmungsort	*VA*-Werte	Bestimmungsort
"A-H"	PE_i; Prozeß p; Eingabestrom 1	"A-H"	PE_i; Prozeß p; Eingabestrom 2
"I-Q"	PE_j; Prozeß k; Eingabestrom 1	"I-Q"	PE_j; Prozeß k; Eingabestrom 2
"R-Z"	PE_k; Prozeß l; Eingabestrom 1	"R-Z"	PE_k; Prozeß l; Eingabestrom 2

Tabelle 1: Beispiel von Split-Operatoren.

von *A* wird jeweils derselbe Split-Operator angewendet, der für alle Ausgabetupeln den Bestimmungsort (Prozessorelement, Prozeß und Eingabestrom) spezifiziert. Tupeln mit *VA*-Werten von "A-H" werden zu PE_i, Tupeln mit *VA*-Werten von "I -Q" werden zu PE_j und Tupeln mit Werten von "R-Z" zu PE_k - jeweils an den ersten der beiden Join-Eingabeströme - geschickt. Analog werden die Tupeln der beiden Partitionen von *B* umverteilt, mit dem Unterschied, daß sie zu dem zweiten Join-Eingabestrom geschickt werden. Jede der Join-Ausführungseinheiten sieht je einen sequentiellen

7. In analoger Weise können mit Hilfe von beliebigen Programmen u.a. hash-basierte Split-Operatoren realisiert werden.

Eingabestrom mit Tupeln von Relation *A* und einen Strom mit Tupeln von Relation *B*.
Als lokalen Join-Algorithmus nehmen wir einen Sort-Merge-Join an. Beim Sort-Merge-Join werden zunächst beide Eingabeströme nach dem Join-Attribut sortiert. Anschließend werden Scans auf den sortierten Strömen durchgeführt und Tupelpaare mit dem gleichen Join-Attributwert dem Ausgabestrom der Ausführungseinheit zugewiesen. Das endgültige Resultat erhält man schließlich durch Mischen der Ausgabeströme aller Ausführungseinheiten in der Verwaltungseinheit.
Während für Scan-Operatoren auf den Basisrelationen der Grad an Intra-Operator-Parallelität bereits durch die Datenverteilung statisch festgelegt ist, bestehen für die parallele Join-Ausführung große Freiheitsgrade: sowohl die Anzahl der Ausführungseinheiten, als auch deren Zuordnung zu Prozessorelementen ist grundsätzlich frei wählbar.

Wir wollen im folgenden untersuchen, wie sich die Anzahl der Ausführungseinheiten eines Operators (Scan- bzw. Join-Operator) auf die Gesamtkosten der Anfrage auswirkt. Dazu schätzen wir die Anzahl der notwendigen Instruktionen (Pfadlänge) bzw. Verzögerungen (Platten-E/A-Zeiten, Nachrichtenlaufzeiten) in Abhängigkeit von der Anzahl der Ausführungseinheiten in einer Näherung ab, die die wesentlichen Antwortzeitkostenanteile erfaßt.
Eine Anfrage wird in Form einer Transaktion ausgeführt, deren einziger DML-Befehl ein Selektionsbefehl auf einer oder mehreren Relationen ist. Im ersten Fall sprechen wir von Scan-Anfragen; im letzteren Fall von Join-Anfragen. Bei Join-Anfragen beschränken wir uns auf 2-Wege-Joins, d.h. Anfragen, die zwei Relationen miteinander verknüpfen. Für unsere Kostenabschätzung wollen wir folgende Antwortzeitkomponenten heranziehen:

- Overhead für die Transaktionsverwaltung: d.h. Transaktions-Initialisierungskosten zu Beginn der Transaktion (BOT Begin Of Transaction) sowie Kosten des verteilten Commit-Protokolles bei Ende der Transaktion (EOT End Of Transaction),
- CPU-Kosten für den Zugriff auf Datenbankobjekte (Indexobjekte oder Tupeln von Basisrelationen bzw. temporären Relationen) im Hauptspeicher (z.B. Vergleich von Attributwerten, Sortieren temporärer Relationen oder Mischen von Eingabeströmen),
- CPU-Kosten für das Senden/Empfangen von Nachrichen sowie das Kopieren von Daten vom bzw. in den Hauptspeicher beim Senden/Empfangen von Aktivierungsnachrichten, Zwischenergebnissen und Commit-Nachrichten,
- Nachrichtenlaufzeiten über das Kommunikationsnetzwerk,
- sowie E/A-Kosten (CPU-Overhead, Kontroller-Belegungszeit, Seitenübertragungszeit sowie Plattenzugriffszeit) für Datenbankseiten, die nicht im Hauptspeicher verfügbar sind.

Die im folgenden verwendeten Abkürzungen sind in Tabelle 2 aufgelistet.

	Anzahl Instruktionen für:		
bot	Initialisierung der Transaktion	m	Anzahl paralleler Operatorausführungseinheiten
eot	Beendigung der Transaktion	$kard_i$	Anzahl der Tupeln in Relation i
ijoin	Initialisierung einer Join-Ausführungseinheit	ts_i	Tupelgröße von Relation i (Bytes)
iscan	Initialisierung einer Scan-Ausführungseinheit	sel_i	Scan-Selektivitätsfaktor auf Relation i
join	Tupelreferenz beim Join	$nrPE_i$	Anzahl PE, auf die Relation i verteilt ist
merge * n	Mischen von n Tupeln	h_i	Höhe der Indexstruktur auf Relation i
receive	Empfangen einer Nachricht	restuples	Anzahl der Ergebnistupel des Joins
send	Senden einer Nachricht	jrs	Größe pro Join-Ergebnistupel (Bytes)
copy	Kopieren eines Bytes von/in Hauptspeicher	srs	Größe pro Scan-Ergebnistupel (Bytes)
scan	Tupel- bzw. Indexreferenz beim Scan	$block_i$	Blockungsfaktor von Relation i
sort*n*log_2n	Sortieren von n Tupeln	buffer	Puffergröße pro PE (Seiten)
		nrPE	Anzahl der involvierten PE
		nettime	Netzübertragungszeit pro Byte
mips	CPU-Leistung pro PE	io	E/A-Verzögerung pro Seite

Tabelle 2: Notation der Kostenabschätzung.

3.2 Die parallele Verarbeitung von Scan-Anfragen

Im Falle eines unirelationalen Selektionsbefehls besteht die zugehörige Datenbankanfrage im wesentlichen aus einem Scan-Operator, der die Selektion auf der entsprechenden Basisrelation *i* ausführt. Der Grad an Intra-Operator-Parallelität wird durch die Datenverteilung statisch festgelegt, d.h., die Anzahl der Scan-Ausführungseinheiten entspricht im allgemeinen der Anzahl ($nrPE_i$) der Prozessorelemente, auf denen Partitionen der Relation allokiert sind (sofern einzelne PE - bei Übereinstimmung von Verteilattribut der Relation und Selektionsattribut der Anfrage - nicht von der Anfrage ausgenommen werden können). Wir werden im folgenden betrachten, wie sich die Anzahl $nrPE_i$ der Scan-Ausführungseinheiten auf die Antwortzeit der Anfrage auswirkt.

Transaktionsverwaltungs-Overhead

Da wir uns hier v.a. auf Aspekte der Operatorparallelisierung konzentrieren, wollen wir eine einfache Modellierung der Transaktionsverwaltungskosten wählen. Die BOT-Kosten setzen wir als konstanten Anteil (*bot*) fest. Die EOT-Behandlung schließt die Ausführung des verteilten Zwei-Phasen-Commit-Protokolls [Mohan et al. 1986, Özsu und Valduriez 1991] ein, das alle PE ($nrPE_i$) betrifft, auf denen Ausführungseinheiten der Anfrage bearbeitet wurden. Die dabei entstehenden Kommunikationskosten berücksichtigen wir explizit. Legt man die in [Mohan et al. 1986] vorgeschlagene Optimierung zugrunde, bei der rein lesende Teilanfragen lediglich an der ersten Commit-Phase teilnehmen, so ist für die Scan-Anfrage nur eine Commit-Phase vonnöten. Die Verwaltungseinheit fordert alle beteiligten PE auf, die erste Commit-Phase einzuleiten (Sperrfreigabe), und wartet auf die Bestätigung der

erfolgreichen Beendigung der verteilten Ausführungseinheiten, bevor die Transaktion endgültig beendet wird. Die Verwaltungseinheit sendet und empfängt somit $nrPE_i$ Nachrichten. Das Senden und Empfangen der Nachrichten in den verteilten Ausführungseinheiten wird überlappt durch diese Aktivitäten in der Verwaltungseinheit, so daß die Nachrichtenkosten der Ausführungseinheiten nicht in die Antwortzeit einfließen. Die lokalen EOT-Kostenanteile fassen wir in einem konstanten Anteil (*eot*) zusammen. Dieser Anteil umfaßt Kosten für lokales Logging sowie die Sperrfreigabe. Bei einer CPU-Leistung von *mips* MIPS pro PE ergibt sich folgende Bearbeitungszeit für die Transaktionsverwaltung:

(S1) $$TransManage = (bot + eot + nrPE_i \times (send + receive)) \times \frac{1}{mips}$$

Verarbeitung des Scan-Operators

Die Eingaberelation des Scans sei auf $nrPE_i$ Prozessorelemente verteilt. Für die Initialisierung der verteilten Scan-Ausführungseinheiten auf der Relation fällt pro Ausführungseinheit je eine Aktivierungsnachricht an. In jedem PE, dem eine Ausführungseinheit eines Scans zugeordnet ist, wird die Aktivierungsnachricht empfangen und die Ausführungseinheit initialisiert (*iscan*). Das Empfangen der Aktivierungsnachrichten und das Initialisieren der Scans erfolgt in den Ausführungseinheiten (durch das sequentielle Senden der Aktivierungsnachrichten zeitversetzt) parallel. Damit ergibt sich folgender Aufwand für die Initialisierung der Scans:

(S2) $$InitScan = (nrPE_i \times send + iscan + receive) \times \frac{1}{mips}$$

Bei der Berechnung der Kosten der verteilten Scan-Bearbeitung (Gleichung S3) gehen wir davon aus, daß die Tupelreferenzen eines Scans gleichmäßig auf die Ausführungseinheiten verteilt sind :

(S3) $$ProcScan = \begin{cases} \left(\frac{kard_i}{nrPE_i}\right) \times \frac{scan}{mips} + io \times \left(\frac{kard_i}{block_i \times nrPE_i}\right) \times f_{rs} & \text{Relationen-Scan} \\ \left(\frac{kard_i \times sel_i}{nrPE_i} + h_i\right) \times \frac{scan}{mips} + io \times \left(\frac{kard_i \times sel_1}{block_i \times nrPE_i}\right) \times f_{ind} & \text{geclusterter Index} \\ \left(\left(\frac{kard_i \times sel_i}{nrPE_i}\right) \times 2 + h_i\right) \times \frac{scan}{mips} + io \times \left(\frac{kard_i \times sel_i}{nrPE_i}\right) \times f_{ind} & \text{nicht-geclusterter Index} \end{cases}$$

Generell sind die Kosten der Scan-Ausführungseinheiten proportional zu der Anzahl der Tupel- bzw. Indexreferenzen. Im Falle von Relationen-Scans muß in jeder Ausführungseinheit jeweils die gesamte lokale Partition sequentiell durchlaufen werden ($kard_i/nrPE_i$ Tupeln). Dabei werden - entsprechend dem Blockungsfaktor $block_i$ der Relation - alle lokalen, d.h. genau $kard_i/(nrPE_i*block_i)$ Datenbank-

seiten referenziert. Die hierfür notwendigen E/A-Kosten schätzen wir mit *io* Zeiteinheiten pro Seite ab. Diese Konstante umfaßt sowohl den CPU-Overhead einer E/A-Operation als auch die E/A-Verzögerungszeit (Plattenzugriffszeit, Übertragungszeit und Kontroller-Belegungszeit). Die Wahrscheinlichkeit, daß sich eine referenzierte Seite nicht bereits im Hauptspeicher befindet und von Platte gelesen werden muß (Fehlseitenbedingung), betrage dabei f_{rs} (siehe unten). Kann eine Indexstruktur für den Zugriff genutzt werden, so werden - neben den Referenzen auf die Index-Struktur - lediglich die das Selektionsprädikat erfüllenden Tupeln referenziert. In jeder Scan-Ausführungseinheit wird ein Anteil von sel_i der lokalen Tupeln als Treffer bestimmt. Im Falle einer Index-Struktur, die die Tupeln nach dem Zugriffsattribut clustert (d.h. Tupel mit dem gleichen Attributwert in der selben physischen Seite ablegt), genügt ein einmaliger Durchlauf der Index-Struktur, d.h., jede Ebene wird einmal referenziert. Sind die Datenseiten untereinander verkettet, so fallen keine weiteren Indexreferenzen an, so daß die Index-Zugriffskosten proportional zur Höhe h_i der Indexstruktur sind. Liegt eine Clusterung der Datenbankobjekte nach dem Zugriffsattribut vor, so müssen lediglich $(kard_i * sel_i)/(nrPE_i * block_i)$ Datenbankseiten referenziert werden, wobei wir die Wahrscheinlichkeit einer Fehlseitenbedingung mit f_{ind} (siehe unten) beschreiben[8]. Ist der Index nicht geclustert, so muß in der Regel die unterste Ebene der Indexstruktur vor jeder Tupelreferenz referenziert werden, da hier kein Verkettung der Datenseiten untereinander genutzt werden kann. Außerdem wird im allgemeinen bei jeder Tupelreferenz eine neue Datenbankseite referenziert, und es ist mit der Wahrscheinlichkeit f_{ind} eine E/A-Operation erforderlich. In diesem Fall sind daher $f_{ind} * (kard_i * sel_i)/nrPE_i$ E/A-Vorgänge notwendig. Für die Wahrscheinlichkeit einer Fehlseitenbedingung verwenden wir folgende Abschätzung:

$$f_{rs} = \begin{cases} 1 & \text{falls } buffer < kard_i/(block_i * nrPE_i) \\ 0 & \text{falls } buffer >= kard_i/(block_i * nrPE_i) \end{cases} \qquad \text{Relationen-Scan}$$

$$f_{ind} = 1 - MIN\left(1, \frac{nrPE_i \times buffer \times block_i}{kard_i}\right) \qquad \text{Indexunterstützung}$$

Aufgrund des sequentiellen Referenzierungsmusters des Relationen-Scans sind keine Treffer zu erwarten ($f_{rs} = 1$), wenn die $kard_i/(block_i * nrPE_i)$ Datenbankseiten der $nrPE_i$ lokalen Partitionen der Basisrelation nicht in die lokalen Puffer (*buffer* Seitenrahmen) der PE passen. Sobald die lokalen Partitionen im Puffer Platz finden, ist keine E/A erforderlich ($f_{rs} = 0$). Im Falle des indexunterstützten Datenzugriffes nehmen wir eine Gleichverteilung der Seitenreferenzen auf die Datenbankseiten an, so daß die in Kapitel 2 vorgestellte Formel für wahlfreie gleichverteilte Zugriffe zur Anwendung

8. Wir berücksichtigen an dieser Stelle keine E/A von Indexseiten, da Indexseiten aufgrund ihrer hohen Referenzierungshäufigkeit zumeist von der Pufferverwaltung im Hauptspeicher gehalten werden. Lediglich für Blatt-Indexseiten ist ggf. E/A-Aufwand zu berücksichtigen. Dies kann in analoger Weise zu den E/A-Kosten von Datenobjekten geschehen.

kommt. Die Gesamtanzahl der Datenbankseiten berechnet sich als Quotient aus Kardinalität und Blockungsfaktor der Relation.

