in: Tagungsband der GI-Fachtagung 'Datenbanken in Büro, Technik und Wissenschaft' (BTW'95), Informatik aktuell, Dresden, März 1995, S.232-251.

Ein Kostenmodell der parallelen Anfragebearbeitung in Shared-Nothing-Datenbanksystemen

Robert Marek

Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, 67618 Kaiserslautern
e-mail: marek@informatik.uni-kl.de

Kurzfassung:

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 weit verbreiteten mengenorientierten, relationalen Anfragesprachen erlaubt vor allem Intra-Operator-Parallelität eine effektive Antwortzeitverkürzung. Die Antwortzeit kann jedoch durch zunehmende Parallelisierung nicht beliebig verkürzt werden. Wird ein gewisser Parallelisierungsgrad überschritten, tritt 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.

1 Einführung

In der Datenbankverarbeitung gewinnen komplexe Ad-hoc-Anfragen zunehmend an Bedeutung, 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].

Parallele Datenbanksysteme sind daher 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. 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 von Transaktionen bzw. Anfragen (Queries) wird Intra-Transaktionsparallelität benötigt - in Form von *Inter-* oder *Intra-DML-Parallelität*. Inter-DML-Parallelität bezeichnet die konkurrente Ausführung verschiedener DML-Befehle (DB-Operationen, Anweisungen der DBS-Anfrage- und Manipulationssprache) 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

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

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. Infolgedessen 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 dem Ziel, globale Antwortzeit- und Durchsatzvorgaben möglichst gut zu erfüllen. Der Parallelisierungsgrad im Mehrbenutzerbetrieb kann - je nach Systemauslastung - zum Teil deutlich von dem des Einbenutzerbetriebs abweichen [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 unirelationalen Anfragen (*Scan-Anfragen*) schätzen wir in Kap. 3 die Koeffizienten des analytischen Kostenmodells ab. Daran anschließend wollen wir das Kostenmodell einschließlich der errechneten Koeffizienten anhand der parallelen Bearbeitung von Scan-Anfragen in Shared-Nothing-Systemen validieren. Dazu haben wir die parallele Bearbeitung von Anfragen mit Hilfe eines Simulationsmodells untersucht. Wir präsentieren in Kap. 4 die durchgeführten Versuche und bewerten das analytische Kostenmodell. In Kap. 5 werden wir die Übertragung des Kostenmodells auf Anfragen skizzieren, die zwei Relationen über einen Join-Operator miteinander verknüpfen.

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).

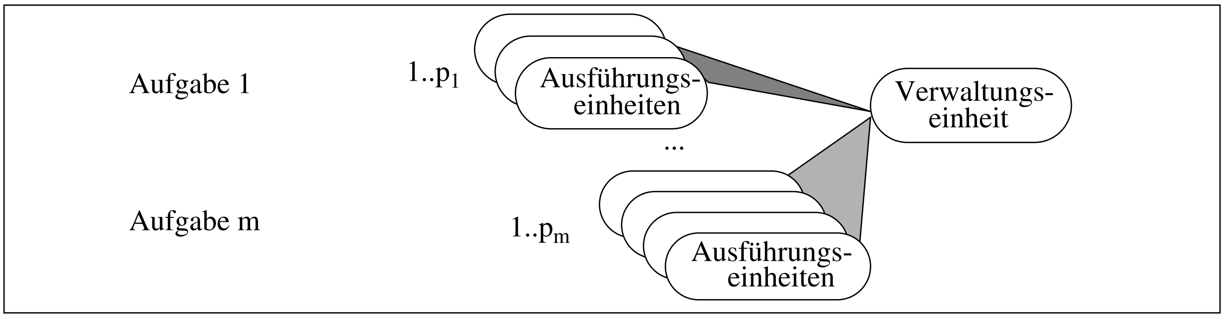


Bild 1: Ein abstraktes Modell der parallelen Anfragebearbeitung.

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 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 Speichermedium (Hauptspeicher bzw. Platte) der Operanden. Die Koeffizienten der Kostenformel werden wir auf analytischem Wege in Kap. 3 herleiten.

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 wird in [Wilschut et al. 1992] der Einfluß des Speichermediums nicht explizit modelliert, und Operatoren mit superlinearen Verarbeitungskosten werden nicht berücksichtigt. 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. Kap. 3).

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 von Selektionsbefehlen verteilter Ausführungseinheiten. Sicherlich sollten weiterführende Arbeiten den Einfluß von Skew auf die Kostenentwicklung der Operatorparallelisierung berücksichtigen.

