Entwicklung und Test von Verfahren zur Bewertung von Anfrageoptimierungen

Studienarbeit

Universität Rostock, Fachbereich Informatik

vorgelegt von: Michael Jonas

Betreuer: Prof. Dr. Andreas Heuer
Dr. Ing. Holger Meyer
Dipl.-Ing. Astrid Lubinski
Dipl.-Inf. Joachim Kröger

Inhaltsverzeichnis

Kapitel 1

Einführung

Dieses Kapitel veranschaulicht das Kernproblem, mit dem sich diese Studienarbeit befaßt, und dessen allgemeine, als auch spezielle Bedeutung für die Projekte HEaD [1], CROQUE [2] und MoVi [3]. Das Kapitel endet mit eine kurzen Übersicht über die Gliederung der nachfolgenden Kapitel.

1.1 Einleitung

Die Anfrageverarbeitung besitzt eine zentrale Bedeutung für ein Datenbanksystem (DB S), da die Effizienz eines DBS direkt von der Leistungsfähigkeit der Anfrageverarbeitung abhängt. Eine wichtige Rolle spielt dabei der Anfrageoptimierer. Dieser hat die Aufgabe, eine Anfrage in einer deklarativen Anfragesprache (z.B. SQL) in einen Anfrageplan umzuformen, der gemäß dem vorgegebenen Optimierungsziel optimal ist. Es stellt sich dabei die Frage, ob der Anfrageoptimierer seine Aufgabe gut erfüllt hat oder nicht. Möglicherweise ist der erzeugte Anfrageplan gut oder aber auf Grund von Fehlern in einer oder mehreren Optimiererkomponenten schlecht. Da der Optimierungsprozeß selbst schon Zeit in Anspruch nimmt, kann es sein, daß die sofortige Ausführung des initialen Plans besser gewesen wäre, als die Ausführung des als optimal ermittelten Anfrageplans, im Anschluß an die durchgeführte Optimierung. In dieser Arbeit sollen Ansätze zur Bewertung von Anfrageoptimierungen vorgestellt werden, die eine nachfolgende Adaption des Optimierungsprozesses ermöglichen sollen.

1.2 Motivation

Nur sehr wenige Anfrageoptimierer wurden bisher auf ihre reale Performance hin analysiert. Detaillierte Untersuchungen wurden für den Optimierer von System

[1] Heterogeneous, Extensible and Distributed Relational Database Managment System
[2] Cost- and Rulebased Optimization of object-oriented Queries
[3] Mobile Visualisierung

R in [ASK80] und für System R* in [ML86a] für lokale Anfragen und in [ML86b] für verteilte Anfragen durchgeführt. Eine genauere Analyse des Optimierers und seiner Performance ermöglicht das Erkennen von Fehlern im Anfrageoptimierer bzw. in seinen Komponenten, beispielsweise Ungenauigkeiten im Kostenmodell. Auf Grundlage dadurch gewonnener Erkenntnisse läßt sich der Anfrageoptimierer verbessern oder an andere Gegebenheiten, zum Beispiel eine andere Datenbankkonfiguration oder Veränderung der Anfrageprofile, anpassen. Idealerweise sollte diese Anpassung ohne Einwirkung des Datenbankherstellers oder -administrators erfolgen, d.h. der Anfrageoptimierer soll selbstständig lernen und sich selbst optimieren. Ein adaptives System dieser Art ist stark von einer effektiven und genauen Bewertung seiner Arbeit abhängig. Im Folgenden sollen einige Projekte des Fachbereichs Informatik der Universität Rostock, die von den Ergebnissen dieser Arbeit profitieren, kurz vorgestellt werden.

1.2.1 CROQUE

Im Rahmen dieses DFG-Projekts werden Optimierungsverfahren für Objekt-Datenbanksysteme entwickelt. Im Mittelpunkt der Arbeiten stehen Untersuchungen zu Verfahren des 'Query-Rewriting' für regelbasierte und algebraische Anfragesprachen, Auswertungsverfahren für objektorientierte Anfragen sowie die Optimierung des physischen Entwurfs von Objekt-Datenbanken bezüglich eines gegebenen Anwendungsprofils.

Ziel des Projektes ist die Entwicklung eines selbstoptimierenden, objektorientierten Anfragebearbeitungssystems [HK96]. Der Aspekt der Selbstoptimierung setzt voraus, daß das System eine Rückmeldung über die Qualität seiner "Arbeit " bekommt. Dazu ist es notwendig die Optimierungen der an das System gestellten Anfragen qualitativ einzuordnen.

1.2.2 MoVi

Das Projekt MObile VIsualisierung hat zum Ziel, eine offene und adaptive Visualisierungsarchitektur für ein global verteiltes Informationssystem zu schaffen, das den mobilen Zugang zu allen denkbaren Daten und Diensten, dem Infoverse, ermöglicht.
Im Mittelpunkt des im Projekt verfolgten mobilen Szenarios steht der Nutzer, dessen Mobilität durch den Wechsel zwischen stationären und mobilen Geräten und Netzen entsteht. Die dynamischen Faktoren der mobilen Umgebung des Nutzers (z.B. Aufenthaltsort, Übertragungsbandbreite) werden durch den Verarbeitungskontext beschrieben und klassifiziert. Die Informationsbereitstellung in mobilen Umgebungen wirft Probleme auf, die von kommerziellen DBS nicht erfüllt werden können. Dies betrifft beispielsweise Adaptivität, die abhängig von der Dynamik der Kontexte vor allem eine dynamische Anfragemodifikation, -optimierung und -aggregation erfordert. Bei einer kontextabhängigen Anfrageoptimierung kann das

Optimierungsziel durch wechselnde Ressourcen der benutzten Geräte und Netze und andere Einflüsse veränderlich sein, was sich in einer dynamischen Kostenfunktion des Optimierers widerspiegelt. Deshalb müssen neben klassischen Optimierungszielen, wie Antwortzeitminimierung und Durchsatzmaximierung, weitere, wie die Anpassung an begrenzte Ressourcen, gefunden werden. Abhängig vom konkreten Kontext müssen die Optimierungsziele gewichtet werden, so daß die jeweils relevanten Ziele Beachtung finden [HL96]. Die Anpassung der Optimierung an den jeweiligen Nutzerkontext sollte dynamisch, ohne Einwirkung des Menschen, erfolgen. Die Steuerung der adaptiven Optimierung erfordert eine Bewertung der konkreten Optimierungen.

1.2.3 HEaD

Gegenstand des HEaD-Projektes sind Untersuchungen auf dem Gebiet der parallelen Anfrageverarbeitung in heterogenen, verteilten DBMS. Neben horizontaler Parallelität, die durch die Datenfragmentation und -replikation impliziert wird, liegt der Schwerpunkt auf Pipelining und Partitionierung. In HEaD werden Methoden der Lastverteilung und -balancierung benutzt, das Level der Multiprogrammierung und die Zuweisung freier Operatoren zu Rechnerknoten zu kontrollieren, um die optimale Bearbeitung komplexer Anfragen in einem verteilten DBMS zu gewährleisten. Die Optimierung soll durch Einbeziehung unbelasteter Netzwerkknoten in die Anfrageverarbeitung zu einer Verringerung der Ausführungszeit und einem allgemeinen “speedup“ führen [Lan97].