Mischen der lokalen Scan-Ergebnisströme

Nach erfolgter Bearbeitung der lokalen Scan-Ausführungseinheiten müssen die verteilten Ergebnismengen gemischt werden. Dazu sendet jede Scan-Ausführungseinheit ihre Treffertupeln in Verbindung mit einer Quittungsmeldung an die Verwaltungseinheit. Bei einer Gesamtanzahl von $kard_i*sel_i$ Ergebnistupeln nehmen wir $(kard_i*sel_i)/nrPE_i$ Ergebnistupeln pro Scan-Ausführungseinheit an. Jede Ausführungseinheit sendet somit eine Nachricht und kopiert die $(kard_i*sel_i)/nrPE_i$ lokalen Treffertupeln der Tupelgröße *srs* vom Hauptspeicher in das Kommunikationsmedium, wobei pro Byte ein Kopieraufwand von *copy* Instruktionen berechnet wird:

(S4) $$SendRes = \frac{send}{mips} + \frac{copy}{mips} \times \left(\frac{kard_i \times sel_i \times srs}{nrPE_i} \right)$$

Da die Ergebnisnachrichten unter Umständen recht umfangreich sein können, werden sie unter Umständen signifikant durch das Kommunikationsnetzwerk verzögert. Unterstellen wir eine hinreichend große Bandbreite des Netzwerkes, die eine weitestgehend kollisionsfreie Übertragung mehrerer Nachrichten erlaubt, und dauert die Übertragung eines Bytes *nettime*[9] Zeiteinheiten, so wird jede der $(kard_i*sel_i*srs)/nrPE_i$ Bytes großen Ergebnisnachrichten um folgende Dauer verzögert:

(S5) $$Transmission = \left(\frac{kard_i \times sel_i \times srs}{nrPE_i} \right) \times nettime$$

Die Verwaltungseinheit empfängt pro Ausführungseinheit eine Nachricht und kopiert alle $kard_i*sel_i$ Treffertupeln in den lokalen Hauptspeicher:

(S6) $$ReceiveRes = \frac{receive}{mips} \times nrPE_i + \frac{copy}{mips} \times kard_i \times sel_i \times srs$$

Die Ergebnisströme werden schließlich mit linearem Aufwand gemischt, wobei dieser Aufwand nur bei $nrPE_i > 1$ Ausführungseinheiten berechnet wird:

(S7) $$MergeRes = kard_i \times sel_i \times \frac{merge}{mips}$$

Der Gesamtaufwand - gemessen in Zeiteinheiten - für die Anfrage ergibt sich aus der Summe der Kostenterme (S1) bis (S7). Aus diesen Termen wollen wir nun die gesuchten Koeffizienten der Kostenformel aus Kapitel 2 ermitteln.

---

9. In der Regel werden als Übertragungsgranulat Pakete fester Größe angeboten. Beispielsweise überträgt das Netzwerk der EDS-Maschine Datenpakete der Größe 128 Bytes (zzgl. Verwaltungsinformation) in 8 Mikrosekunden [Watson und Townsend 1991].

Im allgemeinen Fall der parallelen Bearbeitung von Scan-Anfragen kommt die folgende Kostenformel zur Anwendung (vgl. Kapitel 2), die die Anfrageantwortzeit in Abhängigkeit von der Anzahl $nrPE_i$ der Scan-Ausführungseinheiten beschreibt:

$$R(nrPE_i) = a + b \times nrPE_i + \frac{c \times M}{nrPE_i} + io \times \frac{P}{nrPE_i} \times f$$

Indexobjekte wollen wir im folgenden als hauptspeicherresident annehmen. Sind auch die Datenobjekte hauptspeicherresident, so gilt für die Wahrscheinlichkeit *f* einer Fehlseitenbedingung *f*=0, so daß der letzte Term aus der Kostenformel herausfällt. Ist die Basisrelation auf Platten allokiert, so können die oben ermittelten Wahrscheinlichkeiten für Fehlseitenbedingungen f_{rs} und f_{ind} - in Abhängigkeit von der Zugriffsmethode - für *f* eingesetzt werden.
Die Anzahl *M* der Objektreferenzen pro Scan-Anfrage ist abhängig von der Zugriffsmethode. In Abhängigkeit von Zugriffsmethode und Anfrageselektivität ergeben sich folgende Werte für *M*:

$$M = \begin{cases} kard_i & \text{Relationen-Scan} \\ kard_i \times sel_i & \text{geclusterter Index} \\ kard_i \times sel_i \times 2 & \text{nicht-geclusterter Index} \end{cases}$$

Für die Anzahl *P* der Seitenreferenzen gilt entsprechend folgende Unterscheidung:

$$P = \begin{cases} \frac{kard_i}{block_i} & \text{Relationen-Scan} \\ \frac{kard_i \times sel_i}{block_i} & \text{geclusterter Index} \\ kard_i \times sel_i & \text{nicht-geclusterter Index} \end{cases}$$

Bringt man nun die Kostenterme der Gleichungen (S1) bis (S7) in die Form der obigen Kostenformel *R(m)*, so erhält man für die Koeffizienten a, b und c folgende Werte:

$$a_1 = \frac{bot + eot + iscan + receive + send + (copy \times kard_i \times sel_i \times srs)}{mips} + \begin{cases} \frac{h_i \times scan}{mips} & \text{Index} \\ 0 & \text{kein Index} \end{cases}$$

$$a = a_1 + \frac{merge}{mips} \times kard_i \times sel_i \qquad \text{für } nrPE_i > 1$$

$$b = \frac{2 \times send + 2 \times receive}{mips}$$

Werden die verteilten Ergebnisströme nur bei einer wirklichen Scan-Parallelisierung gemischt, so ergibt sich obige Unterscheidung zwischen $nrPE_i = 1$ (sequentielle Bearbeitung) und $nrPE_i > 1$ (paral-

$$c = \frac{scan}{mips} + \begin{cases} \langle \frac{copy}{mips} + nettime \rangle \times sel_i \times srs & \text{Relationen-Scan} \\ \left(\frac{copy}{mips} + nettime \right) \times srs & \text{geclusterter Index} \\ \left(\frac{copy}{mips} + nettime \right) \times \frac{srs}{2} & \text{nicht-geclusterter Index} \end{cases}$$

lele Bearbeitung) Ausführungseinheiten für den Koeffizienten *a*.
Im Zuge der Simulationsversuche in Kapitel 5 werden wir die Werte dieser Koeffizienten entsprechend der Parametrisierung des Simulationssystems errechnen und die Kostenformel validieren.

3.3 Die parallele Verarbeitung von Join-Anfragen

Join-Anfragen, die zwei Basisrelationen zu einer Ergebnismenge verknüpfen, enthalten neben dem eigentlichen Join-Operator zwei Scan-Operatoren, die die Eingabeströme des Joins generieren. Grundsätzlich sind alle drei Basisoperatoren parallelisierbar. Im Hinblick auf eine isolierte Betrachtung der Parallelisierung des Join-Operators gehen wir im folgenden von einer konstanten Datenverteilung aus, so daß der Scan-Aufwand der Anfragen konstant bleibt. Wir untersuchen, wie sich die Anzahl *m* der Join-Ausführungseinheiten auf die Antwortzeit von Join-Anfragen auswirkt.

Transaktionsverwaltungs-Overhead
Für den Overhead der Transaktionsverwaltung ergeben sich gegenüber unirelationalen Anfragen keine Unterschiede, da es sich in beiden Fällen um rein lesende Transaktionen handelt. Wir können den Kostenterm *TransManage* somit für Join-Anfragen übernehmen:

(J1) $$TransManage = (bot + eot + nrPE \times (send + receive)) \times \frac{1}{mips}$$

Die Transaktionsverwaltung schließt alle *nrPE* Prozessorelemente ein, auf denen Ausführungseinheiten von Scan-und Join-Operatoren der Anfrage bearbeitet wurden.

Verarbeitung der Scan-Operatoren
Die beiden Eingaberelationen des Joins seien auf $nrPE_1$ bzw. $nrPE_2$ Prozessorelemente verteilt. Für die Initialisierung der verteilten Scan-Ausführungseinheiten auf den beiden Relationen fällt pro Ausführungseinheit und Relation je eine Aktivierungsnachricht an. In jedem PE, dem eine Ausführungseinheit eines Scans zugeordnet ist, wird die Aktivierungsnachricht empfangen und die Ausführungseinheit initialisiert (*iscan*). Damit ergibt sich folgender Aufwand für die Initialisierung der Scans:

(J2) $$InitScan = ((nrPE_1 + nrPE_2) \times send + iscan + receive) \times \frac{1}{mips}$$

Bei der Berechnung der Kosten der verteilten Scan-Bearbeitung müssen wir beachten, in welcher Weise die Scan-Ausführungseinheiten den PE zugeordnet sind. Wir wollen hier vereinfachend anneh-

men, daß beiden Scans entweder dieselben PE zugeordnet sind, oder, daß die PE-Mengen der Scans disjunkt sind. Im ersten Fall führen diejenigen PE, denen die Scans zugeordnet sind, jeweils beide Scans aus. Unterstellen wir, daß echte Parallelverarbeitung nur auf verschiedenen PE möglich ist, so errechnen sich die Scan-Kosten aus der Summe der Kosten beider Scan-Ausführungseinheiten[10]. Werden die Scans auf disjunkten PE ausgeführt, so ist der teurere der beiden Scans kostenbestimmend (Gleichung J3). Die Scan-Kosten $Scan_i$ errechnen sich jeweils entsprechend der Kostenformel *ProcScan* (Gleichung S3) der parallelen Scan-Bearbeitung, wobei die Anzahl *m* der Scan-Ausführungseinheiten $nrPE_1$ bzw. $nrPE_2$ beträgt.

(J3)
$$ProcScan = \begin{cases} Scan_1 + Scan_2 & \text{Relationen auf denselben PE allokiert} \\ MAX(Scan_1, Scan_2) & \text{Relationen auf disjunkten PE allokiert} \end{cases}$$

Umverteilung der temporären Relationen

Die Ausgabeströme (temporäre Relationen) der Scan-Ausführungseinheiten müssen im allgemeinen gemäß der Anzahl und Auswahl der Join-Prozessoren umverteilt werden. Bei der Berechnung der Umverteilungskosten behandeln wir in unserem Modell alle Nachrichten (insbesondere auch lokale Nachrichten[11]) so, als würden sie über das Kommunikationsnetzwerk geschickt werden. Daher liegen die von uns abgeschätzten Kosten geringfügig höher als die tatsächlich entstehenden Kosten. Der Aufwand für das Verschicken der temporären Relationen ist davon abhängig, wie die Scan-Ausführungseinheiten den Prozessorelementen zugeordnet sind. Wiederum unter der Annahme identischer bzw. disjunkter PE der Ausführungseinheiten beider Scans berechnet sich der Umverteilungsaufwand aus der Summe der Umverteilungskosten beider Ausgabeströme bzw. den Kosten der teureren Umverteilung (Gleichung J4). Jedes PE, dem eine Scan-Ausführungseinheit zu-

(J4)
$$SendTempRel = \begin{cases} MAX(Send_1, Send_2) & \text{Relationen auf disjunkten PE allokiert} \\ Send_1 + Send_2 & \text{Relationen auf denselben PE allokiert} \end{cases}$$

wobei

$$Send_i = \frac{send}{mips} \times m + \underbrace{\frac{copy}{mips} \times \frac{kard_i \times sel_i \times ts_i}{nrPE_i}}_{\text{Kopierkosten}} + \underbrace{nettime \times \frac{kard_i \times sel_i \times ts_i}{nrPE_i \times m}}_{\text{Übertragungsdauer}}$$

10. Im Falle eng gekoppelter Multiprozessoren als Prozessorelemente innerhalb des Shared-Nothing-Systems könnten die beiden Ausführungseinheiten der Scans innerhalb der PE echt parallel verarbeitet werden. Die Kosten entsprächen dann dem zweiten Fall obiger Unterscheidung.

11. Als lokale Nachrichten werden Nachrichten bezeichnet, die nicht über das Kommunikationsnetz geschickt, sondern an einen Operator innerhalb desselben PE weitergegeben werden.

geordnet ist, generiert i.a. entsprechend der Anzahl der Join-Ausführungseinheiten m Ausgabeströme, d.h., jede Scan-Ausführungseinheit sendet *m* Nachrichten. Die Größe der lokalen Treffermenge einer jeden Scan-Ausführungseinheit beträgt $(kard_i * sel_i)/nrPE_i$ Tupeln, so daß jede Ausführungseinheit $(kard_i * sel_i * ts_i)/nrPE_i$ Bytes vom Hauptspeicher in die Ergebnisnachricht kopieren muß. Die Kopierkosten betragen *copy* Instruktionen pro Byte. Da sich der Ergebnisstrom auf *m* Nachrichten aufteilt, ist jede einzelne Nachricht $(kard_i * sel_i * ts_i)/(nrPE_i * m)$ Bytes lang. Die Übertragung der Nachrichten nimmt (wiederum unter der Annahme einer kollisionsfreien Übertragung) *nettime* Zeiteinheiten pro Byte in Anspruch.
Neben dem Aufwand für das Versenden der temporären Relationen fallen in den Join-Ausführungseinheiten Kosten für das Empfangen der temporären Relationen an. Hier errechnet sich der Aufwand aus der Summe für das Empfangen beider temporärer Relationen. Der Aufwand für das Empfangen

$$\text{(J5)} \quad ReceiveTempRel = \frac{receive}{mips} \times (nrPE_1 + nrPE_2) + \frac{copy}{mips} \times \left(\frac{kard_1 \times sel_1 \times ts_1 + kard_2 \times sel_2 \times ts_2}{m} \right)$$

einer temporären Relation errechnet sich aus der Anzahl der Eingabenachrichten und der Größe der Eingabemengen. In jeder Join-Ausführungseinheit treffen ebensoviele Nachrichten ein, wie Prozessorelemente die betreffenden Eingaberelationen halten ($nrPE_i$). Jede der *m* Ausführungseinheiten empfängt dabei *1/m* aller $kard_1 * sel_1 + kard_2 * sel_2$ Eingabetupeln, so daß jeweils $(kard_1 * sel_1 * ts_1 + kard_2 * sel_2 * ts_2)/m$ Bytes in die lokalen Hauptspeicher jeder Ausführungseinheit zu kopieren sind.