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 Treffertupel durch eine zentrale Instanz. Der Aufwand für die Merge-Operation hängt von der Anzahl der Treffertupel, 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 Tupel insgesamt

$$c \times M \qquad \text{(ii)}$$

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

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

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 von Operatoren mit linearen Verarbeitungskosten auf die Anfrageantwortzeit auswirkt. Für ein unirelationales SELECT beträgt die Anfrageantwortzeit R in Abhängigkeit vom Parallelisierungsgrad m des Scan-Operators

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

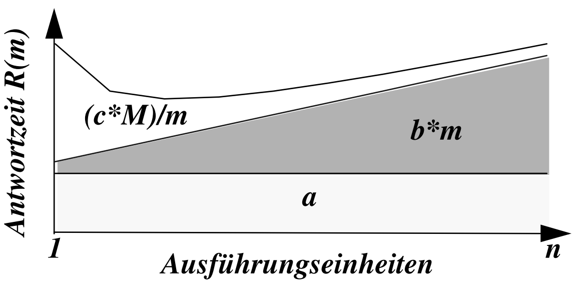
Zeiteinheiten, wobei eine hauptspeicherresidente Speicherung der Eingaberelation angenommen ist. Wir können bei der parallelen Anfragebearbeitung eine Antwortzeitentwicklung beobachten, wie sie in Bild 2 dargestellt ist. Denjenigen Parallelisierungsgrad m_{opt}, der die geringste Antwortzeit erzielt, 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 Tupel 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. In unserem Beispiel beträgt der Antwortzeit-Speedup:

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



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

Bild 2: Einfluß der parallelen Anfragebearbeitung auf die Antwortzeit.

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 (z.B. Sortieralgorithmen, wie sie u.a. für die Implementierung von Sort-Merge-Joins benötigt werden). Der zugehörige Kostenterm läßt sich in analoger Weise zu linearen Kostenfunktionen beschreiben (vgl. Kap. 5).

Berücksichtigung von Platten-E/A:
Die bisher beschriebenen Kostenterme beinhalten diejenigen Operatorkosten, die beim Hauptspeicherzugriff der Operatoren auf deren Eingabetupel 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 ($0 \leq f \leq 1$), so sind pro Ausführungseinheit zusätzlich zu den Hauptspeicherzugriffen

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

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[2]:

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

Unter Berücksichtigung der Terme (iv) und (v) erkennen wir, daß mit zunehmender PE-Anzahl einerseits 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 Kap. 4 werden wir die Koeffizienten der Kostenformel auf analytischem Wege ermitteln. Zu diesem Zweck wollen wir im folgenden Kapitel unsere Sicht der parallelen Anfrageverarbeitung präzisieren.

---

2. Dabei unterstellen wir Lokalität im Referenzierungsverhalten derart, daß Datenbankseiten einer Relation durch mehrere Benutzer bzw. Anfragen gemeinsam benutzt werden (*inter transaction locality* [Härder 1987]). Werden mehrere Relationen im gemeinsamen Puffer abgelegt, so vergrößert sich die Fehlseitenwahrscheinlichkeit entsprechend dem Umfang dieser Relationen. Liegt keinerlei Inter-Transaktions-Lokalität vor, so muß jede Anfrage im allgemeinen alle von ihr benötigten Datenbankseiten einlesen. Dies entspricht einer Fehlseitenwahrscheinlichkeit von 1.

3 Die Verfeinerung des Kostenmodells für Scan-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-Anfragen detailliert nachvollziehen und daraus eine Abschätzung der gesuchten Koeffizienten für Scan-Operatoren treffen.

3.1 Lastprofil

Die Anfragen, die wir hier zugrundelegen, verwenden den *Scan* als einzigen relationalen Basisoperator. 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. Für die verteilte Anfragebearbeitung benötigen wir einen weiteren (Meta-) Operator: der sogenannte *Merge-Operator* “mischt” mehrere (sortierte) parallele Datenströme in einen sequentiellen Strom.

Wir wollen im folgenden untersuchen, wie sich die Anzahl der Ausführungseinheiten eines Scan-Operators 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 Relation ist. 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-Protokolls 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 oder Mischen von Eingabeströmen),
- CPU-Kosten für das Senden/Empfangen von Nachrichten 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 hier verwendeten Abkürzungen sind in Tabelle 2 aufgelistet.