1.3 Zielstellung

Die Arbeit beschäftigt sich damit, Verfahren zur Bewertung von Optimierungen zu entwickeln. Dafür müssen je nach Optimierungsziel Kriterien für eine Bewertung ermittelt werden.
Im Zusammenhang mit der Bewertung der Qualität einer Optimierung liegt ein wesentlicher Schwerpunkt auf der Untersuchung des Anfrageoptimierers selbst, da dieser bzw. seine Einzelkomponenten die Qualität einer Optimierung direkt beeinflussen. Dabei wird versucht, wichtige Bestandteile des Anfrageoptimierers zu bestimmen und Methoden zu erarbeiten, die es ermöglichen, diese zu validieren.

1.4 Überblick

Kapitel 2 gibt einen kurzen Überblick über den allgemeinen Prozeß der Anfrageoptimierung und beschreibt davon ausgehend den Aufbau eines Anfrageoptimierers und die Funktion seiner wichtigsten Komponenten.
In Kapitel 3 werden vorhandene Methoden zur Bewertung von Anfrageoptimierern und Optimierungen untersucht. Der Schwerpunkt liegt dabei auf der Be-

trachtung von Benchmarks, die besonders den Anfrageoptimierer testen.
Kapitel 4 beschreibt Ansätze zur Bewertung konkreter Optimierungen für eine fest vorgegebene Anfrage und bewertet ihre praktische Anwendbarkeit.
In Kapitel 5 werden wichtige Komponenten eines Anfrageoptimierers genannt und Ansätze zu ihrer Qualitätsbewertung vorgestellt.
Kapitel 6 faßt wesentliche Fakten dieser Arbeit zusammen und gibt einen kurzen Ausblick.

Kapitel 2

Anfrageverarbeitung in Datenbanksystemen

In diesem Kapitel werden Grundlagen der Anfrageverarbeitung erklärt. Es werden die Rolle der Anfrageverarbeitung in DBS erläutert und ihre einzelnen Phasen beschrieben. Davon ausgehend werden die Komponenten des Anfrageprozessors identifiziert und der Prozess der Anfrageverarbeitung erläutert. Die Gliederung und der Inhalt des folgenden Kapitels orientieren sich an [Mit95] [1], das eine gute Einführung in die Anfrageverarbeitung gibt.

Anschließend wird die Realisierung der Anfrageverarbeitung in den Projekten HEaD, MoVi und CROQUE kurz beschrieben.

2.1 Aufgaben der Anfrageverarbeitung

Eines der hervorstechenden Merkmale relationaler DBS ist die deskriptive Nutzerschnittstelle. Im Gegensatz zu navigierenden Anfragesprachen in hierarchischen oder netzwerkartigen DBS, spezifiziert der Nutzer nur *welche* Daten er haben will und nicht *wie* diese beschafft werden sollen. Auf der einen Seite vereinfacht sich die Anfrageformulierung drastisch, auf der anderen Seite bleibt die Konstruktion des Anfrageergebnisses völlig dem DBS überlassen. Neben Namensauflösung (externe/interne Namen) und Formatkonversion (externes/internes Datenformat) hat das DBS nun die Aufgaben Anfrageoptimierung und -ausführung zu übernehmen. Davon ausgehend kann man aus Sicht der Anfrageverarbeitung in einem DBS den *logischen DB-Prozessor* und den *physischen DB-Prozessor* unterscheiden. Der logische DB-Prozessor übernimmt die Integritäts- und Zugriffskontrolle und die Anfrageverarbeitung; er wird deswegen im Folgenden Anfrageprozessor genannt. Die Aufgabe der physischen DB-Prozessors ist es, das Ergebnis der Anfrage gemäß des vom logischen DB-Prozessor ermittelten Anfrageplanes zu beschaffen.

[1] Seite 24 ff.

Auf die Anfrageausführung soll im weiteren nicht tiefergehend eingegangen werden, da sie für das Thema dieser Arbeit nicht weiter interessant ist. Von daher soll der Schwerpunkt des Kapitels auf der Betrachtung des Anfrageprozessors liegen.

Die Aufgabe des Anfrageprozessors ist die Erzeugung eines Anfrageplans, der auf der DB mit minimalen Kosten ausgeführt werden kann. die Suche dieses Anfrageplan kann unterschiedliche Optimierungsziele realisieren. In einer Einzelnutzerumgebung könnten das sein:

- Bearbeitung einer Anfrage mit minimalen Kosten
- Bearbeitung einer Anfrage mit minimaler Bearbeitungszeit

In einer Mehrnutzerumgebung, wo viele Anfragen gleichzeitig bearbeitet werden müssen, könnten die Leistungsanforderungen folgendermaßen aussehen:

- Minimierung der Antwortzeit (Zeit von Entgegennahme der Anfrage durch den Anfrageprozessor bis zur Lieferung der ersten Ergebnisse)
- Durchsatzmaximierung
- Einhaltung von Antwortzeitschranken

Die genannten Optimierungsziele sind kombinierbar. Bei Ad-hoc-Anfragen beispielsweise sollen Antwortzeitschranken eingehalten werden, auch wenn das globale Optimierungsziel sonst die Durchsatzmaximierung ist.

2.2 Übersicht über die Anfrageverarbeitung

Der folgende Abschnitt stellt die Verarbeitung einer Anfrage im Überblick dar. Ausgehend von einer Phaseneinteilung der Anfrageverarbeitung wird eine Komponentensicht des Anfrageprozessors entwickelt. Im Anschluß werden die Phasen der Anfrageverarbeitung, der Anfrageverarbeitungsprozess, genauer beschrieben.

2.2.1 Phasensicht

Um die Komplexität der Anfrageverarbeitung beherrschbar zu machen, teilt man den Prozess der Anfrageverarbeitung in mehrere Phasen auf. Jede dieser Phasen hat eine Teilaufgabe bei der Anfrageverarbeitung zu lösen.

- Schritt 1: *Interndarstellung der Anfrage*
  Neben der Prüfung auf syntaktische Richtigkeit der Anfrage werden Zugriffsrechte und Integrität kontrolliert. Für die weitere Anfrageverarbeitung muß die Anfrage in eine interne Darstellung umgewandelt werden. Diese Umwandlung sollte schnell und einfach auszuführen sein, andererseits muß die Interndarstellung Anforderungen wie Erweiterbarkeit und Umformbarkeit erfüllen.

- Schritt 2: *Anfrageumformung*
  Die interne Darstellung wird mit Hilfe logischer Transformationen standardisiert, vereinfacht und verbessert. Redundanzen in der Interndarstellung sollen erkannt werden und Umformungen durchgeführt werden, die die Weiterverarbeitung erleichtern und damit die Ausführung effizienter gestalten.