Lokale Join-Verarbeitung

Analog zur Initialisierung der Scan-Ausführungseinheiten müssen auch die Join-Ausführungseinheiten von der Verwaltungseinheit initialisiert werden. Dazu sendet die Verwaltungseinheit je eine Aktivierungsnachricht zu jedem PE, dem eine Join-Ausführungseinheit zugeordnet werden soll. Dort wird jeweils diese Aktivierungsnachricht empfangen und ein lokaler Join-Operator initialisiert. Hieraus ergibt sich folgende Abschätzung für den Initialisierungsaufwand:

$$\text{(J6)} \quad InitJoin = (m \times send + receive + ijoin) \times \frac{1}{mips}$$

Ist die Umverteilung der temporären Relationen erfolgt, respektive liegen in einer Join-Ausführungseinheit alle Eingabetupeln vor, so kann mit der eigentlichen Join-Verarbeitung begonnen werden. Im Falle der von uns angenommenen lokalen Sort-Merge-Strategie beginnt die Join-Ausführung mit dem Sortieren der eintreffenden Tupeln beider Relationen nach dem Join-Attribut (Gleichung J7). Für das Sortieren ist ein Aufwand von $n*log(n)$ in Abhängigkeit von der Tupelanzahl *n* vonnöten.
Nach erfolgter Sortierung werden Scans auf beiden Eingaberelationen eröffnet, die durch geeignetes

(J7) $$SortInputStreams = Sort_1 + Sort_2$$

wobei $$Sort_i = \frac{sort}{mips} \times \left(\left(\frac{kard_i \times sel_i}{m} \right) \times \log \left(\frac{kard_i \times sel_i}{m} \right) \right) \qquad i = 1,2$$

Fortschalten für jeden Wert des Join-Attributes die zugehörigen Tupeln liefern[12]. Bei einem 1:n- Join genügt ein einmaliges Referenzieren jedes Tupels, bei einem m:n- Join müssen für jeden Join-Attributwert die n Tupeln der zweiten Relation m-fach referenziert werden. In jedem Fall ist der Aufwand für diesen Teilschritt in Abhängigkeit von der Anzahl beteiligter Tupeln linear. Für unsere Zwecke nehmen wir einen 1:n-Join an, so daß wir folgende Abschätzung für die Scans auf den beiden Eingaberelationen erhalten:

(J8) $$ProcJoin = \left(\frac{kard_1 \times sel_1}{m} + \frac{kard_2 \times sel_2}{m} \right) \times \frac{join}{mips}$$

Mischen der lokalen Join-Ergebnisströme

Nach erfolgter Bearbeitung der lokalen Join-Ausführungseinheiten müssen die verteilten Ergebnismengen gemischt werden. Dazu sendet jede Join-Ausführungseinheit ihre Treffertupeln an die Verwaltungseinheit. Bei einer Gesamtanzahl von *restuples* Ergebnistupeln nehmen wir *restuples / m* Ergebnistupeln pro Join-Ausführungseinheit an. Bei einer Tupelgröße von *jrs* fällt der in Gleichung J9 beschriebene Kopieraufwand pro Join-Ausführungseinheit, und der *m*-fache Empfangsaufwand in der Verwaltungseinheit an.

(J9) $$SendRes = \underbrace{\frac{send}{mips}}_{\text{Sende- und Kopieraufwand pro Join-Ausführungseinheit}} + \underbrace{\frac{copy}{mips} \times \frac{restuples \times jrs}{m}}_{\text{Sende- und Kopieraufwand pro Join-Ausführungseinheit}} + \underbrace{\frac{receive}{mips} \times m}_{\text{Empfangs- und Kopieraufwand der Verwaltungseinheit}} + \underbrace{\frac{copy}{mips} \times restuples \times jrs}_{\text{Empfangs- und Kopieraufwand der Verwaltungseinheit}}$$

Die Übertragungszeit der jeweils *(restuples*jrs)/m* Bytes großen Nachrichten nimmt dabei

(J10) $$Transmission = nettime \times \frac{restuples \times jrs}{m}$$

Zeiteinheiten in Anspruch. Wurde der Join von $m > 1$ Ausführungseinheiten bearbeitet, nehmen wir

12. Die Tupeln der Join-Eingabeströme wurden nach deren Umverteilung zu den Join-Ausführungseinheiten in die lokalen Hauptspeicher kopiert (s.o.). Wir wollen für unser Kostenmodell vereinfachend annehmen, daß alle Eingabetupeln resident im Hauptspeicher verfügbar sind und keinerlei E/A im Rahmen der Join-Verarbeitung anfällt.

für das Mischen der Ergebnistupeln schließlich folgenden linearen Aufwand an:

(J11) $$MergeRes = restuples \times \frac{merge}{mips}$$

Der Gesamtaufwand - gemessen in Zeiteinheiten - für die Anfrage ergibt sich aus der Summe der Kostenterme (J1) bis (J11). Auch hier können wir nun die gesuchten Koeffizienten der Kostenformel aus Kapitel 2 ermitteln.

Im allgemeinen Fall der parallelen Bearbeitung von Join-Anfragen kommt die folgende Kostenformel zur Anwendung (vgl. Kapitel 2):

$$R(m) = a + b \times m + c_1 \times \frac{M_1}{m} + c_2 \times \frac{M_2}{m} + d_1 \times \frac{M_1}{m} \times \log\left(\frac{M_1}{m}\right) + d_2 \times \frac{M_2}{m} \times \log\left(\frac{M_2}{m}\right)$$

Die Anzahl M_i der Objektreferenzen pro Join-Anfrage berechnet sich aus der Selektivität der Scan-Operationen sowie der Kardinalität der Eingaberelationen:

$$M_i = kard_i \times sel_i$$

Bringt man nun die Kostenterme der Gleichungen (J1) bis (J11) in die Form der obigen Kostenformel *R(m)*, so erhält man für die Koeffizienten a, b, c und d folgende Werte:

$$a = TransManage + InitScan + ProcScan + \underbrace{\frac{copy}{mips} \times \frac{kard_1 \times set_1 \times ts_1}{nrPE_1}}_{\text{konstante Anteile aus } Send_1 \text{ und } ReceiveTempRel} + \underbrace{\frac{receive}{mips} \times (nrPE_1 + nrPE_2)}_{\text{konstante Anteile aus } InitJoin \text{ und } SendRes} + \underbrace{\frac{receive + ijoin}{mips}}_{\text{konstante Anteile aus } InitJoin \text{ und } SendRes} + \underbrace{\frac{send}{mips} + \frac{copy}{mips} \times restuples \times jrs}_{\text{konstante Anteile aus } InitJoin \text{ und } SendRes} + \underbrace{\begin{cases} \frac{merge}{mips} \times restuples & m > 1 \\ 0 & m = 1 \end{cases}}_{\text{Merge-Kosten}}$$

$$b = \frac{2 \times send + receive}{mips}$$

$$c_1 = \underbrace{\frac{copy}{mips} \times ts_1}_{\text{variabler Anteil aus } ReceiveTempRel} + \frac{join}{mips} + \underbrace{\frac{ts_1}{nrPE_1} \times nettime}_{\text{variabler Anteil aus } Send_1}$$

$$c_2 = \underbrace{\frac{copy}{mips} \times ts_2}_{\text{variabler Anteil aus } ReceiveTempRel} + \frac{join}{mips} + \underbrace{\frac{copy}{mips} \times jrs \times k}_{\text{variable Anteile aus } SendRes \text{ und } Transmission} + \underbrace{nettime \times jrs \times k}_{\text{variable Anteile aus } SendRes \text{ und } Transmission}$$

$$d_1 = d_2 = \frac{sort}{mips}$$

Dabei wurde angenommen, daß die beiden Basisrelationen auf disjunkten Prozessorelementen allokiert sind und daß der Scan auf Relation 1 mindestens so hohe Verarbeitungs- und Sendekosten verursacht wie der Scan auf Relation 2 (vgl. Gleichungen J3 und J4). Für den Koeffizienten *a* ist bezüglich des Merge-Aufwands der Ergebnistupeln zu unterscheiden, ob der Join sequentiell ($m = 1$), oder parallel ($m > 1$) verarbeitet wird. Der Merge-Aufwand fällt nur bei $m > 1$ Join-Ausführungseinheiten an. Die Anzahl der Ergebnistupeln (*restuples*) wurde zur Berechnung der Koeffizienten c_i als ein Vielfaches der Eingabetupeln von Relation 2 abgeschätzt ($k*kard_2*sel_2$).
Im Zuge der Simulationsversuche in Kapitel 5 werden wir die Werte dieser Koeffizienten errechnen und die Kostenformel der parallelen Join-Verarbeitung validieren.

4 Simulationsmodell

Für Leistungsanalysen der parallelen Anfragebearbeitung haben wir in einem umfassenden Simulationsmodell ein generisches Shared-Nothing-System hinsichtlich seiner Architektur und seiner Verarbeitungskonzepte modelliert. Das System wurde mit Hilfe des Simulationswerkzeugs *DeNet* [Livny 1989] implementiert.
In vorhergehenden Simulationsstudien ([Marek und Rahm 1992] und [Marek und Rahm 1993]) wurde bereits darauf verwiesen, daß das Simulationssystem das Verhalten realer Systeme hinreichend gut annähert, so daß wir von einer korrekten Modellierung des Simulationsansatzes ausgehen können. Da uns erst die Übereinstimmung des Simulationsansatzes mit realen Systemen die Legitimation verleiht, das vorliegende Kostenmodell mit Hilfe des Simulationssystems - stellvertretend für ein reales System - zu validieren, wollen wir den Aspekt der korrekten Systemmodellierung am Ende dieses Kapitels genauer betrachten (Kapitel 4.3).
Das Simulationssystem besteht aus den wesentlichen Komponenten der Lasterzeugung, -verteilung und -verarbeitung (Bild 4). Die Lasterzeugung simuliert Benutzerterminals und generiert Benutzeranfragen, die von der Lastverteilungskomponente auf die Prozessorelemente des Verarbeitungssystems verteilt werden. Im folgenden stellen wir diese Komponenten im Überblick vor.

4.1 Lastgenerierung und -verteilung

Datenbankmodell

Das von uns zugrunde gelegte Datenbankmodell unterstützt vier Objektgranularitäten: Datenbank, Partitionen, Seiten und DB-Objekte (Tupel). Die Datenbank besteht aus einer Menge von Partitionen, welche z.B. Relationen, Relationenfragmente oder Indexstrukturen repräsentieren können. Eine Partition besteht aus einer Menge von Seiten, die jeweils eine bestimmt Anzahl von Tupeln enthalten. Für jede Relation kann eine geclusterte oder nicht-geclusterte Indexstruktur definiert werden.
Wir unterstützen eine horizontale Datenverteilung von Partitionen (Relationen und Indexstrukturen) auf der Objektebene. Definiert wird die Datenverteilung mit Hilfe einer relativen Verteiltabelle, die

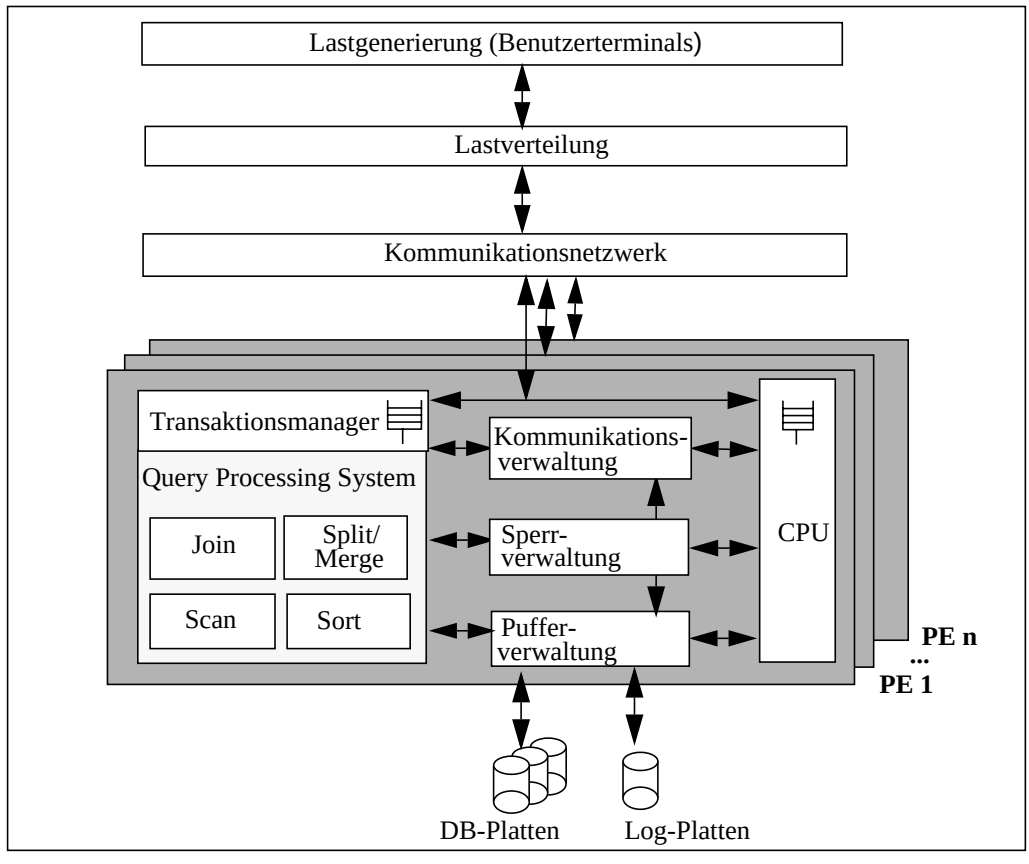


Bild 4: Grobaufbau des Simulationssystems.

für jede Partition P_j und jedes Prozessorelement PE_i definiert, welcher Anteil von P_j auf PE_i allokiert ist. Dieser Ansatz modelliert eine Datenverteilung basierend auf Wertebereichen von Attributen (*Range Partitioning*) und unterstützt sowohl die Verteilung einer Partition auf alle PE (*Full Declustering*) als auch auf eine Teilmenge der PE (*Partial Declustering*) [Özsu und Valduriez 1991].