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 Über-

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

Tabelle 2: Notation der Kostenabschätzung.

einstimmung 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)

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

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 $kard_i/(nrPE_i*block_i)$ Datenbankseiten referenziert. Die hierfür notwendigen E/A-Kosten schätzen wir mit *io* Zeiteinheiten pro Seite ab[3]. 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 Indexstruktur - 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 Indexstruktur, die die Tupel nach dem Zugriffsattribut clustert (d.h. Tupel mit dem gleichen Attributwert in der selben physischen Seite ablegt), genügt ein einmaliger Durchlauf der Indexstruktur, 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[4]. Bei einer Indexstruktur ohne Cluster-Bildung muß in der Regel die unterste Ebene der Indexstruktur vor jeder Tupelreferenz referenziert werden, da hier keine Verkettung der Datenseiten untereinander genutzt werden kann. Insgesamt werden somit $(kard_i*sel_i)/(nrPE_i)$ Einträge der Indexseiten und ebenso viele Tupel gelesen. 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:

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

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

Aufgrund des sequentiellen Referenzierungsmusters des Relationen-Scans sind keine Treffer zu er-

---

3. Hier wird vereinfachend angenommen, daß Seitenzugriffe in wahlfreier und sequentieller Reihenfolge gleich teuer sind. Bietet das E/A-System kürzere Zugriffszeiten bei sequentiellem Zugriffsmuster (z.B. infolge kürzerer Plattenzugriffszeiten oder bei Unterstützung von Prefetching), so ist dies durch Definition zugriffsmusterspezifischer E/A-Kosten zu berücksichtigen.

4. Wir berücksichtigen an dieser Stelle keine E/A von Indexseiten. Indexseiten höherer Baumebenen können aufgrund ihrer hohen Referenzierungshäufigkeit zumeist von der Pufferverwaltung im Hauptspeicher gehalten werden. Für Blatt-Indexseiten kann der E/A-Aufwand analog zu den Datenobjekten abgeschätzt werden.

warten (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 Kap. 2 vorgestellte Formel für wahlfreie gleichverteilte Zugriffe zur Anwendung kommt. Die Gesamtanzahl der Datenbankseiten berechnet sich als Quotient aus Kardinalität und Blockungsfaktor der Relation[5].

Mischen der lokalen Scan-Ergebnisströme

Nach erfolgter Bearbeitung der lokalen Scan-Ausführungseinheiten müssen die verteilten Ergebnismengen gemischt werden. Dazu sendet jede Scan-Ausführungseinheit ihre Treffertupel in Verbindung mit einer Quittungsmeldung an die Verwaltungseinheit. Bei einer Gesamtanzahl von $kard_i*sel_i$ Ergebnistupeln nehmen wir $(kard_i*sel_i)/nrPE_i$ Ergebnistupel pro Scan-Ausführungseinheit an. Jede Ausführungseinheit sendet somit eine Nachricht und kopiert die $(kard_i*sel_i)/nrPE_i$ lokalen Treffertupel 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 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*[6] Zeiteinheiten, so wird jede der $(kard_i*sel_i*\mathrm{srs})/nrPE_i$ 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 Kap. 2 ermitteln.

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

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

---

5. In dieser Arbeit steht vor allem der Aspekt der Parallelverarbeitung im Vordergrund. Wir begnügen uns daher mit einer vergleichsweise einfachen Abschätzung lokaler Verarbeitungskosten (insb. Anzahl physischer Seitenzugriffe und Puffer-Trefferraten), welche für unsere Zwecke eine hinreichende Güte bietet. Ist eine detailliertere Berücksichtigung der E/A-Kosten erforderlich, so können diesbezüglich aus der Literatur bekannte Kostenabschätzungen integriert werden [Waters 1976, Yao 1977, Mackert und Lohman 1989].

6. 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].

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 und der Anfrageselektivität:

$$M = \begin{cases} kard_i & \text{Relationen-Scan} \\ kard_i \times sel_i & \text{Index mit Cluster-Bildung} \\ kard_i \times sel_i \times 2 & \text{Index ohne Cluster-Bildung} \end{cases}$$