- Schritt 3: *Plangenerierung*
  In dieser Phase werden mögliche Anfragepläne generiert. Die umgeformte interne Darstellung der Anfrage wird auf verschiedene Folgen von Elementaroperationen abgebildet. Da die Anzahl der in Frage kommenden Anfragepläne oft sehr groß ist, ist es wichtig, nur potentiell effiziente Pläne auszuwählen.

- Schritt 4: *Kostenabschätzung*
  Die Kosten für jeden Anfrageplan werden berechnet und der billigste ausgewählt.

Diese Schritte bilden im wesentlichen die Vorgehensweise, um zu einer Anfrage den hoffentlich optimalen Anfrageplan zu finden, der anschließend ausgeführt wird und dem Nutzer das gewünschte Anfrageergebnis liefert.

2.2.2 Komponentensicht

Ausgehend von der Phasensicht der Anfrageverarbeitung kann nun die entsprechende Komponentensicht für den Anfrageprozessor entwickelt werden, die in Abbildung 2.1 dargestellt wird. Der Anfrageprozessor ist dabei in eine *Übersetzungskomponente* und eine *Optimierungskomponente* unterteilt. Die *Übersetzungskomponente* generiert mit Hilfe von Informationen aus den externen und Konzeptuellen Schemata der Datenbank die Interndarstellung und realisiert somit Schritt 1. Die *Optimierungskomponente* ermittelt für die Anfrage in Interndarstellung den Anfrageplan. Die Anfragerestrukturierung übernimmt Schritt 2, die Anfragetransformation realisieren die Schritte 3 und 4. Für die Anfragetransformation werden, um gültige Anfragepläne zu generieren, Informationen des internen Schemas und für die Kostenabschätzung und Auswahl des besten Anfrageplans Informationen zum Kostenmodell sowie zu DB-Statistiken benötigt. Die *Ausführungskomponente* realisiert die Anfrageausführung. Je nach Optimierungszeitpunkt, worauf im folgenden Abschnitt „Optimierungszeitpunkte“ eingegangen werden soll, wird der Anfrageplan entweder interpretiert oder compiliert und ausgeführt und das Anfrageergebnis anschließend dem Nutzer bereitgestellt.

Optimierungszeitpunkte Die drei in Abbildung 2.1 gezeigten Komponenten realisieren die drei aufeinanderfolgenden Schritte der Anfrageverarbeitung.

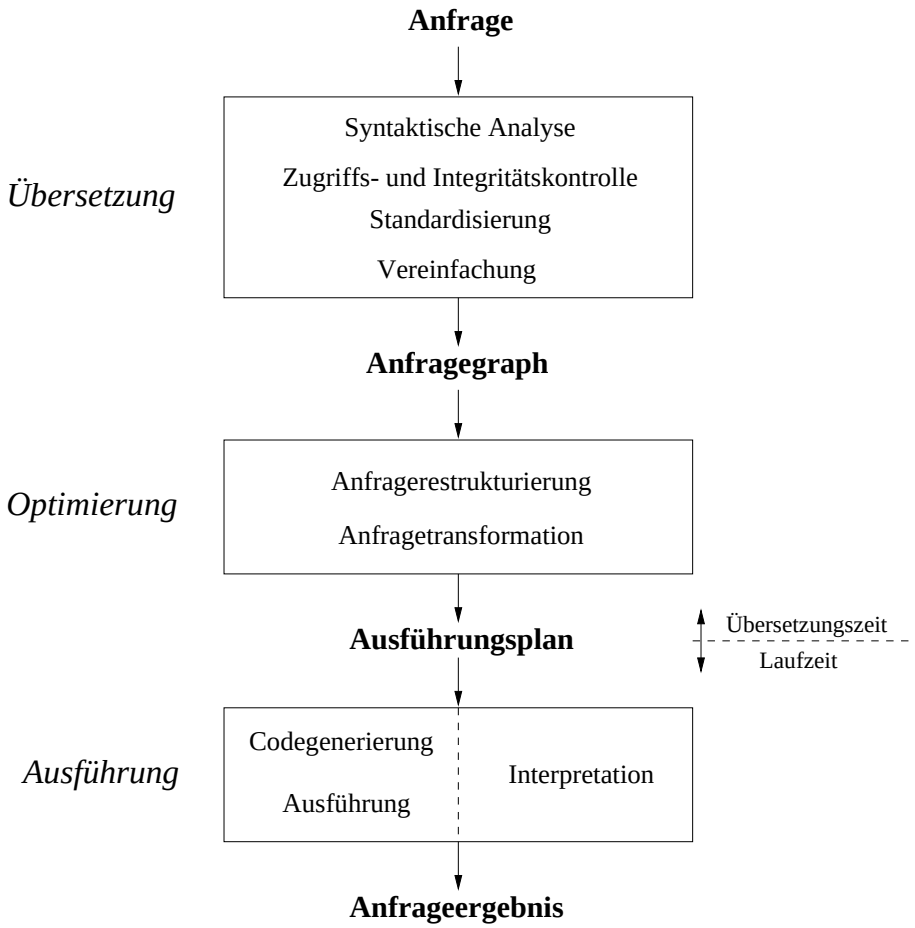

Abbildung 2.1: Komponentensicht des Anfrageprozessors [Mit95]

Dabei wird jedoch keine Aussage über die Zeitpunkte, wann welche Phase auszuführen ist, getroffen. Eine Möglichkeit ist, alle drei Schritte sofort hintereinander auszuführen. Das bedeutet, die Anfrage wird sofort nach der Entgegennahme durch den Anfrageprozessor übersetzt, optimiert und direkt daran anschließend ausgeführt. Da alle Phasen zur Laufzeit abgewickelt werden, nennt man diesen Ansatz *dynamische Optimierung*. Sämtliche Optimierungsentscheidungen werden zur Laufzeit getroffen, d.h. es werden die aktuellsten DB-Metadaten, wie internes Schema, Statistiken usw. genutzt. Da die Bindung von Anfrage zur Datenbank zum spätest möglichen Zeitpunkt erfolgt, wird ein hoher Grad an Unabhängigkeit erreicht. Besonders bei interaktiven DB-Anfragen, also Ad-hoc-Anfragen, ist eine unmittelbar aufeinanderfolgende Übersetzung, Optimierung und Ausführung der Anfrage erforderlich. Der große Nachteil der dynamischen Optimierung tritt jedoch bei in Anwendungsprogramme eingebetteten Anfragen zu Tage, da jede erneute Ausführung einer DB-Anfrage eine komplette Wiederholung der drei Anfrageverarbeitungsschritte nach sich zieht.