Lastmodell

Unser Simulationsmodell unterstützt heterogene, d.h. aus mehreren verschiedenen Transaktions- bzw. Anfragetypen zusammengesetzte Anwendungslasten. Transaktionstypen können sich in der Struktur ihrer Operatorbäume, referenzierten Relationen, Selektionsprädikaten etc. unterscheiden. In diesem Bericht betrachten wir unirelationale Anfragen sowie Join-Anfragen auf zwei Relationen. Beiden Anfragetypen ist gemeinsam, daß sie jeweils nur einen einzigen DML-Befehl (Selektionsoperation) enthalten.

Für die Anfragegenerierung legen wir das in Kapitel 3.1 vorgestellte Lastprofil zugrunde. Die Basisoperatoren werden zu Operatorbäumen zusammengefügt, die unirelationale Anfragen bzw. 2-Wege-Joins beschreiben. Für die Zuordnung von Operatorausführungseinheiten zu Prozessorelementen stehen verschiedene Strategien zur Verfügung. Für Scan-Operatoren erfolgt die PE-Zuordnung stets in

Abhängigkeit von der Datenverteilung der referenzierten Relationen. Für die PE-Zuordnung zu Join-Ausführungseinheiten stehen diverse statische und dynamische Strategien zur Verfügung (z.B. wahlfrei oder basierend auf der CPU-Auslastung) [Rahm und Marek 1993].

Lastverteilung
Generell ist zwischen zwei verschiedenen Ebenen der Lastverteilung zu unterscheiden. Zunächst ist jede neu eintreffende Anfrage einem Prozessorelement zuzuteilen, das die Koordination der (verteilten) Verarbeitung dieser Anfrage übernimmt. Diese Strategie bestimmt die Zuordnung der Verwaltungseinheiten zu PE. Als Verteilungsstrategie ganzer Anfragen bietet unser System verschiedene Alternativen wie z.b. wahlfreie Zuordnung oder Routing-Tabellen[13]. Die zweite Ebene der Lastverteilung bezieht sich auf die Zuordnung von Ausführungseinheiten zu PE (siehe oben).

4.2 Lastverarbeitung

Die Lastverarbeitungskomponente modelliert die Verarbeitung von Benutzeranfragen und Transaktionen in einem Shared-Nothing-System mit einer (beliebigen) Anzahl von Prozessorelementen, die über ein Kommunikationsnetzwerk miteinander verbunden sind. Jedes PE hat Zugriff auf private Datenbank- und Log-Dateien, allokiert auf externen Speichermedien (Platten). Intern wird jedes PE repräsentiert durch einen Transaktionsverwalter, eine Anfrageverarbeitungskomponente (Query Processing System), eine Pufferverwaltung, eine Sperrverwaltung, eine Kommunikationsverwaltung sowie einen CPU-Server (Bild 4).
Der Transaktionsverwalter kontrolliert die (verteilte) Ausführung von Transaktionen. Die Anfrageverarbeitungskomponente modelliert relationale Basisoperatoren (Sort, Scan und Join) sowie die Metaoperatoren *Merge* und *Split* zur Unterstützung von Intra-Query-Parallelität.

Die Verarbeitung einer Transaktion beginnt mit der Bearbeitung des Transaktionsinitialisierungs-Overheads. Bei der eigentlichen Anfragebearbeitung werden die in den relationalen Anfragebäumen enthaltenen Operatoren bearbeitet. Prinzipiell verarbeiten die relationalen Operatoren lokale Eingabeströme (Relationenfragmente, Zwischenresultate) und produzieren Ausgabeströme. In die Commit-Behandlung der Transaktionen werden alle PE einbezogen, die bei der Bearbeitung der betreffenden Transaktion mitgewirkt haben. Hierzu haben ein Zwei-Phasen-Commit-Protokoll mit der in [Mohan et al. 1986] vorgeschlagenen Optimierung für reine Lese-Ausführungseinheiten implementiert.

CPU-Anforderungen der Transaktionen werden durch je einen CPU-Server pro PE bedient. Die mittlere Anzahl von Instruktionen pro Anforderung kann für verschiedene Anforderungstypen separat definiert werden. Damit eine realitätsnahe Modellierung der Anfragekosten gewährleistet ist, wird CPU-Aufwand für alle wesentlichen Schritte der Anfragebearbeitung angefordert, insbesondere für Transaktionsinitialisierung, Objektzugriffe im Hauptspeicher (z.B. für den Vergleich von Attributwerten,

13. Eine Routing-Tabelle spezifiziert für jeden Transaktionstyp T_j und Prozessorelement PE_i, welcher Anteil der Transaktionen vom Typ T_j PE_i zugeteilt wird.

für das Sortieren temporärer Relationen etc.), E/A-Overhead, Kommunikations-Overhead und Commit-Verarbeitung.

Datenbankpartitionen können (zur Simulation von Hauptspeicher-Datenbanken) resident im Hauptspeicher gehalten, oder auf Platten allokiert werden. Platten und Platten-Kontroller wurden explizit als Server modelliert, so daß im E/A-System auftretende Wartezeiten berücksichtigt werden. Der Zugriff auf die Daten erfolgt über die Pufferverwaltung, die die DB-Seiten im Hauptspeicher auf Anforderung bereitstellt. Als Seitenersetzungsstrategie innerhalb der Hauptspeicher wird LRU (Least Recently Used) verwendet.

Die Kommunikation bei der Transaktionsverarbeitung erfolgt über das Kommunikationsnetzwerk, das als Basisfunktion die Übermittlung von Datenpaketen fester Größe anbietet.

4.3 Validierung des Simulationssystems gegen reale Shared-Nothing-Systeme

In vorhergehenden Simulationsstudien konnte ein qualitativ gleichwertiges Verhalten des Simulationssystems zu realen Systemen für unterschiedliche Lasten beobachtet werden. Die Übereinstimmungen bezogen sich auf die beiden Leistungsziele Antwortzeit und Durchsatz (Transaktionen pro Sekunde).

Durchsatzversuche mit OLTP-Lasten:
Im Rahmen von Durchsatzversuchen zu Online Transaction Processing (OLTP) diente in unserem Simulationsversuchen die Debit-Credit-Transaktionslast [Gray 1991] als Grundlage. Diese Last konstituiert den Standard-Benchmark für OLTP-Anwendungen [Rahm 1993b]. In [Marek und Rahm 1992] wurde für ein Shared-Nothing-System eine lineare Durchsatzsteigerung für Debit-Credit-Transaktionen mit der Anzahl der Prozessorelemente gezeigt. Tandem demonstrierte eine gleichsam lineare Leistungscharakteristik (wenngleich auf einem niedrigeren Durchsatzniveau) für Debit-Credit-Transaktionen unter NonStop SQL [The Tandem Database Group 1988]. Angesichts der Debit-Credit-Leistungsvorhersagen für den Shared-Nothing-Prototypen EDS - [Watson und Townsend 1991] prognostiziert 12.000 Transaktionen pro Sekunde bei 256 Rechnerknoten und 30% Auslastung - erscheint auch das von uns auf simulativem Wege ermittelte Leistungsniveau (10.000 Transaktionen pro Sekunde bei 64 PE und 90% Auslastung) realistisch.

Antwortzeitversuche mit relationalen Lasten:
Wichtiger für die vorliegende Arbeit sind Antwortzeitversuche zur parallelen Bearbeitung von relationalen Anfragen. In [Marek und Rahm 1992] wurden Versuche zur Parallelisierung von Scan-Anfragen durchgeführt. Insbesondere wurde die Effektivität der Parallelisierung in Abhängigkeit von den Einflußgrößen Zugriffsmethode und Anfrageselektivtät betrachtet. Das beobachtete Verhalten deckt sich weitestgehend mit Ergebnissen von Messungen an dem (plattenbasierten) Shared-Nothing-System Gamma [DeWitt et al. 1990]. Die Gemeinsamkeiten zwischen unserem Simulationsansatz und den in Gamma gemessenen Werten beziehen sich u.a. auf den Antwortzeit-Speedup und das re-

lative Antwortzeitverhalten von parallel bearbeiteten Scan-Anfragen beim Zugriff mit verschiedenen Zugriffsmethoden und Selektivitäten. Abweichungen sind in den absoluten Antwortzeitergebnissen gegeben - die absoluten Abweichungen sind jedoch ausschließlich eine Frage der Parametrisierung des Simulationssystems.
Speziell für die vorliegende Arbeit haben wir zur Validierung des Simulationssystems weitere Vergleichssimulationen zu einem realen System durchgeführt. Bild 5 zeigt einen Antwortzeitvergleich

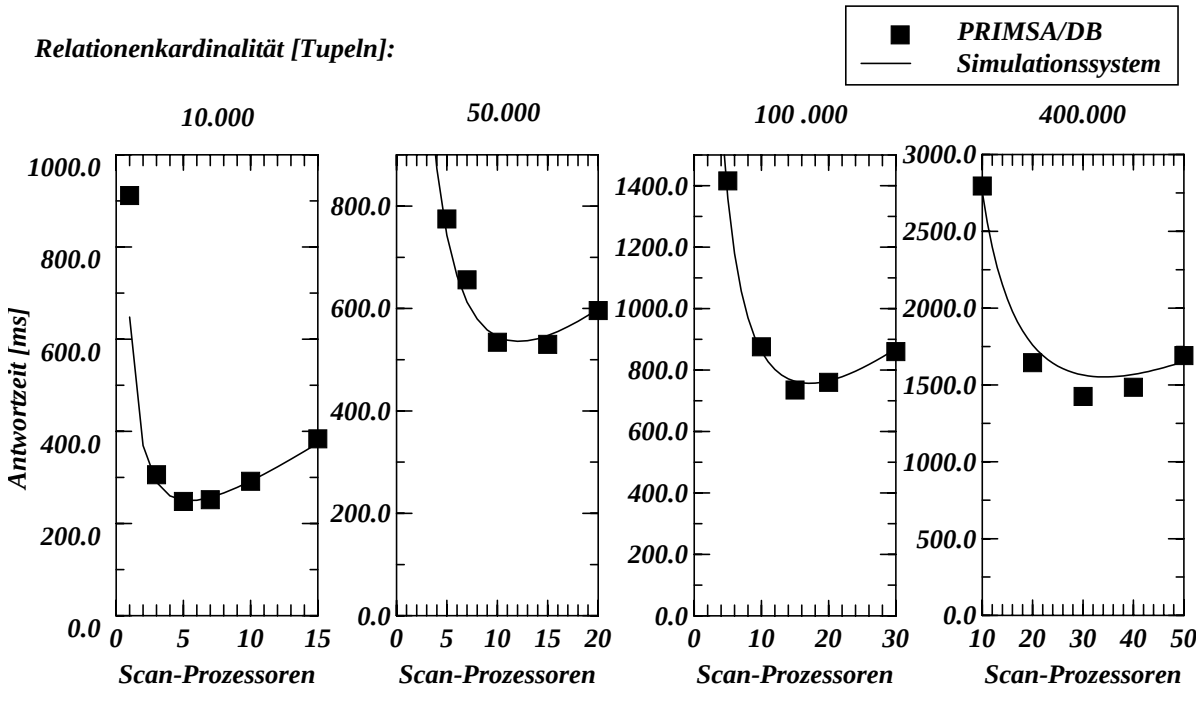


Bild 5: Scan-Anfragen: Vergleich zwischen PRISMA/DB und dem Simulationssystem.

der parallelen Bearbeitung von Scan-Anfragen zwischen dem hauptspeicherbasierten Shared-Nothing-Prototypen PRISMA/DB [Wilschut et al. 1992] und unserem Simulationssystem. Der Datenzugriff erfolgt in dieser Versuchsreihe ohne Indexunterstützung. Variiert wurde die Kardinalität der Eingaberelation zwischen 10.000 und 400.000 Tupeln. Die Ergebnismenge der Scans umfaßt jeweils 1% der Tupeln der Relationen. Wir erkennen, daß sich das Verhalten von PRISMA/DB - insbesondere in quantitativer Hinsicht - durch unseren Simulationsansatz recht gut annähern läßt. Wir dürfen daher annehmen, daß unser Simulationssystem die parallele Bearbeitung von Scan-Anfragen in Shared-Nothing-Systemen hinreichend genau modelliert.

Zur Validierung der im Simulationssystem modellierten hash-basierten Join-Strategie sei der folgende Vergleich des Simulationssystems mit dem plattenbasierten Shared-Nothing-Prototypen Gamma [DeWitt et al. 1990] angeführt (Bild 6). Es werden jeweils zwei Fälle betrachtet, die sich darin unterscheiden, ob die Eingaberelationen nach dem Join-Attribut partitioniert sind oder nicht. Bei einer Par-

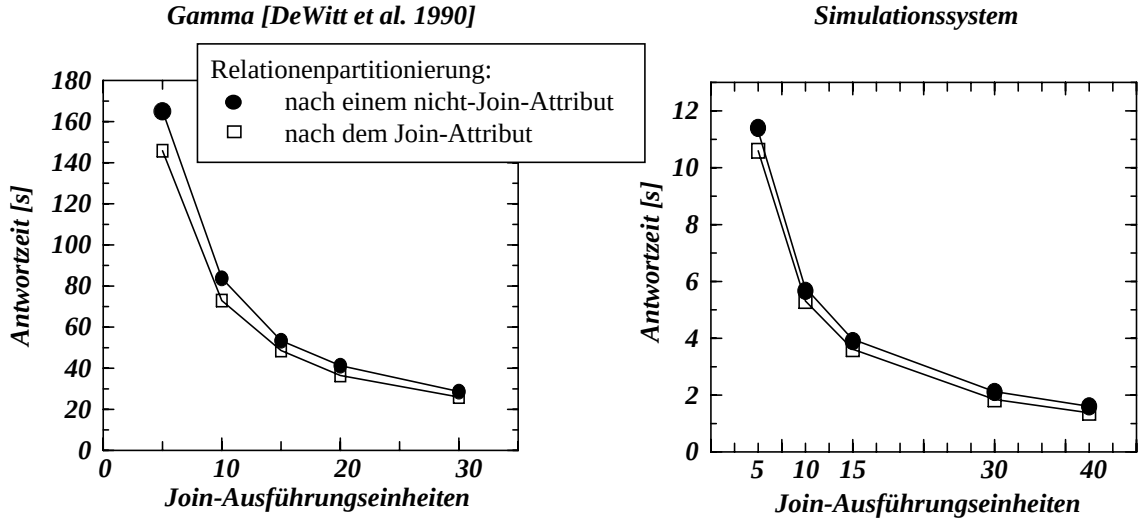


Bild 6: Hash-basierte, parallele Join-Verarbeitung: Vergleich zwischen Gamma und dem Simulationssystem.

titionierung der Relationen ist keine Datenumverteilung notwendig; im anderen Fall müssen beide Relationen mit Hilfe einer auf dem Join-Attribut anzuwendenden Hash-Funktion auf die Join-Ausführungseinheiten umverteilt werden. Unterschiede zwischen unseren simulierten und den in Gamma gemessenen Ergebnissen sind auch hier lediglich quantitativer Natur. Die absoluten Antwortzeitunterschiede (ca. Faktor 14) sind durch das voneinander abweichende Mengengerüst der Join-Untersuchungen zu erklären. Bei den Gamma-Messungen nahmen 1.100.000 Tupeln, bei unseren Simulationen nur 110.000 Tupeln am Verbund teil (Faktor 10)[14]. Sowohl bei Gamma, als auch bei unseren Simulationen wurde jedoch das gleiche relative Antwortzeitverhalten der beiden Join-Varianten untereinander beobachtet und beinahe linearer Speedup in allen Fällen erzielt. Der lineare Speedup ist zum Teil durch die hohe Anzahl von Tupelreferenzen der Joins, vor allem aber durch den hohen E/A-Aufwand bei der Join-Berechnung bedingt [Marek und Rahm 1993]. Die qualitativen Übereinstimmungen zwischen den in Gamma gemessenen und den von uns simulierten Antwortzeiten deutet auf eine korrekte Modellierung sowohl der hash-basierten Join-Strategie als auch der anfallenden E/A-Kosten hin.