Ohne Indexunterstützung muß die gesamte Relation gelesen werden ($kard_i$ Referenzen). Bei Indexunterstützung mit Cluster-Bildung kann direkt auf die Tupel, die das Selektionskriterium erfüllen ($kard_i$*sel_i), zugegriffen werden. Ohne Cluster-Bildung ist vor jedem Tupelzugriff anhand eines Zugriffs auf einen Eintrag der Indexstruktur festzustellen, in welcher Seite sich das gesuchte Tupel befindet, so daß insgesamt (2*$kard_i$*sel_i) Referenzen zu bearbeiten sind.
Für die Anzahl P der Seitenreferenzen gilt entsprechend folgende Unterscheidung:

$$P = \begin{cases} \dfrac{kard_i}{block_i} & \text{Relationen-Scan} \\ \dfrac{kard_i \times sel_i}{block_i} & \text{Index mit Cluster-Bildung} \\ kard_i \times sel_i & \text{Index ohne Cluster-Bildung} \end{cases}$$

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:

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

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

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

$$c = \frac{scan}{mips} + \begin{cases} \left(\dfrac{copy}{mips} + nettime\right) \times sel_i \times srs & \text{Relationen-Scan} \\ \left(\dfrac{copy}{mips} + nettime\right) \times srs & \text{Index mit Cluster-Bildung} \\ \left(\dfrac{copy}{mips} + nettime\right) \times \dfrac{srs}{2} & \text{Index ohne Cluster-Bildung} \end{cases}$$

Werden die verteilten Ergebnisströme nur bei einer wirklichen Scan-Parallelisierung gemischt, so ergibt sich obige Unterscheidung zwischen $nrPE_i = 1$ (sequentielle Bearbeitung) und $nrPE_i > 1$ (parallele Bearbeitung) Ausführungseinheiten für den Koeffizienten a.
Im Zuge der Simulationsversuche in Kap. 4 werden wir die Werte dieser Koeffizienten entsprechend der Parametrisierung des Simulationssystems errechnen und die Kostenformel validieren.

4 Validierung des Kostenmodells paralleler Anfragebearbeitung

In diesem Kapitel untersuchen wir das Leistungsverhalten der parallelen Verarbeitung von unirelationalen Anfragen. Unser Ziel ist es zu überprüfen, wie gut das entwickelte Kostenmodell das beobachtete Leistungsverhalten annähert.

Für Leistungsanalysen der parallelen Anfragebearbeitung haben wir in einem umfassenden Simulationsmodell ein generisches Shared-Nothing-System hinsichtlich seiner Architektur und seiner Verarbeitungskonzepte modelliert. Dieses Modell wollen wir hier zur Validierung des Kostenmodells heranziehen.
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. In [Marek und Rahm 1992; Marek 1993] wurden Versuche zur Parallelisierung von Scan-Anfragen durchgeführt. Insbesondere wurde die Effektivität der Parallelisierung in Abhängigkeit von den Einflußgrößen Zugriffsmethode und Anfrageselektivität 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 relative 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.

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 Kostenmodells auf unser Simulationssystem - stellvertretend für ein reales System - beziehen dürfen.

Wir geben in Abschnitt 4.1 einen Überblick über die Parametrisierung des Shared-Nothing-Modells und der Datenbankanfragen. Aus den Simulationsparametern berechnen wir die Koeffizienten der Kostenformeln für Scan- und Join-Anfragen und präsentieren schließlich in Abschnitt 4.2 die Ergebnisse der durchgeführten Simulationsversuche.

4.1 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 A (250.000 Tupel) 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 A 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 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.

4.2 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 hauptspeicherresidenten 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 (vgl. Kap. 3):

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

Durch Einsetzen der Simulationsparameter in die zugehörigen Gleichungen aus Kap. 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

Systemkonfiguration	**Parameterwerte**	**Systemkonfiguration**	**Parameterwerte**
Anzahl PE	40	**Pufferverwaltung:**	
CPU-Leistung pro PE	20 MIPS	Seitengröße	8 KB
Mittl. Anzahl Instruktionen:		Puffergröße pro PE	250 Seiten (2MB)
BOT	25000	**Kommunikationsnetzwerk:**	
EOT	25000	Paketgröße	128 Bytes
Initialisierung von Scan-Aus-		mittlere Übertragungszeit	8 Mikrosekunden
führungseinheiten	25000	**Relation A:**	(50MB)
E/A	3000	Anzahl Tupel	250.000
Nachricht senden	5000	Tupelgröße	200 Bytes
Nachricht empfangen	10000	Tupeln pro Seite	40
Kopieren einer 8 KByte		Indextyp	B^*-Baum mit Cluster-Bildung
Nachricht in Hauptspeicher	5000	Höhe der Indexstruktur	2
Scan-Objektreferenz	1000	Speichermedium	Hauptspeicher/Platte
Mischen von n Tupeln	n * 100	PE-Zuordnung	1 bis 40
Plattengeräte:		**Scan-Anfragen**	auf Relation A
Kontroller-Belegungszeit	1 ms (pro Seite)	Zugriffsmethode	via Index/ sequentiell
Übertragungszeit pro Seite	0.4 ms	Selektivität	0.1%-10% der Tupel
mittl. Plattenzugriffszeit	15 ms	Scan-Ausführungseinheiten	1-40

Tabelle 3: Datenbank-, Anfrage- und Systemparameter.