Dieses Problem läßt sich dadurch lösen, daß man Übersetzung und Optimierung der Anfrage von deren Ausführung zeitlich trennt. Die Anfrage wird nur einmal übersetzt und optimiert. Das Ergebnis dieser beiden Phasen wird gespeichert und kann dann beliebig oft ausgeführt werden. Durch die häufigere Ausführung der Anfrage gleicht sich der hohe Optimierungsaufwand aus. Im Gegensatz zur *dynamischen Optimierung* wird dieser Ansatz als *statische Optimierung* bezeichnet. Gleichzeitig mit der Übersetzung des Anwendungsprogramms werden auch die DB-Anweisungen in Ausführungspläne 'übersetzt', die zur Laufzeit nur noch von der Anfrageausführungskomponente auszuführen sind. Das bedeutet, daß die Anfrage erheblich früher an die Datenbank gebunden wird als bei der dynamischen Optimierung. Dies kann dazu führen, daß Anfragepläne aufgrund von Änderungen in den Datenstrukturen und Zugriffspfaden ungültig werden und nachoptimiert werden müssen. Auch wenn der Anfrageplan nicht ungültig wurde, kann eine Nachoptimierung notwendig sein, falls neue und bessere Zugriffspfade angelegt worden sind oder die DB-Statistiken wegen größeren Datenbewegungen Ungenauigkeiten aufweisen. Die statische Optimierung ist also besonders geeignet für DBS mit langfristig stabilen Daten und Statistiken.

Die dritte Möglichkeit versucht, die Vorteile der statischen und dynamischen Optimierung zu vereinen. Das heißt, die Anfrage wird statisch optimiert und zur Ausführungszeit werden Teile der Anfrage nachoptimiert, falls sich zum Beispiel die DB-Statistiken oder Zugriffspfade geändert haben. Dadurch wird die Effizienz des statischen und die Flexibilität des dynamischen Ansatzes kombiniert.

Aus den vorherigen Betrachtungen ist ersichtlich, daß der Zeitpunkt der Optimierung für die Effizienz der Anfrageoptimierung eine wichtige Rolle spielt. Je nach Anwendung muß entschieden werden, welcher Optimierungszeitpunkt gewählt wird. Abbildung 2.2 faßt abschließend die Eigenschaften der Optimierungszeitpunkte zusammen.

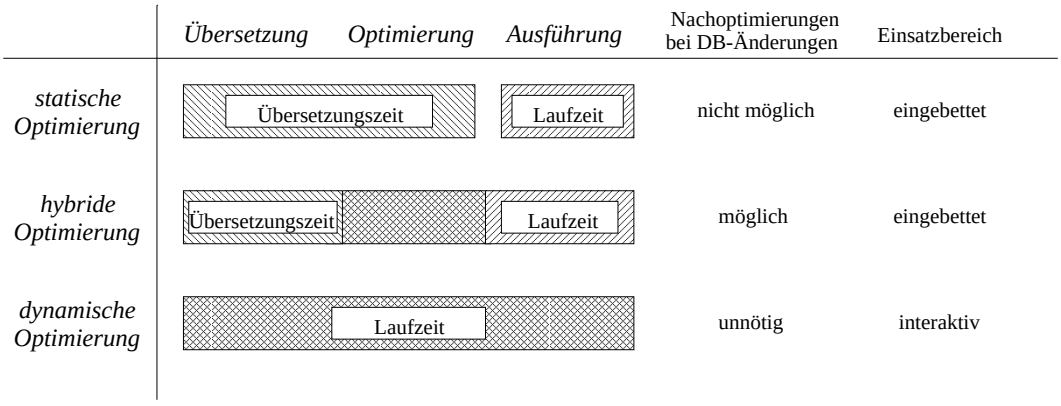

Abbildung 2.2: Eigenschaften verschiedener Optimierungszeitpunkte [Mit95]

2.2.3 Der Anfrageverarbeitungsprozess

In diesem Abschnitt sollen die Phasen Übersetzung und Optimierung genauer betrachtet werden. Die Ausführungsphase soll im weiteren unbeachtet bleiben, da sie für den Inhalt dieser Arbeit unerheblich ist.

Übersetzung

Aufgabe der Übersetzungskomponente ist es, die Anfrage in eine für den folgenden Optimierungsschritt günstige Interndarstellung umzuwandeln. Gleichzeitig muß geprüft werden, ob eine im Sinne des externen und konzeptuellen Schemas korrekte Anfrage vorliegt. Dazu werden Metadaten, wie Schemabeschreibungen und Informationen zu Zugriffsrechten, benötigt.

Das Problem ist, ein *Darstellungsschema* zu finden, mit dem eine geeignete Interndarstellung der Anfrage ermöglicht wird. Dieses Darstellungsschema sollte den folgenden Anforderungen genügen.

- *Prozeduralität*
  Im Unterschied zur gestellten Anfrage, die in der Regel in deskriptiver Form vorliegt (wie z.B. SQL), muß die Interndarstellung die Anfrage in prozeduraler Form darstellen. Eine Möglichkeit ist, die deskriptive Form der Anfrage in eine der Relationenalgebra ähnliche Darstellungsform umzuwandeln. Das bedeutet, die deklarative Form der Anfrage wird übersetzt in eine Folge von Algebraoperatoren, die den Algorithmus zur Beschaffung des Ergebnisses darstellen. Die meisten DBS verwirklichen diesen Ansatz und stellen die üblichen Algebraoperatoren, wie Projektion, Selektion, Join und Mengenoperationen, bereit. Manche DBS bieten darüber hinaus noch andere Operatoren für Gruppierungen, Sortierung oder Schachtelung von Relationen an.

- *Flexibilität*
  Die Interndarstellung sollte hinsichtlich Erweiterungen der DB-Sprache und hinsichtlich der Transformationen der nachfolgenden Optimierungsschritte flexibel sein. Im ersten Fall sollte die Interndarstellung die Erweiterung um neue Operatoren oder das Austauschen von Operatoren zulassen. Der zweite Gesichtspunkt der Flexibilität ist, das die Transformationen im Übersetzungsschritt und in den folgenden Schritten auf der Interndarstellung ausführbar und die Ergebnisse damit darstellbar sind.

- *Effizienz*
  Um zu entscheiden, welche Transformationen angewendet werden sollen, ist es notwendig, die Interndarstellung zu durchsuchen und in ihr zu navigieren. Daher ist es für den Übersetzungschritt und die folgenden Schritte wichtig effizient auf die Interndarstellung zugreifen zu können.

Diese Überlegungen zeigen, daß die Interndarstellung der Anfrage die zentrale Datenstruktur für die Anfrageübersetzung und -optimierung darstellt und daß die Wahl des Darstellungsschemas entscheidend für Effizienz und Erweiterbarkeit des Anfrageprozessors ist. Im folgenden sollen einige der vielen in der Literatur und DBS existierenden Darstellungsschemata vorgestellt werden. Als Beispiel sei folgende SQL-Anfrage gewählt:

```
SELECT   R1.A, R1.B
FROM     R1, R2
WHERE    R1.C = R2.Z
AND      R1.B = 10
AND      R2.Y = 20
```

Relationenalgebra In der Mathematik wird eine Algebra durch einen Wertebereich und darauf definierten Operationen charakterisiert. Bei der Relationenalgebra sind die Werte die Inhalte der Datenbank und die Operatoren Funktionen zur Berechnung der Anfrageergebnisse. Eine minimale Relationenalgebra bestünde z.B. aus einer Datenbank mit den Operationen Projektion (PROJ), Selektion (SEL), Verbund (JOIN), Vereinigung (UNION), Differenz (MINUS) und Umbenennung (REN). Hier die Beispielanfrage in der minimalen Relationenalgebra.

```
PROJ(JOIN(SEL(R1, B = 10),
          SEL(R2, Y = 20),
          R1.C = R2.Z),
     {R1.A, R1.B})
```