Insgesamt bietet das Simulationssystem somit eine hinreichend genaue Annäherung an das Verhalten realer Systeme bei der parallelen Anfragebearbeitung, so daß wir uns bei der nun folgenden Validierung des Kostenmodelles auf unser Simulationssystem - stellvertretend für ein reales System - beziehen dürfen.

14. In dieser Größenordnung (Verbundoperation mit mehr als 1.000.000 Tupeln) stößt das Simulationssystem an seine Grenzen. Die Simulationen werden sehr platz- und zeitaufwendig, so daß wir uns auf einfachere Konfigurationen beschränken.

5 Validierung des Kostenmodells paralleler Anfragebearbeitung

In diesem Kapitel untersuchen wir das Leistungsverhalten der parallelen Verarbeitung von unirelationalen Anfragen und 2-Wege-Joins. Unser Ziel ist es zu überprüfen, wie gut das entwickelte Kostenmodell das beobachtete Leistungsverhalten annähert. Wir geben in Kapitel 5.1 einen Überblick über die Parametrisierung des Shared-Nothing-Modelles und der Datenbankanfragen. Aus den Simulationsparametern berechnen wir die Koeffizienten der Kostenformeln für Scan- und Join-Anfragen und präsentieren schließlich in den Kapiteln 5.2 und 5.3 die Ergebnisse der durchgeführten Simulationsversuche.

5.1 Parametrisierung der Simulationsversuche

Tabelle 3 faßt die wesentlichen Einstellungen der Datenbank- sowie der Anfrage- und Systemparameter zusammen. Die meisten Parameter sind selbsterklärend; einige werden wir bei der Diskussion der Ergebnisse erläutern. Die unirelationalen Anfragen führen je einen Scan auf der Eingaberelation C (250.000 Tupeln) durch. Wir untersuchen, wie sich der Parallelisierungsgrad des Scan-Operators auf die Anfrageantwortzeit in Abhängigkeit von der Anfrageselektivität, der Zugriffsmethode sowie dem Speichermedium (Hauptspeicher bzw. Platte) auswirkt. Dazu variieren wir die Datenverteilung von Relation C in den Versuchen mit unirelationalen Anfragen zwischen 1 und 40 PE. Wir vergleichen den sequentiellen Zugriff (Relationen-Scan) mit einem Index-unterstützten Zugriff. Für jede lokale Datenpartition nehmen wir einen B*-Baum mit je zwei Ebenen an. Wir untersuchen sowohl nach dem Zugriffsattribut geclusterte, als auch nach einem anderen als dem Zugriffsattribut geclusterte Zugriffstrukturen.
Wir betrachten hier Anfragen, die auf dem Anfrageprofil und dem Datenbankschema des Wisconsin-Benchmarks basieren [Gray 1991]. Dieser Benchmark wurde häufig für Leistungsanalysen paralleler Datenbanksysteme herangezogen [Englert et al. 1990, DeWitt et al. 1990, Wilschut et al. 1992]. Die Join-Anfragen in diesem Bericht entsprechen weitgehend der *joinABprime* Anfrage des Wisconsin Benchmarks. Jede Anfrage führt Scans (in Verbindung mit Selektionen) auf den beiden Eingaberelationen A und B durch und verknüpft deren Ausgabeströme mit Hilfe des Joins. Relation A enthält 1 Million Tupeln, Relation B 250.000 Tupeln. Entsprechend der Schärfe der Selektionsprädikate wählen die Selektionen auf den Relationen A und B bei Ausführung der Scans jeweils einen vorgegebenen Anteil (Selektionsfaktor) der Eingabetupeln aus, der die Ergebnismenge der Scans bildet. Beide Selektionen werden beim Datenzugriff durch eine Indexstrukur (B*-Baum), geclustert nach dem Join-Attribut, unterstützt. Die Ergebnismenge des Joins hat die gleiche Größe wie die Ausgabemenge des Scan auf Relation B. Die Selektivitätsfaktoren wurden einheitlich für beide Eingaberelationen zwischen 0.1% und 10% variiert, so daß die Ergebnismenge zwischen 250 und 25.000 Tupeln umfaßt. Die Anzahl der Join-Ausführungseinheiten bzw. Join-Prozessoren wurde zwischen 1 und 40 variiert. Die Zuordnung der Join-Ausführungseinheiten zu PE erfolgt, ebenso wie die Zuordnung neu eintref-

Systemkonfiguration	Parameterwerte	Datenbank/Anfragen	Parameterwerte
Anzahl PE	40	**Relation A:**	(200MB)
CPU-Leistung	20 MIPS	Anzahl Tupeln	1.000.000
		Tupelgröße	200 Bytes
Mittl. Anzahl Instruktionen:		Tupeln pro Seite	40
BOT	25000	Indextyp	(geclust.) B^*-Baum
EOT	25000	Höhe der Indexstruktur	2
Initialisierung von Scan-/ und		Speichermedium	Hauptspeicher/Platte
Join-Ausführungseinheiten	25000	PE-Zuordnung	32 PE (1..32)
E/A	3000	**Relation B:**	(50MB)
Nachricht Senden	5000	Anzahl Tupeln	250.000
Nachricht empfangen	10000	Tupelgröße	200 Bytes
Kopieren einer 8 KByte		Tupeln pro Seite	40
Nachricht in Hauptspeicher	5000	Indextyp	(geclust.) B^*-Baum
Scan-Objektreferenz	1000	Höhe der Indexstruktur	2
Join-Objektreferenz	500	Speichermedium	Hauptspeicher/Platte
Sortieren von n Tupeln	$n \log_2(n) * 10$	PE-Zuordnung	8 PE (33..40)
Mischen von n Tupeln	n * 100	**Relation C:**	wie Rel. B, jedoch:
		PE-Zuordnung	1-40 PE
Pufferverwaltung:		**Scan-Anfragen**	auf Relation C
Seitengröße	8 KB	Zugriffsmethode	Index/ sequentiell
Puffergröße pro PE	250 Seiten (2MB)	Selektivität	0.1%-10% (je Relation)
		Scan-Ausführungseinheiten	1-40
Plattengeräte:		**Join-Anfragen:**	Verbund von A und B
Kontroller-Belegungszeit	1 ms (pro Seite)	Zugriffsmethode	Index
Übertragungszeit pro Seite	0.4 ms	Sortierung nach dem Join-Attribut	FALSE
mittl. Plattenzugriffszeit	15 ms	Scan-Selektivität	0.1%-10%
		Anzahl der Ergebnistupeln	100-10000
		Größe pro Ergebnistupel	400 Bytes
Kommunikationsnetzwerk:		Join-Ausführungseinheiten	1-40
Paketgröße	128 Bytes	Scan-Ausführungseinheiten	Rel. A: 32; Rel. B: 8
mittlere Übertragungszeit	8 Mikrosekunden	Ankunftsrate	Einbenutzerbetrieb
		Anfragezuordnung zu PE	wahlfrei (gleichmässig über alle PE

Tabelle 3: Datenbank-, Anfrage- und Systemparameter.

fender Anfragen, wahlfrei. Die Eingaberelationen sind entsprechend ihrer Kardinalitäten in jeweils soviele Fragmente aufgeteilt, daß alle Fragmente die gleiche Anzahl von Tupeln enthalten. Relation A ist in 32, Relation B in 8 Fragmente partitioniert. Alle Fragmente sind auf disjunkten PE allokiert, so daß beide Scans der Join-Anfrage echt parallel bearbeitet werden. Bei den Versuchen mit Join-Anfragen ist Relation A auf die PE 1 bis 32, Relation B auf die PE 33 bis 40 allokiert.

Die Parameter des E/A-Systems wurden so gewählt, daß keine Engpässe auftraten (genügend große Anzahl von Platten und Kontrollern). Die Dauer einer E/A-Operation setzt sich zusammen aus der Kontroller-Belegungszeit, der Platten-Zugriffszeit sowie der Übertragungszeit. Die Parameter des Kommunikationsnetzwerkes wurden entsprechend dem EDS-Prototypen [Watson und Townsend 1991] festgelegt.

5.2 Versuchsreihen zur parallelen Scan-Verarbeitung

In dieser Versuchsreihe untersuchen wir die Effektivität der parallelen Verarbeitung von Scan-Operatoren. Wir variieren den Parallelitätsgrad des Scan-Operators in Abhängigkeit von der Datenverteilung der Eingaberelation zwischen 1 und 40 PE. Insbesondere betrachten wir, wie sich die Koeffizienten der Kostenformel *R* in Abhängigkeit von Zugriffsmethode, Anfrageselektivität und Speichermedium der Eingaberelation verhalten. Wir gehen zunächst von einer haupspeicherresidenten Speicherung der Eingaberelation aus, und untersuchen anschließend, wie sich die Plattenallokation auf die Anfrageparallelisierung auswirkt.

Im Falle von Scan-Anfragen unterliegt die Antwortzeit in Abhängigkeit von der Anzahl *m* der Scan-Ausführungseinheiten folgender Kostenformel für Operatoren mit (zur Anzahl ihrer Eingabetupeln) linearen Verarbeitungskosten (vgl. Kapitel 3):

$$R(1) = a_1 + b + c \times M + io \times P \times f \quad \text{für } m = 1, \text{ und} \qquad R(m) = a + b \times m + \frac{c \times M}{m} + io \times \frac{P}{m} \times f \quad \text{für } m > 2$$

Durch Einsetzen der Simulationsparameter in die zugehörigen Gleichungen aus Kapitel 3 erhalten wir die Werte der Koeffizienten dieser Kostenformel. Tabelle 4 zeigt die Koeffizienten in Abhängigkeit

Zugriffsmethode	Selektivität [%]	M	a1 [ms]	a [ms]	b [ms]	c [ms]	P	io [ms]
Index (geclustert)	0.1	250	6.16	7.41	1.5	0.06875	7	16.55
	1.0	2500	20.225	32.725			63	
	10.0	25000	160.85	285.85			625	
Index (nicht geclustert)	0.1	500	6.16	7.41		0.05937	250	
	1.0	5000	20.225	32.725			2500	
	10.0	50000	160.85	285.85			25000	
Relationen-Scan	1.0	250000	20.125	32.625		0.05018	6250	

Tabelle 4: Koeffizientenwerte der Kostenformel *R*.

von Zugriffsmethode und Selektivität. Für die Wahrscheinlichkeit *f* einer Fehlseitenbedingung gilt *f*=0 im Falle einer hauptspeicherresidenten Allokation der Basisrelation. Ist die Basisrelation auf Platten allokiert, so gilt in Abhängigkeit von der Zugriffsmethode:

$$f = \begin{cases} 1 & \text{falls } m < 25 \\ 0 & \text{falls } m >= 25 \end{cases} \qquad \text{für Relationen-Scans}$$

$$f = 1 - MIN(1, m \times 0.4) \qquad \text{bei Index-Unterstützung}$$

Die lokale Puffergröße beträgt in unserem Versuchen 250 Seiten. Die Tupeln von Relation C sind auf 6250 Seiten verteilt (Blockungsfaktor 40). Für 25 oder mehr PE paßt Relation C somit vollständig in die verteilten Puffer der PE, so daß die Tupelreferenzen ausschließlich durch Hauptspeicherzugriffe bearbeitet werden können. Die Zugriffskosten pro Datenbankseite betragen für alle Versuchsreihen konstant 16.55 ms.

Hauptspeicherresidente Allokation der Eingaberelation

Bild 8 zeigt die beobachteten Antwortzeiten der parallelen Scan-Verarbeitung in Abhängigkeit von der Zugriffsmethode und Anfrageselektivität und vergleicht sie mit den aus der Kostenformel *R* resultierenden Antwortzeitkurven. Wir erkennen, daß die Kostenformel für alle Versuchsreihen eine sehr gute Näherung der beobachteten Antwortzeiten bietet. Sowohl die qualitative Entwicklung der Antwortzeiten, als auch die absoluten Antwortzeitwerte stimmen weitgehend überein.

Wir erkennen, daß die Anfrageparallelisierung v.a. für diejenigen Anfragen deutliche Antwortzeitverbesserungen bietet, die hohe Einprozessor-Antwortzeiten (sequentielle Verarbeitung) aufweisen. Ein geringes Parallelisierungspotential ist dagegen bei Indexunterstützung und hoher Anfrageselektivität vorhanden. In diesen Fällen erweist sich nur eine geringe Anzahl von Ausführungseinheiten als sinnvoll. Bei zu hoher Parallelisierung sind infolge des Kommunikations-Overheads sogar starke Antwortzeitverschlechterungen zu beobachten[15]. Dieses Verhalten schlägt sich auch in den zugehörigen Speedup-Kurven nieder (Bild 9). Lediglich im Falle des Relationen-Scans kann über eine große PE-Anzahl hinweg nahezu linearer Speedup erzielt werden (Speedup 19 bei 20 PE bzw. 31 bei 40 PE). Bei Indexunterstützung fallen weitaus weniger Objektreferenzen an, so daß hier aufgrund des geringeren Parallelisierungspotentials nur wenige PE effektiv genutzt werden können. Bei hoher Anfrageselektivität (0.1%) führt nicht einmal die Parallelisierung auf 2 PE zu einer linearen Antwortzeitverbesserung (Speedup 1.2 bei einem nach dem Zugriffsattribut geclusterten Index und 1.45 ohne Clusterung).