Platten allokiert, so gilt in Abhängigkeit von der Zugriffsmethode:

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

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

Die lokale Puffergröße beträgt in unserem Versuchen 250 Seiten. Die Tupel von Relation A sind auf 6250 Seiten verteilt (Blockungsfaktor 40). Für 25 oder mehr PE paßt Relation A 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 3 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.

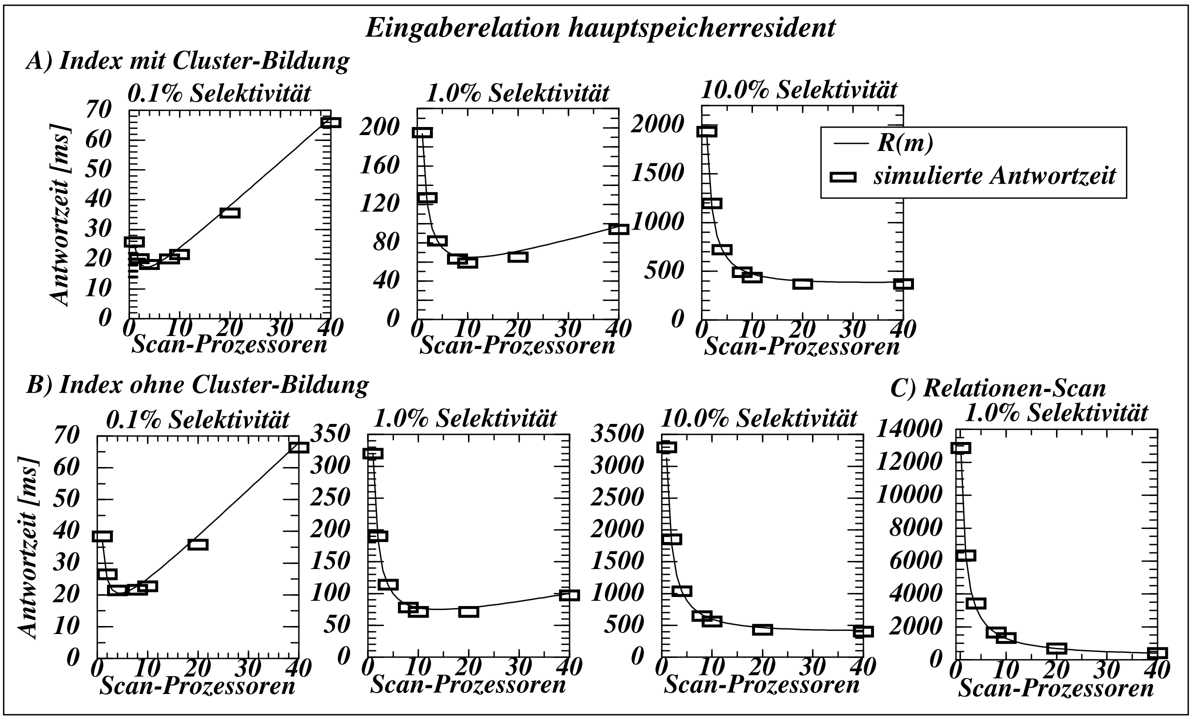


Bild 3: Einfluß von Zugriffsmethode und Anfrageselektivität auf die Effektivität der Parallelisierung von Scan-Operatoren bei hauptspeicherresidenter Speicherung der Eingaberelation.

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[7]. Dieses Verhalten schlägt sich auch in den zugehörigen Speedup-Kurven nieder (Bild 4). Lediglich im Falle des Relationen-Scans kann über eine große PE-Anzahl hinweg nahezu linearer Speedup erzielt werden (Speedup 19 bei