Relationenkalkül Ein Kalkül ist eine formale logische Sprache zur Formulierung von Aussagen. Ein derartiges Kalkül dient hier zur Formulierung von DB-Anfragen, indem Datenbankinhalte als Belegungen von Prädikaten einer Logik und Anfragen als abgeleitete Prädikate angesehen werden. Eine mögliche Darstellung der Beispielanfrage in Kalkülform ist hier dargestellt.

```
(r1.A, r1.B OF EACH r1 IN R1
            EACH r2 IN R2:(r1.C = r2.Z AND
                           r1.B = 10 AND
                           r2.Y = 20))
```

Operatorgraph Der Operatorgraph spiegelt eine prozedurale Darstellung der Anfrage wider. Die Blattknoten stellen Basisrelationen und Nicht-Blattknoten Operatoren der Relationenalgebra dar. Die gerichteten Kanten geben die Richtung des Datenflusses an. Jeder Nicht-Blattknoten beschreibt das Ergebnis des ihm zugrundeliegenden Teilgraphen. Der oberste Knoten des Operatorgraphen definiert das Anfrageergebnis. Abbildung 2.3 zeigt einen möglichen Operatorgraphen für die Beispielanfrage gemäß obiger Darstellung.

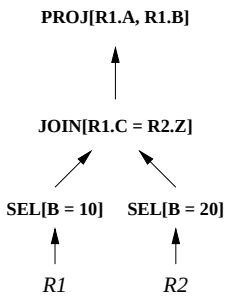

Abbildung 2.3: Operatorgraph der Beispielanfrage

Der Übersetzungsvorgang Die Übersetzung der Anfrage gliedert sich in vier Schritte. Beim *Parsen* wird die Anfrage lexikalisch und syntaktisch analysiert, d.h. die Korrekteit der Anfrage wird hinsichtlich der Sprachgrammatik überprüft. Der nächste Schritt ist die *semantische Analyse*. Hier wird die Anfrage mit Hilfe des konzeptuellen und externen Schemas auf Existenz und Gültigkeit der verwendeten Relationen, Sichten und Attribute überprüft. Falls Sichten verwendet werden, wird jedes Auftreten von Sichtnamen durch die jeweilige Sichtdefinition ersetzt. Weiterhin wird die Zugriffskontrolle durchgeführt und die semantische Korrektheit, sowie die Typkorrektheit der verwendeten Prädikate geprüft. Im nachfolgenden Schritt, der *Standardisierung*, soll die vom Anwender beliebig strukturierte Anfrage in eine Standardform überführt werden. Dabei gibt es zwei Ebenen der Standardisierung zum ersten die der Qualifikationsbedingungen, die in SQL in der `WHERE`-Klausel formuliert werden, und zum zweiten die Ebene der Anfrage mit geschachtelten Unteranfragen und Anfrageausdrücken. Als Standardform für den Qualifikation wird in den meisten Fällen eine disjunktive oder konjunktive Normalform gewählt. Standardisierung auf Anfrageebene erfordert die Eliminierung aller quantifizierten Unteranfragen durch Umwandlung in Joinanfragen. Dies bedeutet meist die Erzeugung einer Prenex-Form. Der letzte Schritt des Übersetzungsvorgang ist die *Vereinfachung*. Ziel dieses Schrittes ist die Eliminierung von Redundanzen aus dem Selektionsausdruck, die leicht durch die Verwendung von Sichten entstehen können. Viele dieser Redundanzen können durch Anwendung der Idempotenzregeln für Boolesche Ausdrücke auf die Selektionsausdrücke eliminiert werden. Eine weitere Technik der Vereinfachung ist die Bildung der transitiven Hülle über die Selektionsbedingungen.

Ergebnis des Übersetzungsschritts ist eine standardisierte und normalisierte Interndarstellung der Anfrage, die den Ausgangspunkt für die weitere Optimierung darstellt.

Optimierung

Die Aufgabe der Optimierungskomponente ist es, eine Abbildung von den logischen Operatoren der Interndarstellung über zugeordnete physische Operatoren

auf effizente Ausführungspläne zu finden. Die Schwierigkeit dieser Aufgabe besteht darin, den besten Ausführungsplan unter sehr vielen möglichen Plänen auszuwählen. Um die Komplexität dieses Problems beherrschbar zu machen soll die Anfrageoptimierung weiter folgendermaßen unterteilt werden.

1. Restrukturierung des Operatorbaumes (Anfragegraph)
2. Zuordnung von physischen Operatoren (Planoperatoren) zu logischen Baumoperatoren
3. Auswahl von Methoden zur Realisierung der Planoperatoren

Schritt 1 wird auch *Anfragerestrukturierung* oder *regelbasierte Optimierung* genannt. Dessen Aufgabe ist es, nicht optimierte Anfragegraphen in optimierte Anfragegraphen umzuformen. Dabei werden im wesentlichen Heuristiken angewendet, die versuchen, günstige Operatorreihenfolgen und eine Minimierung der Größe von Zwischenergebnissen einzuhalten. Der auf logischer Ebene optimale Anfragegraph wird in den Schritten 2 und 3 in einen optimalen Anfrageplan überführt. Dies wird auch als *Anfragetransformation* oder *kostenbasierte Optimierung* bezeichnet.

Anfragerestrukturierung Für die logischen Operatoren der Interndarstellung existieren Regeln zur Restrukturierung. Diese Regeln nutzen die beispielsweise Kommutativität, Assoziativität und Distributivität der logischen Operatoren aus und sind semantikerhaltend. Eine Zusammenstellung dieser Regeln finden sich in einschlägigen Büchern zur Anfrageoptimierung, z.B. in [Mit95]. Die Anwendung der Restrukturierungsregeln soll im folgenden Algorithmus zur Anfrageverbesserung demonstriert werden.

1. Bildung von binären Verbunden durch Zerlegung komplexer Verbundoperationen.
2. Aufsplitten Selektionen mit mehreren Prädikatstermen in einfache Selektionen.
3. Verschiebung von Selektionen zu den Blättern des Anfragegraphen, damit diese möglichst früh ausgeführt werden.
4. Zusammenfassung von einfachen Selektionen auf derselben Relation.
5. Verschiebung von Projektionen zu den Blättern des Anfragegraphen, damit diese möglichst früh ausgeführt werden, aber nicht vor die Selektion, da diese den Datenumfang voraussichtlich stärker reduzieren. Dabei Vermeidung teurer Duplikateliminierungen.
6. Zusammenfassung einfacher Projektionen auf derselben Relation.