Mit Hilfe der ermittelten Koeffizienten können wir durch Ableitung der Kostenformel *R* für jede der Versuchsreihen denjenigen Parallelisierungsgrad *m* errechnen, der das Antwortzeitoptimum erzielt:

$$m_{opt} = \begin{cases} \sqrt{\frac{0.05018 \times M}{1.5}} & \text{Relationen-Scan (1.0\% Selekt.)} \\ \sqrt{\frac{0.06875 \times M}{1.5}} & \text{Index (geclustert)} \\ \sqrt{\frac{0.05937 \times M}{1.5}} & \text{Index (nicht geclustert)} \end{cases}$$

Tabelle 5 faßt die auf diesem Wege ermittelten optimalen Parallelisierungsgrade des Scan-Operators bei hauptspeicherresidenter Speicherung der Eingaberelation zusammen.

Ist das typische Anfrageprofil (Anfragetypen und deren Anteile an der gesamten Anfragelast) auf ei-

15. Eine detaillierte Diskussion findet der Leser in [Marek und Rahm 1992].

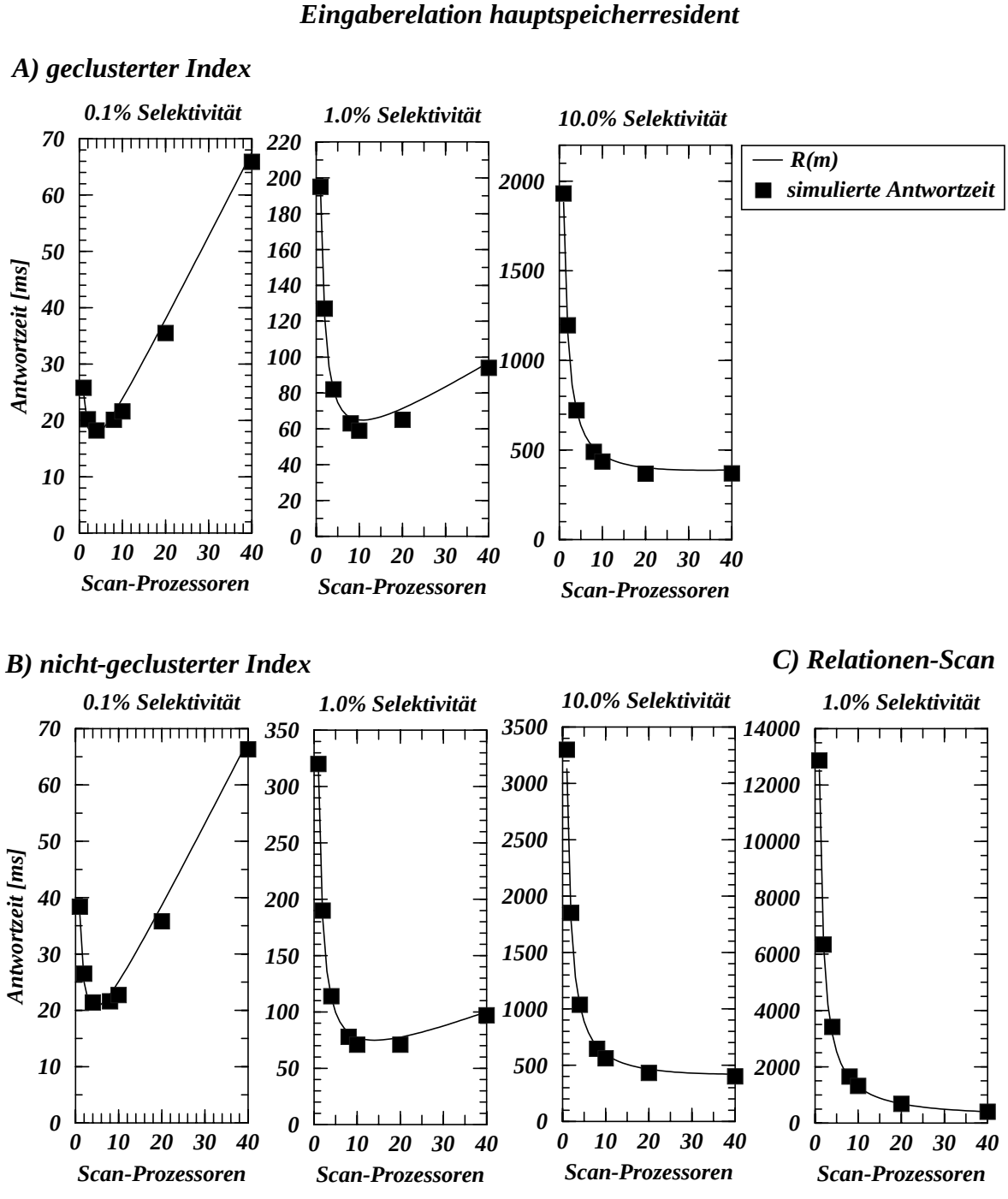


Bild 8: Einfluß von Zugriffsmethode und Anfrageselektivität auf die Effektivität der Parallelisierung von Scan-Operatoren bei hauptspeicherresidenter Speicherung der Eingaberelation.

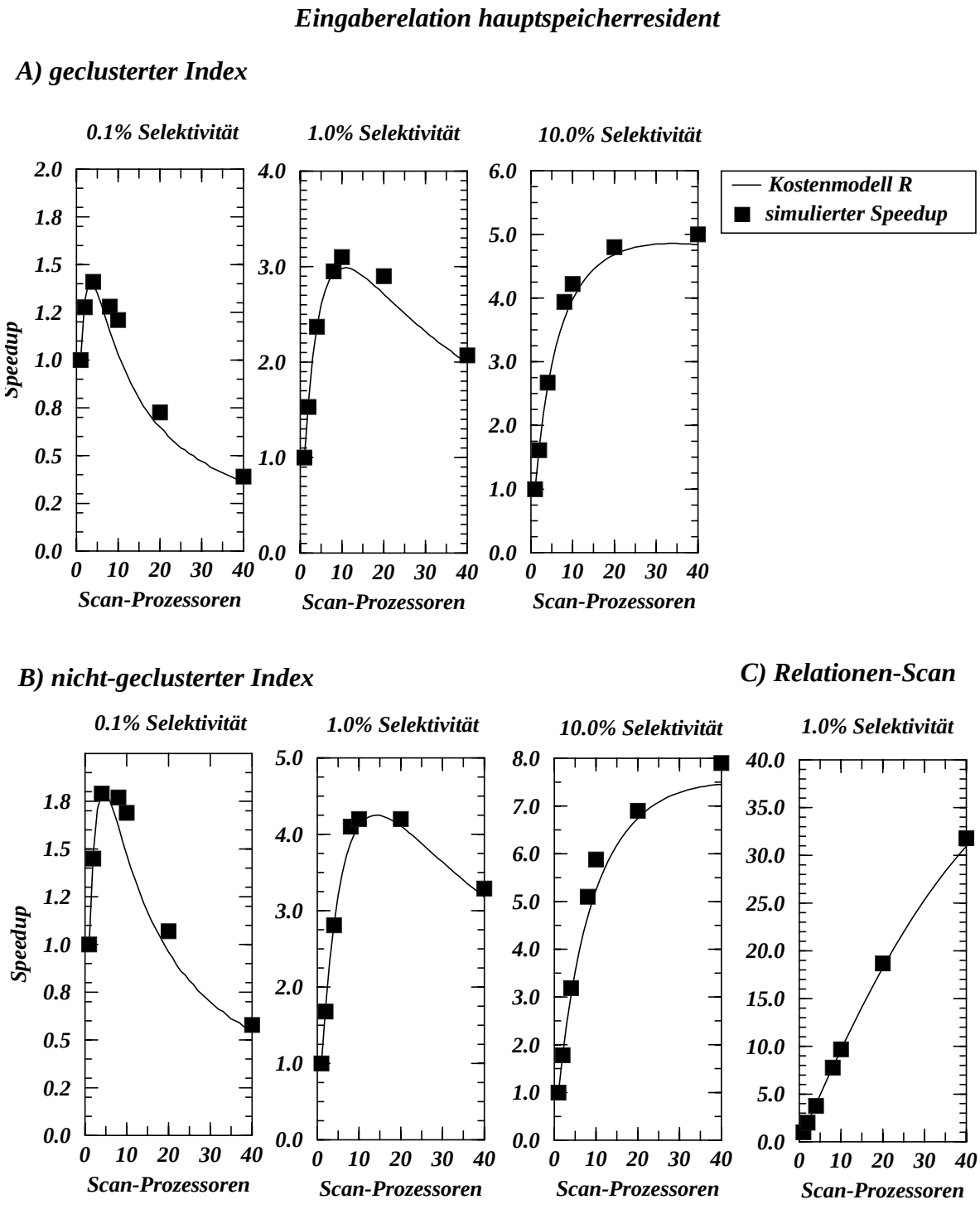


Bild 9: Einfluß von Zugriffsmethode und Anfrageselektivität auf den Antwortzeit-Speedup.

Zugriffsmethode	Selektivität [%]	M	m_{opt}
Index (geclustert)	0.1	250	3
	1.0	2500	11
	10.0	25000	34
Index (nicht geclustert)	0.1	500	5
	1.0	5000	14
	10.0	50000	45
Relationen-Scan	1.0	250000	91

Tabelle 5: Optimale Anzahl der Scan-Ausführungseinheiten in Abhängigkeit von Zugriffsmethode und Anfrageselektivität.

ner gegebenen Relation *C* bekannt, so können wir auf diese Weise eine Abschätzung eines geeigneten Datenverteilungsgrades für diese Relation treffen. Werden beipielsweise 40% der Anfragen auf Relation *C* durch einen nach dem Zugriffsattribut geclusterten Index, weitere 40% durch einen Index ohne Clusterung sowie 20% der Anfragen durch einen Relationen-Scan bearbeitet, und beträgt die Anfrageselektivität jeweils 1%, so bietet ein Datenverteilungsgrad (= Scan-Parallelisierungsgrad) von 28 eine im Mittel optimale Parallelisierung der Scan-Operatoren auf Relation *C* (vgl. Tabelle 6).

Anfragetypen mit Zugriffsmethode	Selektivität [%]	Anteil an der Gesamtlast [%]	m_{opt}	m_{opt} (global)
Index (geclustert)	1.0	40.0	11	0.4 * 11
Index (nicht geclustert)	1.0	40.0	14	0.4 * 14
Relationen-Scan	1.0	20.0	91	0.2 * 91
				28

Tabelle 6: Optimale Anzahl der Scan-Ausführungseinheiten für einen Anfrage-Mix.

Zu beachten ist jedoch, daß lediglich ein im Mittel optimaler Datenverteilungsgrad bestimmt und ausgewählt werden kann. Auf Basis der gewählten Datenverteilung müssen alle Anfragen bearbeitet werden. Anfragen mit einem von diesem Datenverteilungsgrad abweichenden optimalen Parallelisierungsgrad werden unter Umständen stark benachteiligt. Bei großen Änderungen im Lastprofil ist daher im allgemeinen eine Reorganisation der Datenbank (im laufenden Betrieb) notwendig. Gerade angesichts immer größer werdender Datenbanken ([Silberschatz et al. 1991] prognostiziert Datenbanken im Terabyte-Bereich) stellt die zeitraubenden physische Umverteilung ein großes Problem dar, da die Verfügbarkeit der neu zu allokierenden Daten durchgehend gewährleistet werden sollte[16].

16. Neuere Überlegungen zur Architektur paralleler DBS führen daher in die Richtung von *Shared-Disk*-Architekturen ([Rahm 1993a, Valduriez 1993b]), die diese inhärenten Nachteile von Shared-Nothing-Systemen ausschließen.

Plattenallokation der Eingaberelation

Zur Validierung der Kostenformel *R* für Scan-Operatoren bei Plattenallokation der Eingaberelation haben wir exemplarisch Versuche mit und ohne indexunterstütztem Datenzugriff bei einer Anfrageselektivität von 1.0% durchgeführt (Indexobjekte wurden als hauptspeicheresident angenommen).

Bild 10 zeigt die beobachteten Antwortzeiten der parallelen Scan-Verarbeitung im Falle der Plattenallokation in Abhängigkeit von der Zugriffsmethode im Vergleich zu den aus der Kostenformel *R* resultierenden Antwortzeitkurven. Die Kostenformel *R* bietet auch im Falle der Plattenallokation der Basisrelation eine gute Näherung der beobachteten Antwortzeiten.
Grundsätzlich sind gegenüber dem hauptspeicherbasierten Fall - je nach Anzahl der Seitenzugriffe - starke Antwortzeitverschlechterungen zu beobachten. Die Parallelisierung erlaubt jedoch stärkere Antwortzeitverbesserungen als sie im hauptspeicherbasierten Fall gemessen wurden. Anfragen mit vielen Seitenreferenzen (Relationen-Scan und nicht-geclusterter Index) erlauben bis zu einer gewissen Anzahl von Ausführungseinheiten zum Teil sogar superlineare Antwortzeitverbesserungen. Ab 25 PE paßt die gesamte Relation in die verteilten Puffer (Trefferrate 100% ab 25 PE), so daß die gemessenen Antwortzeiten denen des hauptspeicherbasierten Falles entsprechen. Während die Trefferraten bei Indexunterstützung bis 24 PE linear mit der PE-Anzahl wachsen, werden beim Relationen-Scan bis 24 PE keine Treffer im Puffer erzielt. Dies erklärt den Knick der Antwortzeitkurve zwischen

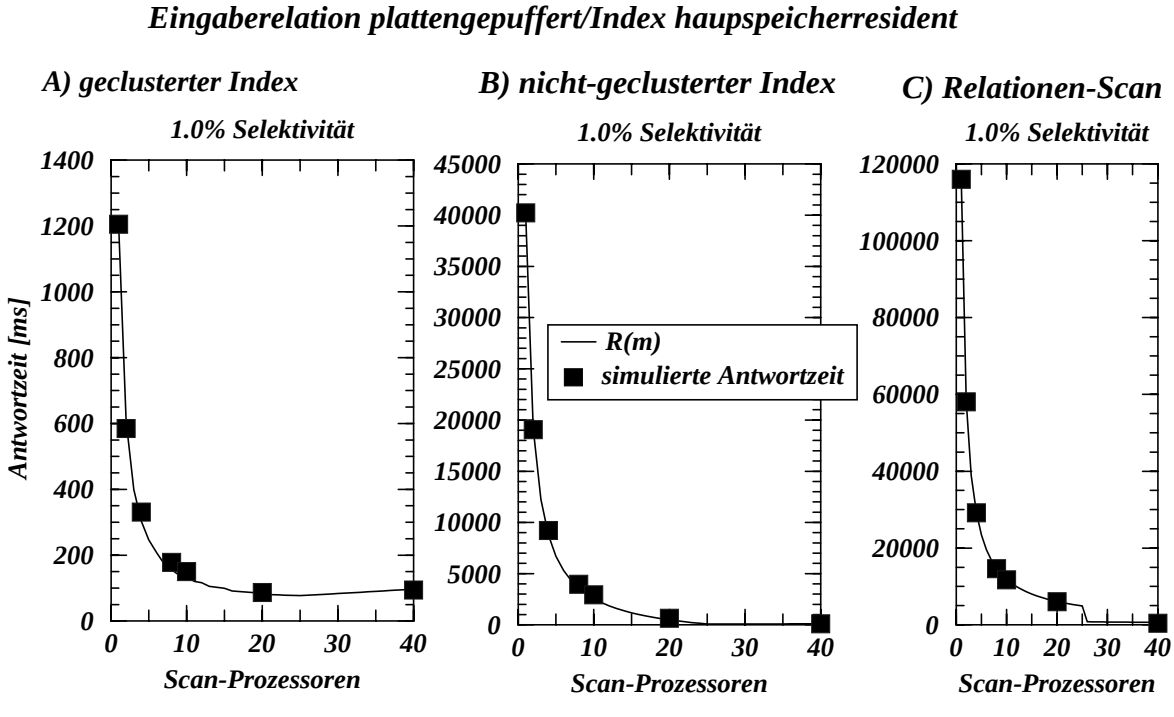


Bild 10: Einfluß von Zugriffsmethode und Anfrageselektivität auf die Effektivität der Parallelisierung von Scan-Operatoren bei Plattenpufferung der Eingaberelation.