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

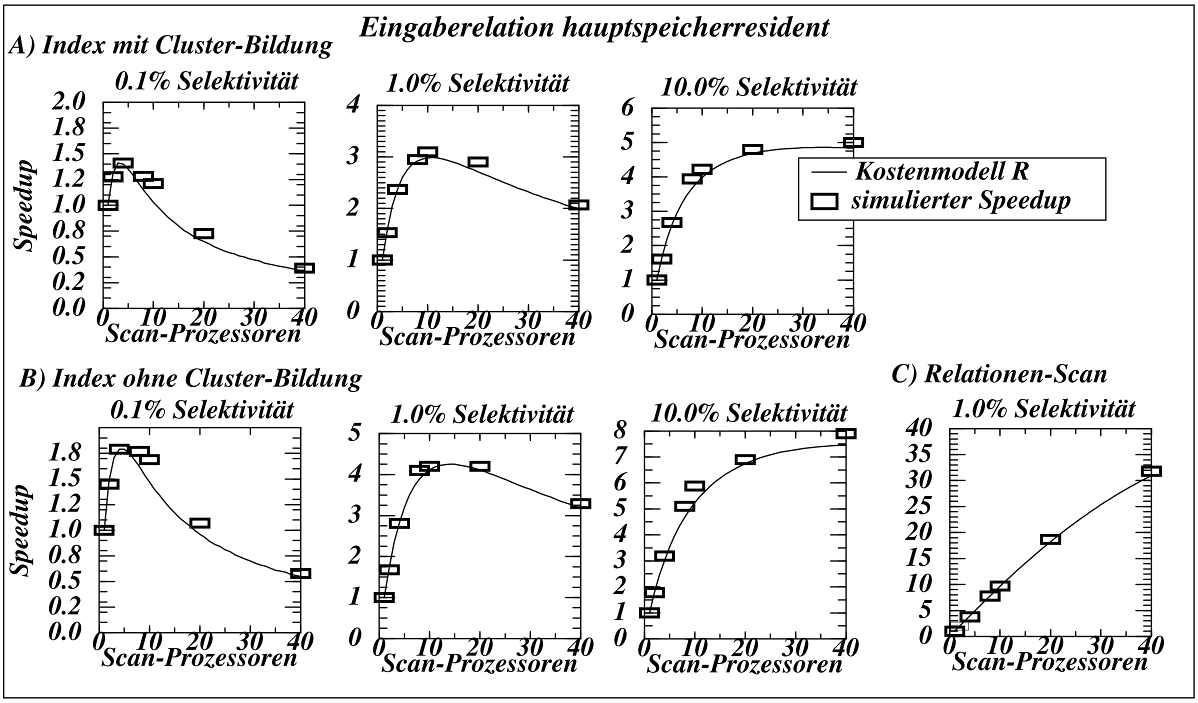


Bild 4: Einfluß von Zugriffsmethode und Anfrageselektivität auf den Antwortzeit-Speedup.

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:

$$m_{opt} = \begin{cases} \sqrt{\dfrac{0.05018 \times M}{1.5}} & \text{Relationen-Scan (1.0\% Selekt.)} \\ \sqrt{\dfrac{0.06875 \times M}{1.5}} & \text{Index mit Cluster-Bildung} \\ \sqrt{\dfrac{0.05937 \times M}{1.5}} & \text{Index ohne Cluster-Bildung} \end{cases}$$

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

Zugriffsmethode	Selektivität [%]	M	m_{opt}
Index mit Cluster-Bildung	0.1	250	3
	1.0	2500	11
	10.0	25000	34
Index ohne Cluster-Bildung	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.

Ist das typische Anfrageprofil (Anfragetypen und deren Anteile an der gesamten Anfragelast) auf einer gegebenen Relation *A* bekannt, so können wir auf diese Weise eine Abschätzung eines geeigneten Datenverteilungsgrades für diese Relation treffen. Werden beispielsweise 40% der Anfragen auf Relation *A* 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 *A* (vgl. Tabelle 6)[8].

Anfragetypen mit Zugriffsmethode	Selektivität [%]	Anteil an der Gesamtlast [%]	m_{opt}	m_{opt} (global)
Index (mit Cluster-Bildung)	1.0	40.0	11	0.4 * 11
Index (ohne Cluster-Bildung)	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.