Die Heuristik die sich hinter diesem Algorithmus verbirgt, versucht die zu verarbeitenden Tupel und Attribute zu vermindern. Deshalb sollen Projektionen und Selektionen möglichst früh ausgeführt werden (Schritt 3 und 5). Schritt 1 und 2 schaffen die nötigen Voraussetzungen, durch die Zerlegung komplexer Verbund- und Selektionsoperationen in binäre Verbunde und einfache Selektionen zerlegt. Durch das Zusammenfassen von Projektionen und Selektionen (Schritt 4 und 6) soll erreicht werden, daß die notwendigen Operationen auf einem Tupel nur einmal durchzuführen sind. Nicht berücksichtigt durch diese Heuristik wurden Idempotenzen sowie gemeinsame Teilausdrücke im Operatorbaum. Weiterhin werden existierende Zugriffspfade nicht beachtet, was das Verschieben von Projektionen oft nicht sinnvoll erscheinen läßt. Die Eliminierung gemeinsamer Teilausdrücke (common subexpression elimination, CSE) sollte sich an den Restrukturierungsalgorithmus anschließen. Ziel der CSE ist es, die mehrfache Bearbeitung gleicher Teilbäume zu verhindern, zum Beispiel Relationen nicht mehrfach einzulesen.

Anfragetransformation Der nächste Schritt der Anfrageoptimierung besteht darin, die logischen Operatoren des Anfragegraphen physischen Operatoren zuzuordnen, bzw. mehrere logische Operatoren zu einem physischen Operator zusammenzufassen. Wie diese Zuordnung erfolgt, hängt teilweise von der Parametrisierung und Implementierung der physischen Operatoren ab. Möglich wäre zum Beispiel, die Zusammenfassung nebeneinanderstehender Selektions- und Projektionsoperationen zu einem physischen Operator `SCAN`, der die Funktionalität dieser beiden Operationen besitzt.

Für die physischen Operatoren existieren im allgemeinen verschiedene Implementationen. Für die Verbundoperation wären beispielsweise die Methoden *Nested-Loop-*, *Merge-* oder *Hash-Join* denkbar. Für jeden physischen Operator muß eine Implementation dieser Operation ausgewählt werden. Dabei soll der resultierende Anfrageplan hinsichtlich des jeweiligen Optimierungszieles der günstigste sein. Dieses Optimierungsproblem wird durch drei Komponenten gekennzeichnet:

- Suchraum
- Suchstrategie
- Kostenabschätzung

Der Suchraum wird durch die Gesamtheit aller möglichen äquivalenten Anfragepläne aufgespannt. Die Suchstrategie bestimmt die Reihenfolge und die Anzahl der zu generierenden und zu besuchenden alternativen Anfragepläne. Die Kostenabschätzung dient der Bewertung der generierten Anfragepläne.

Da es normalerweise sehr viele mögliche äquivalente Anfragepläne gibt, dafür aber nur sehr wenige gute bzw. optimale Pläne, ist es notwendig, den Suchraum systematisch zu durchsuchen. Dazu bieten sich verschiedene Möglichkeiten an.

- *Deterministische Suchverfahren*
  Für deterministische Suchverfahren ist charakteristisch, daß zu jedem Zeitpunkt jeder auszuführende Schritt genau festgelegt ist. Ein deterministischer Suchalgorithmus ermittelt bei gleichem Suchraum und Startpunkt also immer dieselbe Lösung. Zu den deterministischen Suchverfahren rechnet man die *exhaustive Suche*, bei der die Kosten für alle möglichen Anfragepläne mit Hilfe des Kostenmodells abgeschätzt werden und der günstigste Plan in jedem Fall ausgewählt wird. Diese Art der Suche ist aufgrund ihrer Komplexität nur für kleine Suchräume anwendbar. Zu den deterministischen Suchverfahren, die nicht alle Anfragepläne bewerten, zählen beispielsweise:
  - dynamische Programmierung
  - Greedy Algorithmus
  - lokale Suche
  - Branch and Bound
- *Nichtdeterministische Suchverfahren*
  Nichtdeterministische Suchalgorithmen generieren zufällig neue Zustände und bewerten deren Qualität mit Hilfe des Kostenmodells. So kann entschieden werden, ob ein Zustand weitergenutzt oder verworfen wird. Im Gegensatz zu deterministischen Suchverfahren kann bei gleichem Startpunkt und Suchraum trotzdem eine andere Lösung gefunden werden. Zu nichtdeterministischen Suchverfahren zählen:
  - random walk
  - iterative improvement
  - simulated annealing
  - genetische Algorithmen

Eine ausführlichere Klassifikation und Beschreibung von Suchverfahren ist in [Lan97] enthalten. Ziel der Suche ist es, eine möglichst kleine Menge von Plänen zu generieren, die aber den optimalen Anfrageplan enthält. Um den optimalen Plan überhaupt zu erkennen und die Suche rechtzeitig zu beschränken, muß eine Kostenbewertung vorgenommen werden. dazu wird ein Kostenmodell zugrunde gelegt, in welches folgende Faktoren einfließen:

- Speichermodell der betroffenen Relationen (Dateiorganisation und Zugriffspfade)
- Selektivitätsabschätzungen für die Anfrageprädikate
- Zwischenergebnisabschätzungen

- Bearbeitungskosten für die Planoperatoren

Die Kostenbewertung beruht im allgemeinen auf zwei Grundannahmen, die sich in der Selektivitätsabschätzung und Zwischenergebnisabschätzung niederschlagen:

1. Die Attributwerte in einer Relation sind gleichmäßig verteilt.
2. Die Werte verschiedener Attribute sind unabhängig, d.h. die Selektionsprädikate sind voneinander unabhängig.

Diese Annahmen sind oft nicht realistisch und können zur Verfälschung der geschätzten Kosten führen. Abhilfe können verbesserte Statistiken oder die Verwendung von Histogrammen, wie z.B. in [IP95] beschrieben, für die Relationen schaffen.

2.3 Realisierung in konkreten Systemen

Die Anfrageverarbeitung in den Projekten HEaD, MoVi und CROQUE unterscheidet sich von der in den vorherigen Abschnitten beschriebenen Verfahrensweise. Deshalb sollen die Besonderheiten dieser Systeme und die Auswirkung auf die Anfrageverarbeitung in den folgenden Abschnitten auf Basis von [HKLM98] kurz beschrieben werden.

2.3.1 HEaD

Im HEaD-Projekt wurden Konzepte und Mechanismen zur Steuerung der Anfrageverarbeitung auf Basis der Parallelität, Lastverteilung und -balancierung in verteilten Datenbanksystemen (VDBS) untersucht und entwickelt. Das HEaD-VDBS basiert auf dem relationalen Datenbankmodell. Es werden horizontale, vertikale und abgeleited horizontale Fragmentierung unterstützt. Aus Gründen der Effizienz und Verfügbarkeit können Fragmente repliziert werden. Die Basisrelationen und Replikate werden in lokalen DBS (Ingres, Postgres) gespeichert.

Die Anfrageverarbeitung in HEaD is ein mehrstufiger Prozeß. Globale Anfragen bezüglich eines globalen Datenbankschemas werden in SQL gestellt.

Übersetzung: Die SQL-Anfragen werden durch den SQL-Compiler in eine Algebra qualifizierter Relationen (AQUAREL) übersetzt. Ergebnis der Übersetzung ist ein globaler, fragmentierter Algebraausdruck in Form eines Algebrabaumes.