24 und 25 PE im Falle des Relationen-Scans.
Auch hier können wir nun durch Ableitung der Kostenformel diejenige Anzahl der Ausführungseinheiten bestimmen, die die niedrigste Antwortzeit ermöglicht. In Abhängigkeit von der globalen Puffergröße und der Anzahl der Datenseiten von Relation C beträgt *R*:

$$R(m) = \begin{cases} a + b \times m + \frac{c \times M}{m} + e \times \frac{P}{m} \times \left(1 - \frac{m \times B}{D}\right) & \text{falls } m < 25 \quad \text{Index-Scan} \quad \text{(i)} \\ a + b \times m + \frac{c \times M}{m} + e \times \frac{P}{m} & \text{Relationen-Scan} \quad \text{(ii)} \\ a + b \times m + \frac{c \times M}{m} & \text{falls } m >= 25 \quad \text{(iii)} \end{cases}$$

Mit Hilfe der Steigungen der Teilfunktionen (i) bis (iii) können wir auch hier das Antwortzeitminimum der Kostenfunktion *R* errechnen. Tabelle 7 zeigt die errechnete optimale Anzahl von Ausführungseinheiten für die durchgeführten Versuche. Grundsätzlich verschiebt sich das Antwortzeitoptimum gegenüber dem hauptspeicherbasierten Fall zu einer größeren Anzahl von Ausführungseinheiten. Ursache sind die E/A-Kosten für das Bereitstellen der Datenbankseiten, die die Zugriffskosten pro Tupel vergrößern. Mit zunehmender Anzahl der Ausführungseinheiten wächst jedoch der verfügbare Puffer. Ab 25 PE paßt die gesamte Relation in die verteilten Hauptspeicher, so daß keine E/A mehr notwendig ist und die Zugriffskosten pro Tupel denen des hauptspeicherbasierten Falles entsprechen.

Zugriffsmethode	Selektivität [%]	M	P	m_{opt}
Index (geclustert)	1.0	2500	63	25
Index (nicht geclustert)	1.0	5000	2500	25
Relationen-Scan	1.0	250000	6250	91

Tabelle 7: Optimale Anzahl der Scan-Ausführungseinheiten in Abhängigkeit von Zugriffsmethode und Anfrageselektivität bei Plattenpufferung der Eingaberelation.

5.3 Versuchsreihen zur parallelen Join-Verarbeitung

Um den Einfluß von Intra-Transaktionsparallelität und der Größe von Zwischenresultaten auf das Antwortzeitverhalten von Join-Anfragen zu untersuchen, haben wir in diesen Versuchsreihen die Anzahl der Join-Ausführungseinheiten sowie die Selektivität der Selektionsprädikate auf den Eingaberelationen variiert. Die Systemgröße (40 PE) wurde, ebenso wie die Datenverteilung der Basisrelationen (Relation A ist auf PE 1 bis 32, Relation B auf PE 33 bis 40 allokiert), konstant gehalten. Der Scan-Aufwand bleibt somit für eine gegebene Selektivität konstant, so daß die Antwortzeitentwicklung ei-

ner jeden Versuchsreihe ausschließlich auf die Variation des Join-Parallelisierungsgrades zurückzuführen ist. Auch hier führen wir Versuche mit hauptspeicherresidenter Speicherung der Eingaberelationen und deren Plattenallokation durch.
Bei der Parallelisierung des Join-Operators kommt folgende Kostenformel *R* zum Tragen (vgl. Kapitel 2):

$$R(1) = a_1 + b + c_1 \times M_1 + c_2 \times M_2 + d_1 \times M_1 \times \log(M_1) + d_2 \times M_2 \times \log(M_2) \qquad \text{für } m = 1, \text{ und}$$

$$R(m) = a + b \times m + c_1 \times \frac{M_1}{m} + c_2 \times \frac{M_2}{m} + d_1 \times \frac{M_1}{m} \times \log\left(\frac{M_1}{m}\right) + d_2 \times \frac{M_2}{m} \times \log\left(\frac{M_2}{m}\right) \qquad \text{für } m > 2$$

Durch Einsetzen der Simulationsparameter in die zugehörigen Gleichungen aus Kapitel 3 erhalten wir die Werte der Koeffizienten dieser Kostenformel. Tabelle 8 zeigt die Koeffizienten in Abhängigkeit

alle Dateien hauptspeicherresident:

Selektivität [%]	M1	M2	a1 [ms]	a [ms]	b [ms]	c1 [ms]	c2 [ms]	d1 [ms]	d2 [ms]
0.1	1000	250	70.98	72.23	1.0	0.03164	0.06875	0.0005	0.0005
1.0	10000	2500	114.9	127.4					
10.0	100000	25000	554	679					

Tabelle 8: Koeffizientenwerte der Kostenformel *R*.

von der Anfrageselektivität bei haupspeicherresidenter Allokation aller Dateien.
Zu beachten ist, daß der Aufwand für die parallele Bearbeitung der verteilten Scan-Operatoren vom Parallelitätsgrad des Join-Operators unabhängig ist, so daß der Scan-Aufwand in dem konstanten Kostenanteil *a* enthalten ist.

Bild 11 zeigt den Vergleich der simulierten Antwortzeiten mit den Antwortzeitentwicklungen, wie sie durch die Kostenformel *R* ermittelt wurden. In dieser Versuchsreihe wurden alle Dateien (Basisrelationen und Zwischenergebnisse) hauptspeicherresident allokiert. Wir erkennen, daß der Verlauf der Antwortzeitkurven des Kostenmodells jeweils für eine gegebenen Selektivität mit den simulierten Antwortzeiten übereinstimmt. Geringfügige Abweichungen zwischen dem Join-Kostenmodell und den Simulationsergebnissen sind im wesentlichen auf im Kostenmodell enthaltene Vereinfachungen zurückzuführen. Das Kostenmodell berücksichtigt - im Gegensatz zum Simulationsmodell - beispielsweise keine Streuung der Verarbeitungszeiten verteilter Ausführungseinheiten, wie sie vor al-

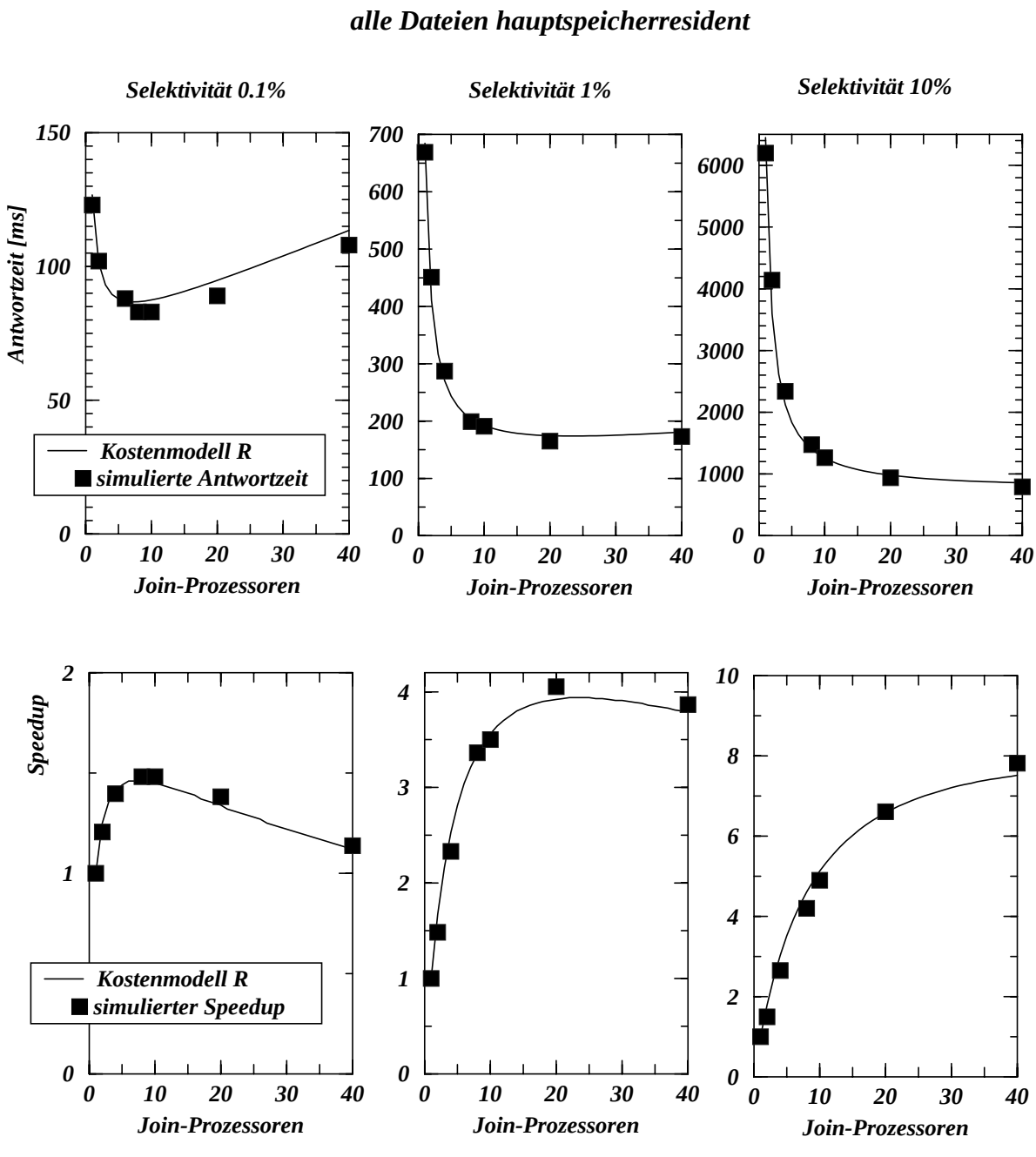


Bild 11: Antwortzeit- und Speedup-Entwicklung der parallelen Join-Verarbeitung - Vergleich zwischen Simulationsergebnissen und Kostenmodell R.

lem bei einer hohen Anzahl von Ausführungseinheiten auftreten. Derartige Streuungen sind v.a. durch Wartezeiten auf DB-Ressourcen bedingt. Im Simulationsmodell geben außerdem Parameterwerte wie beispielsweise CPU-Instruktionen und Plattenzugriffszeiten im Hinblick auf eine realitätsnahe Modellierung keine konstanten Werte an, sondern nur Mittelwerte. Die tatsächlichen Werte sind um diese Mittelwerte exponentialverteilt.

Auch hier können wir wiederum auf Basis des Kostenmodells die Anzahl m_{opt} der Join-Ausführungseinheiten bestimmen, die für eine gegebene Join-Anfrage die Antwortzeit minimiert. Tabelle 9 faßt die berechneten Parallelisierungsgrade in Abhängigkeit von der Anfrageselektivität zusammen.

Scan-Selektivität [%]	M1	M2	m_{opt}
0.1	1000	250	7
1.0	10000	2500	23
10.0	100000	25000	75

Tabelle 9: Optimale Anzahl der Join-Ausführungseinheiten in Abhängigkeit von der Anfrageselektivität.

Betrachtet man nun den Antwortzeit-Speedup, so ergibt sich auch hier ein weitgehend identischer Verlauf der Speedup-Kurve von Simulationsversuchen und Kostenmodell *R*. Insbesondere liefert *R* für alle betrachteten Anfrageselektivitäten den gleichen optimalen Parallelisierungsgrad wie die Simulationsversuche. Wir dürfen daher annehmen, daß das Kostenmodell *R* auch für Join-Anfragen eine realistische Annäherung der Antwortzeitentwickung in Abhängigkeit vom Parallelisierungsgrad liefert.
Zu beachten ist der relativ schlechte Antwortzeit-Speedup. Obwohl für den Join-Operator mit logarithmischem Kostenverlauf ein superlinearer Speedup zu erwarten ist, ist der Antwortzeit-Speedup für alle Selektivitäten deutlich sublinear. Ursache ist der hohe Anteil an konstanten Kosten (v.a. der Scan-Aufwand), der nicht durch die Parallelisierung verbessert werden kann. Beispielsweise im Falle einer Selektivität von 0.1% entfallen auf den konstanten Kostenanteil bereits ca. 71 ms. Dieser Anteil umfaßt somit bereits 58% der Antwortzeit ohne Parallelisierung, so daß die Operatorparallelisierung lediglich dazu dient, 42% der Anfrageantwortzeit zu verbessern. Wir sehen also, daß die Effektivität der Parallelisierung in hohem Maße davon abhängt, wie hoch der durch Parallelisierung verbesserbare Kostenanteil im Verhältnis zu nicht reduzierbaren Anteilen ist[17].