Plattenallokation der Eingaberelation

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

Bild 5 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 Index ohne Cluster-Bildung) 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

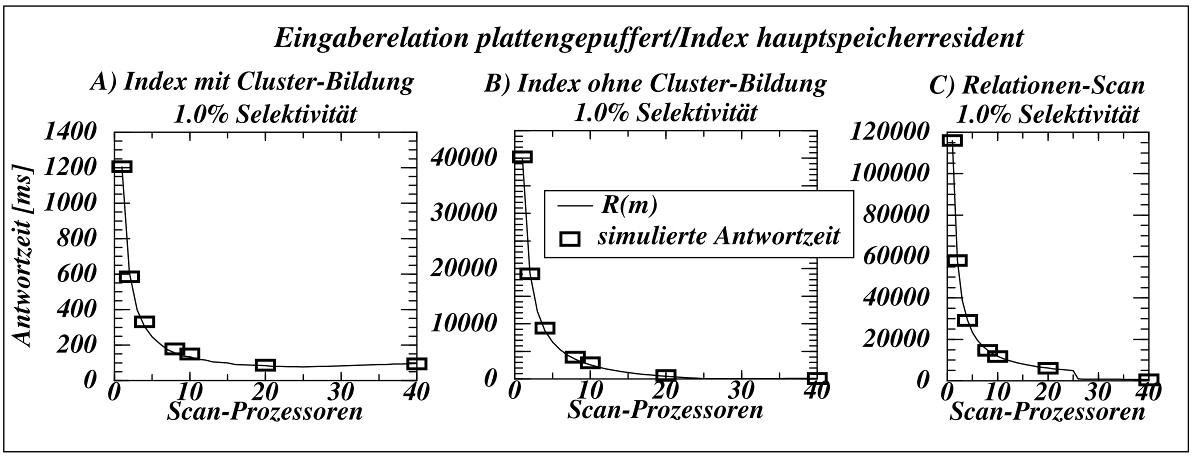


Bild 5: Einfluß von Zugriffsmethode und Anfrageselektivität auf die Effektivität der Parallelisierung von Scan-Operatoren bei Plattenpufferung der Eingaberelation.

8. Hierbei ist angenommen, daß das Optimum für den Anfragemix durch Linearkombination der einzelnen Optima berechnet werden kann. Im allgemeinen kann das Optimum z.B. durch ein Programm berechnet werden, welches für jeden möglichen Datenverteilungsgrad die Antwortzeit des Anfragemixes aus den Antwortzeiten der einzelnen Anfragetypen ermittelt.

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 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 A beträgt *R*:

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

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 mit Cluster-Bildung	1.0	2500	63	25
Index ohne Cluster-Bildung	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 Übertragung des Kostenmodells auf Join-Operatoren

In analoger Weise läßt sich das Kostenmodell auf Anfragen übertragen, die zwei Relationen miteinander über eine Verbundoperation (Join) verknüpfen. Der Join-Operator verknüpft zwei Eingaberelationen *A* und *B* anhand eines Join-Attributs 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 entsprechen (sog. Equi-Join). Hierzu sind beide Relationen zunächst mit Hilfe von Scan-Operatoren zu lesen. Die von den Scan-Operatoren generierten Ausgabemengen bilden die Eingaberelationen des Joins. Vor Ausführung des Joins ist im allgemeinen eine Umverteilung der Eingaberelationen zu den Join-Ausführungseinheiten erforderlich.
Die üblicherweise zur Anwendung kommende Methode ist die *symmetrischen Partitionierung*; sie wird u.a. bei Tandem, Gamma und Teradata angewandt. Bei der symmetrischen Partitionierung werden beide Eingaberelationen nach dem Verbundattribut auf mehrere Join-Ausführungseinheiten verteilt. Die wichtigste Eigenschaft ist, daß Tupel beider Eingaberelationen derart auf die Join-Ausführungseinheiten verteilt werden, daß alle Tupel mit dem gleichen Join-Attributwert der selben Join-Ausführungseinheit zugeordnet werden. Auf diese Weise wird die voneinander unabhängige Ausführbarkeit der Join-Ausführungseinheiten garantiert, da Verbundpartner nur innerhalb der Join-Ausführungseinheiten auftreten können. Das verwendete attributwertbasierte Verteilverfahren

(i.d.R. Hash- oder Bereichspartitionierung) ist dabei unabhängig von der lokalen Join-Methode (z.B. Hash- oder Sort-Merge-Join). Bei Verwendung der symmetrischen Partitionierung können sowohl die Anzahl der Join-Ausführungseinheiten als auch deren Zuordnung zu Prozessorknoten frei gewählt werden; d.h. der Parallelitätsgrad kann dynamisch festgelegt werden und die Join-Bearbeitung kann an beliebigen Knoten erfolgen.