Globale Optimierung/Parallelisierung: Dieser wird durch den Globalen Query Manager (GQM) mit Hilfe von Informationen über Replikate, Relationenstatistiken, Leistungscharakteristika der Rechnerknoten und Lastverteilung in lokal ausführbare Teilanfragen zerlegt (lokale QEP). Optimierungsziel ist die Minimierung der Ausführungszeit für die Teilanfragen unter Nutzung des Parallelisierungspotentials des verteilten Systems.

Lokale Optimierung/Ausführung: Die Bearbeitung von Teilanfragen (lokalen QEP's) auf einem Rechnerknoten wird durch den Lokalen Query Manager (LQM) gesteuert. Dieser kann folgende Optimierungsschritte mit Rücksicht auf die Belastung des Rechnerknotens ausführen:

- Zusammenfassung mehrerer logischer Operationen zu einer physischen Operation (z.B. Selektion, Projektion)
- Zusammenfassung von physischen Operationen zu Ausführungseinheiten (Threads). Mehrere Threads können zu einem Betriebssystemprozess zusammengefaßt werden.
- Teilpläne, die vollständig vom lokalen DBMS abgearbeitet werden können, werden als ein Prozeß ausgeführt.

Operationen die nicht vom lokalen DBMS ausgeführt werden können, werden durch separate Anfrageprozessoren implementiert.

2.3.2 MoVi

Das MoVi-Projekt hat sich zum Ziel gesetzt, beliebige Informationen des Infoverse für einen mobil agierenden Nutzer zu visualisieren. Die Mobilität beinhaltet, neben den Problemen stationärer, verteilter Arbeit, neue Anforderungen, die aus der dynamischen Arbeitsumgebung des Nutzers, dem mobilen Kontext, resultieren. Dieser stellt im wesentlichen den Aufenthaltsort des Nutzers und Informationen über die zur Verfügung stehenden Resourcen (Speicher, Bildschirmparameter, Energie etc.) dar.

Der gesamte Anfrageverarbeitungsprozeß muß sich dem mobilen Kontext anpassen. Die zu verarbeitende Anfrage wird zunächst zerlegt und falls notwendig an Ort und Zeit angepaßt. Die anschließende Optimierung weicht von der klassischen Optimierung ab. Die Parameter der Kostenfunktion können neben Datenbankparametern auch alle Kontextinformationen beinhalten. Die Optimierungsziele unterscheiden sich ebenfalls von den traditionellen Optimierungszielen. So ist eine Antwortzeitoptimierung oft nicht sinnvoll, da Antwortzeiten wegen möglichen Verbindungsunterbrechungen nicht einschätzbar sind. Optimierungsziele, wie die Minimierung der zu übertragenden Datenmenge oder Minimierung des Energieverbrauchs, sind daher von größerer Bedeutung.

Die Anfrage wird entsprechend transformiert und gegebenfalls verteilt abgearbeitet. Zunächst wird versucht, die Anfrage lokal abzuarbeiten. Andernfalls kann die Anfrage so verändert werden, daß die fehlenden Informationen von einem entfernten Server des Infoverse geholt werden. Das Anfrageergebnis wird entsprechend den Anforderungen des aktuellen mobilen Kontexts umgeformt (durch Selektion, Projektion, Sortierung etc.).

2.3.3 CROQUE

Das CROQUE-Projekt befaßt sich mit dem Entwurf und der Realisierung eines objektorientierten Anfrageverarbeitungssystems und Anfrageoptimierers auf Basis von ODMG-OQL[2] und ObjectStore.

Der CROQUE-Optimierer wandelt einen Anfrageplan, der aus einer OQL-Anfrage generiert wurde in einen 'optimalen' physischen Anfrageplan um, der von ObjectStore übersetzt und abgearbeitet werden kann. Der initiale Anfrageplan ist eine Kombination aus einem kalkülartigen Ausdruck und logischer Algebra. Darin werden mittels *pattern matching* Muster identifiziert, die eine effiziente algebraische Umsetzung besitzen und diese durch ihre algebraischen Gegenstücke ersetzt. Ergebnis ist ein Ausdruck der logischen Algebra. Durch zyklische Anwendung logischer Transformationsregeln und Planauswahl werden aus dem logischen Algebraausdruck neue, hoffentlich effizientere, äquivalente Algebraausdrücke erzeugt. Das Rewriting wird beendet, falls keine Transformationsregeln mehr anwendbar sind, eine maximale Anzahl von Zyklen erreicht oder eine Zeitschranke überschritten ist.

Heuristiken, die bei der Regelanwendung verwendet werden sind beispielsweise:

- Bevorzugung von Plänen, die durch große Anzahl von Rewritingschritten entstanden sind
- Bevorzugung von Plänen, die durch Anwendung von Regeln mit hohem Optimierungspotential entstanden sind
- Nur Anwendung der n-besten anwendbaren Regeln auf den ausgewählten Plan (bei definiertem n)

Das Ergebnis des Rewriting ist eine Menge äquivalenter Anfragepläne, die eine erste Dimension des Suchraums aufspannen. Aus diesem Suchraum kann die jeweilige Suchstrategie Pläne auswählen, für die durch den Plangenerator nacheinander alle physischen Ausführungsplane generiert werden. Jeder dieser physischen Pläne wird durch das Kostenmodell bewertet. Die Bewertung dient der Suchstrategie als Entscheidungsbasis für die Auswahl weiterer Pläne aus dem Suchraum. Auf diese Weise kann die Suchstrategie den hoffentlich besten, physischen Ausführungsplan bestimmen.

2.4 Zusammenfassung

Dieses Kapitel sollte eine allgemeine Einführung in die Anfrageverarbeitung geben. Ausgehend von den Phasen der Anfrageverarbeitung, der *Umwandlung in*

[2] ODMG-Object Query Language

eine Interndarstellung, *Anfrageumformung*, *Plangenerierung* und der *Kostenabschätzung*, wurde eine Komponentensicht der Anfrageverarbeitung entwickelt. Die Übersetzungskomponente hat neben der syntaktischen und semantischen Analyse die wichtige Aufgabe, die logischen Transformationen *Standardisierung* und *Vereinfachung* durchzuführen. Dies geschieht vor der Generierung der Interndarstellung (hier Anfragegraph), da dies zu einer vereinfachten Interndarstellung führt. Die Optimierungskomponente wurde in Anfragerestrukturierung und Anfragetransformation unterteilt. Die Anfragerestrukturierung wandelt die standardisierten Anfragegraphen auf der Basis von Heuristiken und Regeln in einen optimierten Anfragegraphen um. In der Anfragetransformationskomponente werden die logischen Operatoren im Anfragegraphen durch physische Operatoren ersetzt. Die Kombinationsmöglichkeiten der verschiedenen physischen Operatoren ermöglichen verschiedene Anfragepläne, die den Suchraum aufspannen. Aus dem Suchraum wählt der Suchalgorithmus Pläne aus, die von der Kostenfunktion bewertet werden. Der kostengünstigste Anfrageplan wird ausgewählt und ausgeführt.