In weiteren Versuchen haben wir untersucht, wie sich die Allokation von Datenbankdateien auf Platten auf die Effektivität der Anfrageparallelisierung respektive die Koeffizienten der Kostenformel auswirkt. Zu diesem Zweck wurden die Basisrelationen auf Platten allokiert und alle temporären Dateien resident im Hauptspeicher gehalten. Bild 12 zeigt die resultierenden Antwortzeitkurven.
Erwartungsgemäß führt die Plattenallokation zu einer Antwortzeiterhöhung, die für speicherintensive

17. Bezüglich weiterer Leistungsaussagen zur parallelen Join-Verarbeitung sei auf [Marek und Rahm 1993] und [Rahm und Marek 1993] verwiesen

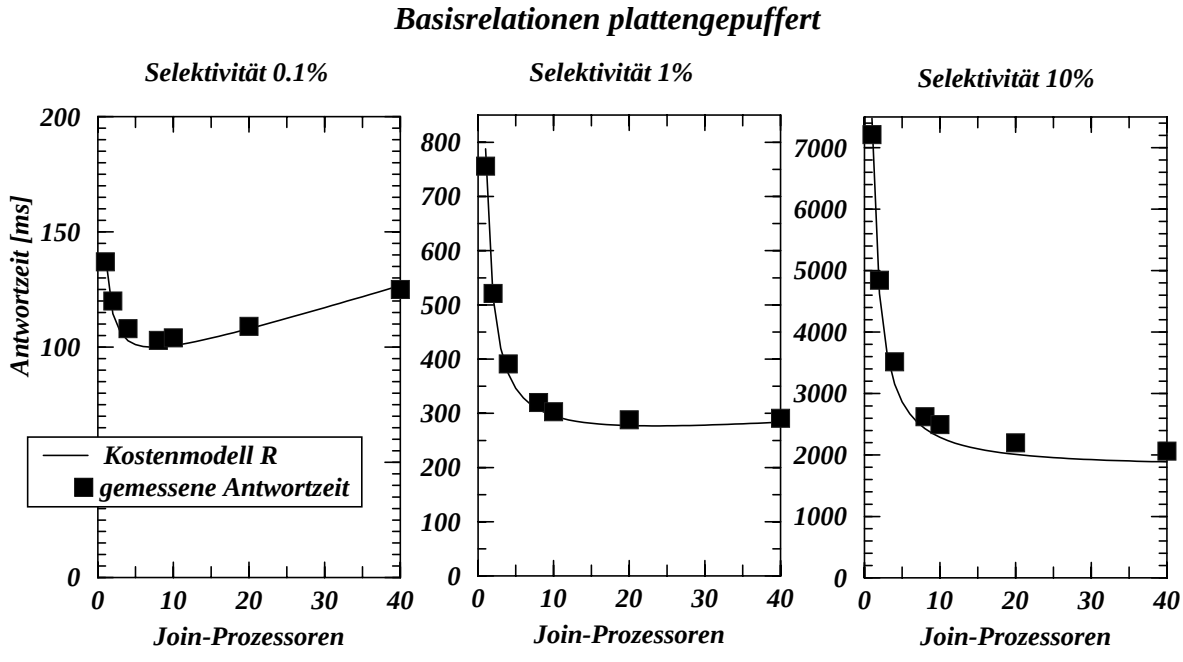


Bild 12: Einfluß der Pattenallokation von Datenbankdateien auf die Antwortzeitentwicklung.

Anfragen (hohe Selektivität bzw. große Zwischenergebnisse) besonders hoch ausfällt.
Da lediglich die Basisrelationen, nicht aber die temporären Relationen, auf Platten allokiert sind, ist die Antwortzeiterhöhung ausschließlich auf die Verteuerung der Scan-Kosten zurückzuführen. Die übrigen Kosten, insbesondere die vom Join-Parallelisierungsgrad abhängigen Kosten, bleiben unberührt. Gegenüber den Versuchsreihen mit hauptspeicherresidenter Allokation aller Dateien vergrößern sich daher lediglich die konstanten Kostenanteile *a* (vgl. Tabelle 10) der Kostenformel *R*; die

Plattenallokation der Basisrelationen:

Selektivität [%]	a1 [ms]	a [ms]
0.1	82.2	8345
1.0	204.66	217.16
10.0	1440	1565

Tabelle 10: Koeffizientenwerte der Kostenformel *R*.

Koeffizienten *b*, *c1* und *c2* sowie *d1* und *d2* bleiben gleich. Infolgedessen führt die Plattenallokation der Basisrelationen gegenüber dem hauptspeicherbasierten Fall zu keiner Änderung in der optimalen Anzahl von Join-Ausführungseinheiten.

6 Zusammenfassung und Ausblick

Für die Anfrageparallelisierung im Einbenutzerbetrieb haben wir ein Kostenmodell vorgestellt, das die Antwortzeitentwicklung von Datenbankanfragen in Abhängigkeit vom Parallelisierungsgrad beschreibt. Mit Hilfe des Kostenmodelles kann für Intra-Operator-Parallelität - die in praxi wichtigste Form von Intra-Query-Parallelität - derjenige Parallelitätsgrad bestimmt werden, der die Antwortzeit von Anfragen minimiert. Das Modell kann somit zur Unterstützung des Optimierers bei der Anfrageparallelisierung eingesetzt werden. Es kann sowohl als Entscheidungshilfe für die statische Bestimmung eines geeigneten Datenverteilungsgrades dienen, als auch für die dynamische Ermittlung des optimalen Parallelitätsgrades von Join-Anfragen genutzt werden. Wir haben sowohl Operatoren mit linearem als auch logarithmischem Kostenverlauf betrachtet. Des weiteren wurde der Einfluß des Speichermediums (Hauptspeicher versus Platte) der Operanden berücksichtigt.
Wir haben drei unterschiedliche Antwortzeitanteile identifiziert, die sich bezüglich ihrer Abhängigkeit vom Parallelisierungsgrad voneinander unterscheiden. In der Regel enthält die Antwortzeit einer Anfrage einen Kostenanteil, der von der Operatorparallelisierung nicht betroffen ist, d.h. unabhängig vom Parallelisierungsgrad konstant ist. Diesem Anteil sind u.a. nicht-parallelisierte Operatoren und Query-Verwaltungsaktivitäten zuzurechnen. Der erwünschte Effekt der Anfrageparallelisierung (Nutzen), nämlich die Verkürzung der Query (bzw. Operator)-Bearbeitungszeit läßt sich in einem mit zunehmendem Parallelisierungsgrad abnehmenden Kostenanteil beschreiben. Die Reduktion der Operatorbearbeitungszeit ist dabei auf eine mit zunehmender Anzahl von Ausführungseinheiten abnehmenden Größe der Operandenmenge (Tupeln bzw. Datenbankseiten) pro Ausführungseinheit zurückzuführen. Die Beziehung zwischen Parallelisierungsgrad und Operatorbearbeitungszeit ist abhängig vom Operatortyp. Im Falle von Operatoren mit linearem Kostenverlauf erlaubt eine Vergrößerung des Intra-Operator-Prallelitätsgrads eine lineare Verbesserung dieses Kostenterms; im Falle logarithmischer Operatoren ist eine superlineare Beziehung gegeben. Dem letzten dieser drei Antwortzeitkostenanteile sind bei der Anfrageparallelisierung entstehenden Kommunikations- und Verwaltungskosten zuzurechnen. Diese Kosten wachsen linear mit zunehmendem Parallelisierungsgrad. Zur Bestimmung dieser Kostenanteile haben wir eine Abschätzung präsentiert, die auf einem detaillierten Modell der parallelen Anfragebearbeitung basiert.
Die Effektivität der Operatorparallelisierung (Speedup) ist von allen drei Kostentermen abhängig. Der optimale, die Antwortzeit minimierende Parallelitätsgrad ist hingegen unabhängig von dem konstanten Kostenanteil. Lediglich das Kosten-/Nutzenverhältnis der von der Operatorparallelisierung abhängigen Antwortzeitanteile in Verbindung mit der Größe der Operandenmenge bestimmt den optimalen Parallelitätsgrad im Einbenutzerbetrieb.

Zur Beurteilung der Güte des Kostenmodells wurde ein simulativer Ansatz gewählt. Für Leistungsanalysen der parallelen Anfragebearbeitung haben wir in einem umfassenden Simulationsmodell ein generisches Shared-Nothing-System hinsichtlich seiner Architektur und seiner Verarbeitungskonzepte modelliert. Das Simulationsmodell wurde anhand von Vergleichssimulationen zu

realen Shared-Nothing-Systemen validiert. Auf Basis des Simulationsmodelles durchgeführte Antwortzeitversuche zur Parallelisierung von Scan- und Join-Operatoren haben die Übereinstimmung von Kosten- und Simulationsmodell gezeigt. Wir haben gesehen, daß sich die beobachteten Antwortzeiten bei der parallelen Scan-und Join-Bearbeitung geeignet durch das vorgestellte Kostenmodell beschreiben lassen. Insbesondere liefern das Kostenmodell einerseits und die beobachteten Antwortzeitkurven andererseits den gleichen optimalen Parallelitätsgrad der Operatorparallelisierung. Ausgehend von der Validierung des Simulationsmodelles gegen reale Systeme dürfen wir annehmen, daß das Kostenmodell eine gute Annäherung an das Verhalten realer Systeme bietet. Leistungsuntersuchungen von Datenbankanfragen im Einbenutzerbetrieb können daher mit Hilfe des analytischen Modelles durchgeführt werden. Verglichen mit simulationsbasierten Untersuchungen ist ein analytischer Ansatz weitaus weniger aufwendig.

In zukünftigen Arbeiten wollen wir uns der Frage widmen, wie der Anfrageoptimierer bei der Operatorparallelisierung im Mehrbenutzerbetrieb geeignet unterstützt werden kann. Im Mehrbenutzerbetrieb gilt es das Verhältnis zwischen Nutzen (Antwortzeitverkürzung) und Kosten (Kooperations- und Kommunikations-Overhead) der Parallelisierung in Abhängigkeit von der Systemauslastung zu optimieren, mit dem Ziel globale Antwortzeit- und Durchsatzvorgaben bestmöglich zu erfüllen. Teilaspekte der zitierten Fragestellung sind eine von der aktuellen Lastsituation abhängige, dynamische Bestimmung des Operatorparallelitätsgrades sowie die Allokation von Ausführungseinheiten zu Prozessorelementen mit dem Ziel der Lastbalancierung. Erste Untersuchungen hierzu wurden bereits in [Rahm und Marek 1993] durchgeführt. Schließlich sollten in weiterführenden Arbeiten einige der bisher getroffenen Vereinfachungen aufgegeben werden. Im Hinblick auf realitätsnahe Anwendungslasten ist vor allem der Einfluß von Skew auf das Leistungsverhalten zu berücksichtigen.

7 Literatur

Apers, P.; van den Berg, C.; Flokstra, J.; Grefen, P.; Kersten, M.; Wilschut, A. 1992:
PRISMA/DB: A Parallel, Main-Memory Relational DBMS. Memoranda Informatica 92-12, University of Twente, Enschede, The Netherlands.

Boral, H.; Alexander, W.; Clay, L.; Copeland, G.; Danforth, S.; Franklin, M.; Hart, B.; Smith, M.; Valduriez, P. 1990:
Prototyping Bubba: A Highly Parallel Database System. *IEEE Trans. on Knowledge and Data Engineering 2*(1), 4-24.

DeWitt, D.J.; Ghandeharizadeh, S.; Schneider, D.A.; Bricker, A.; Hsiao, H.; Rasmussen, R. 1990:
The Gamma Database Machine Project. *IEEE Trans. on Knowledge and Data Engineering 2*(1), 4-62.

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

Englert, S., Gray, J., Kocher, T., Shath, P. 1990:
A Benchmark of NonStop SQL Release 2 Demonstrating Near-Linear Speedup and Scale-Up on Large Databases. *Proc. ACM SIGMETRICS Conf.*, 245-246.

Gray, J. (Hrsg.) 1991:
The Benchmark Handbook. Morgan Kaufmann Publishers Inc.

Härder, T. 1987:
Realisierung von operationalen Schnittstellen. In: Datenbank-Handbuch, Hrsg. P.C. Lockemann und J.W. Schmidt, Springer-Verlag.

Livny, M. 1989:
DeNet Users's Guide, Version 1.5. Computer Science Department, University of Wisconsin, Madison.

Marek, R.; Rahm, E. 1992:
Performance Evaluation of Parallel Transaction Processing in Shared Nothing Database Systems. *Proc. 4th Int. PARLE Conference 1992*, Lecture Notes in Computer Science 605, Springer Verlag, 295-310.

Marek, R.; Rahm, E. 1993:
On the Performance of Parallel Join Processing in Shared Nothing Database Systems. *Proc. 5th Int. PARLE Conference* 1993, Lecture Notes in Computer Science, Springer Verlag.

Mohan, C., Lindsay, B., Obermarck, R. 1986:
Transaction Management in the R* Distributed Database Management System. *ACM Trans. on Database System* 11 (4), 378-396.

Neches, P.M.1986:
The Anatomy of a Database Computer - Revisited. *Proc. IEEE CompCon Spring Conf.*, 374-377.

Özsu, M.T., Valduriez, P. 1991:
Principles of Distributed Database Systems. Prentice Hall.

Pirahesh, H.; Mohan, C.; Cheng, J.; Liu, T.S.; Selinger, P. 1990: Parallelism in Relational Data Base Systems: Architectural Issues and Design Approaches. In *Proc. 2nd Int.Symposium on Databases in Parallel and Distributed Systems*, IEEE Computer Society Press.

Rahm, E.; Marek, R. 1993: Analysis of Dynamic Load Balancing Strategies for Parallel Shared Nothing Database Systems. Erscheint in: *Proc. 19th Int. Conf. on Very Large Data Bases*, August 1993, Dublin, Ireland.

Rahm, E. 1993a: Parallel Query-Processing in Shared Disk Database Systems. Technischer Bericht 1/93, Universität Kaiserslautern, Fachbereich Informatik, März 1993.

Rahm, E. 1993b: DBMS-Leistungsvergleich mit TPC-Benchmarks. IX-Magazin, Mai 1993.

Silberschatz, A.; Stonebraker, M.; Ullman, J. 1991:
Database Systems: Achievements and Opportunities. *Communications of the ACM* 34(10), 110-120.

Stonebraker, M. 1986:
The Case for Shared Nothing. *IEEE Database Engineering* 9(1), 4-9.

The Tandem Database Group 1988:
A Benchmark of NonStop SQL on the Debit Credit Transaction. *Proc. ACM SIGMOD Conf.* 337-341.

The Tandem Database Group 1989:
NonStop SQL, A Distributed, High-Performance, High-Availability Implementation of SQL. Lecture Notes in Computer Science 359, Springer-Verlag, 60-104.

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

Valduriez, P. 1993b: Parallel Database Systems: the case for shared-something. *Proc. 9th IEEE Int. Conf. on Data Engineering*, 460-465.

Walton, C.B; Dale A.G.; Jenevein, R.M. 1991:
A Taxanomy and Performance Model of Data Skew Effects in Parallel Joins. *Proc. 17th Int. Conf. on Very Large Data Bases*, 537-548.

Watson, P., Townsend, P. 1991:
The EDS Parallel Relational Database System. In: Parallel Database Systems (*Proc. PRIMSA Workshop*), Lecture Notes in Computer Science 503, Springer-Verlag, 149-168.

Wilschut, A.; Flokstra, J.; Apers, P. 1992:
Parallelism in a Main-Memory DBMS: The performance of PRISMA/DB. *Proc. 18th Int. Conf. on Very Large Data Bases*, 521-532.