Wir haben in [Marek 1993] eine Kostenabschätzung der parallelen Join-Verarbeitung exemplarisch für eine Sort-Merge-Implementierung durchgeführt. Der Sort-Merge-Join basiert auf einem sukzessiven Lesen und Mischen beider Eingaberelationen (*Merging Scans*), wobei Sortierordnungen über den Verbundattributen beider Relationen ausgenutzt werden. Da im allgemeinen keine Sortierordnungen über den Verbundattributen vorliegen, müssen vor der Scan-Phase zunächst beide Relationen nach dem Verbundattribut sortiert werden. Nach erfolgter Sortierung werden Scans auf beiden Relationen eröffnet, die durch geeignetes Fortschalten für jeden Wert des Verbundattributes die zugehörigen Tupel liefern. Bei einem 1:n-Join genügt ein einmaliges Referenzieren jedes Tupels, bei einem n:m-Join müssen für jeden Join-Attributwert die n Tupeln der zweiten Relation m-fach referenziert werden[9]. In jedem Fall ist der Aufwand für diesen Teilschritt in Abhängigkeit von der Anzahl beteiligter Tupeln linear. Die CPU-Kosten des Sort-Merge-Joins setzen sich zusammen aus den Kosten für das Sortieren der Eingaberelationen (O(n log n)) und dem linearen Kostenanteil für das Mischen der sortierten Ströme.

Zur Antwortzeitabschätzung von Join-Anfragen in Abhängigkeit vom Parallelitätsgrad *m* des Join-Operators wurde in [Marek 1993] folgende Formel hergeleitet:

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

Der Parallelitätsgrad der Scan-Operatoren, die die Basisrelationen lesen, wurde dabei entsprechend der fest vorgegeben Datenverteilung als konstant angesehen. Der Antwortzeitanteil der Scan-Operatoren ist demzufolge in dem konstanten Antwortzeitanteil a enthalten. Die Kardinalität der Join-Eingaberelationen beträgt M_1 bzw. M_2.
Jede Join-Ausführungseinheit verarbeitet $1/m$ der am Verbund teilnehmenden Tupel. Die Koeffizienten d_i beschreiben den Aufwand für das Sortieren der Eingaberelationen je Join-Ausführungseinheit, die Faktoren c_i enthalten vor allem die Kosten für das Empfangen der Eingabemengen und das sukzessive Lesen und Mischen der sortierten Eingaberelationen. Die Kosten der Parallelverarbeitung (Initialisierung der Join-Ausführungseinheiten etc.) werden in dem Koeffizienten b berücksichtigt.
Auf eine detaillierte Koeffizientenbestimmung in Abhängigkeit von Anfrage- und Systemparametern sei an dieser Stelle aus Platzgründen verzichtet. Der interessierte Leser sei diesbezüglich auf [Marek 1993] verwiesen.

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 Kostenmodells 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

9. Bei einem m:n-Verbund (natürlicher Verbund) existieren zu jedem Tupel der äußeren Relation maximal m Verbundpartner in der inneren Relation; zu jedem Tupel der inneren Relation maximal n Verbundpartner in der äußeren Relation. Für m=1 spricht man von einem funktionalen Verbund.

Ermittlung des optimalen Parallelitätsgrades von Join-Operatoren genutzt werden. Insbesondere kann der Einfluß des Speichermediums (Hauptspeicher versus Platte) der Operanden berücksichtigt werden.
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 Anfrage- (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-Parallelitätsgrads eine lineare Verbesserung dieses Kostenterms; im Falle logarithmischer Operatoren ist eine superlineare Beziehung gegeben. Dem letzten der 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 Simulationsmodells durchgeführte Antwortzeitversuche zur Parallelisierung von Scan-Operatoren haben die Übereinstimmung von Kosten- und Simulationsmodell gezeigt. Wir haben gesehen, daß sich die beobachteten Antwortzeiten bei der parallelen Scan-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 Simulationsmodells 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 Modells 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. Teilaspekte dieser 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.

Mackert, L.; Lohman, G. 1989: Index Scans Using a Finite LRU Buffer: A Validated I/O Model. *ACM Trans. on Database System* 14(3), 401-424.

Marek, R. 1993: Ein Kostenmodell der parallelen Anfragebearbeitung in Shared-Nothing-Datenbanksystemen. Technischer Bericht 3/93, Universität Kaiserslautern, Fachbereich Informatik, Mai 1993.

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. *Proc. 19th Int. Conf. on Very Large Data Bases*, August 1993, Dublin, Ireland.

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 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.

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.

Waters, S.J. 1976: Hit Ratio. *Computer Journal* 19(1), 21-24.

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.

Yao, S.B. 1977: Approximating Block Accesses in Database Organizations. *Communications of the ACM* 20(4), 260-261.