2.5 Bewertung von Optimierungen vs. Bewertung von Optimierern

Nach der Untersuchung von Benchmarks zur Bewertung der Optimierungskomponente von DBS im folgenden Kapitel sollen der Anfrageoptimierer und die Datenbankoptimierung an sich genauer betrachtet werden. In Kapitel 4 werden daher konkrete Möglichkeiten zur Bewertung eines Optimierungsvorganges beschrieben. Dabei wird versucht die Frage zu beantworten, ob das vom Anfrageoptimierer gefundene Optimum tatsächlich das reale Optimum ist. Im darauffolgenden Kapitel, wird der Anfrageoptimierer näher betrachtet. Es werden seine wichtigsten Komponenten benannt und Methoden zur Bewertung der Qualität dieser vorgestellt.

Kapitel 3

Existierende Bewertungsmethoden

Dieses Kapitel umfaßt eine Betrachtung existierender Methoden zur Qualitätsbewertung von Anfrageoptimierern und Optimierungen. Ein interessanter Ansatz wird in [SF95] gezeigt. Darin werden zwei Werkzeuge zum Test der Qualität eines Anfrageoptimierers vorgestellt. Diese erlauben es ein Datenbankschema, die dazugehörigen Daten und eine Menge von Anfragen an diese Datenbank zu spezifizieren und zu generieren. Da sich dieses System noch in der Entwicklungsphase befindet und keine Implementation und Testergebnisse vorliegen, soll dieser Ansatz nicht weitergehend betrachtet werden. Der Schwerpunkt des Kapitels liegt auf der Vorstellung von drei Datenbank-Benchmarks, die besonders den Anfrageoptimierer eines DBS testen. Im Anschluß werden die vorgestellten Benchmarks auf ihre Eignung zur Optimierungs- und Optimiererbewertung untersucht.

3.1 Die Notwendigkeit problemspezifischer Benchmarks

Benchmarks sind wichtige Werkzeuge zur Bewertung der Performance von Computersystemen und -applikationen. Sie besitzen für viele Bereiche der Computerindustrie eine große Bedeutung. Programmierer können Benchmarks nutzen, um sich zwischen verschiedenen Implementationsmöglichkeiten zu entscheiden, Hersteller, um ihre Produkte mit denen der Konkurrenten zu vergleichen und nicht zuletzt Kunden, um das für sie günstigste System auszuwählen. Meist liefert das Resultat eines Benchmark-Testes eine Zahl in einer Metrik wie Preis/Performance oder Arbeit/Sekunde. Es ist unmöglich, die Performance eines Computersystems für alle Applikationen mit einer einzigen Zahl allgemeingültig festzulegen. Viele Systeme sind auf die Lösung einiger weniger Probleme zugeschnitten. Beispielsweise sind Datenbankapplikationen eher selten auf Supercomputern zu finden. Benchmarks, die die Hardwareperformance messen, wie zum Beispiel der SPEC-

Benchmark, besitzen keine große Aussagekraft bezüglich der Leistung eines Datenbanksystems, die mehr von der Effektivität der benutzten Algorithmen, als von der Geschwindigkeit der Hardware abhängt. Dies begründet die Notwendigkeit von Benchmarks für spezielle Applikationen. Jeder dieser Benchmarks spezifiziert eine Arbeitslast, die für die jeweilige Problemklasse als typisch angenommen wird. Die Performance eines Systems für diese Last gibt eine ungefähre Angabe der Leistung für diese Problem.

Im folgenden werden drei Benchmarks zur Messung der Anfrageverarbeitungs-Performance kurz vorgestellt. Dies sind der Wisconsin-, der AS^{3}AP- und der Set Query Benchmark, die ausführlich in [Gra91] beschrieben werden. Vorher werden noch die Richtlinien für einen problemspezifischen Benchmark erläutert.

3.2 Kriterien für Benchmarks

Jim Gray nennt in [Gra91][1] vier Kriterien, die ein problemspezifischer Benchmark erfüllen soll.

1. **Relevanz:** Die Spitzen-Performance und das Preis/Leistungs-Verhältnis des Systems für typische Operationen der speziellen Anwendung muß gemessen werden.

2. **Portabilität:** Der Benchmark sollte auch auf anderen Plattformen und Architekturen einfach implementierbar sein.

3. **Skalierbarkeit:** Er sollte für Systeme jeder Größe anwendbar und es sollte möglich sein, ihn für neue Systemarchitekturen anzupassen.

4. **Einfachheit:** Der Benchmark muß einfach und verständlich sein, um eine breite Anwendung zu finden und glaubwürdig zu sein.

Als Beispiel sei ein einfacher Benchmark gewählt, der die Leistung einer Workstation in MIPS (million instructions per second) mißt. Das einzige Kriterium, daß diese Art der Leistungsangabe erfüllt, ist die Einfachheit. Die Portabilität ist nicht gewährleistet, da beispielsweise RISC und CISC Prozessoren auf dieser Basis nicht miteinander vergleichbar sind. MIPS sind auch nicht skalierbar, weil die Anwendbarkeit auf Multiprozessormaschinen unklar ist (ein 1000-mips Prozessor ist nicht gleichzusetzen mit einem Multiprozessor mit 1000 1-mips-Prozessoren). Der Hauptkritikpunkt ist aber, daß die Einheit MIPS nicht relevant ist. Sie mißt keine reale Arbeit, erfaßt nicht die Performance von Software, sondern sagt nur etwas über die Leistung des Prozessors aus.

[1] Abschnitt 1.3, Seite 5 f.

3.3 Wisconsin Benchmark

Der Wisconsin Benchmark wurde 1983 von Bitton, DeWitt und Turbyfill entwickelt, als noch kein Standard-Benchmark für DBS existierte. Die Anfragen des Benchmarks sollten die Leistung der Hauptkomponenten eines relationalen DBS testen und der Aufbau der zugrundeliegenden Relationen sollte einfach verständlich und erweiterbar sein. Der Benchmark wurde bald zum Standard, was die Entwickler aber weniger seiner technischen Qualität zuschrieben. Der Hauptgrund dafür war vielmehr, daß die ersten Bewertungsberichte Zahlen über reale DB-Produkte enthielten, die auch mit Namen benannt wurden. Dies führte zu einer Reihe von "Benchmark-Kriegen" zwischen den Datenbankanbietern, bei denen der jeweils unterlegene Anbieter mit jeder neuen Version seines Produktes neue Zahlen veröffentlichte, die natürlich besser als die der Konkurrenten waren. Auch wenn die Datenbankhersteller viel Zeit und Geld für diese "Benchmark-Kriege" verschwendeten, führte es jedoch auch dazu, daß die teilweise gravierenden Leistungsunterschiede zwischen verschiedenen DBS verschwanden.

3.3.1 Überblick über den Benchmark

Der Wisconsin Benchmark war einer der ersten wichtigen Benchmarks, die mit einer synthetischen Datenbank arbeiteten. Im Gegensatz zu synthetischen Datenbanken sind empirische Datenbanken schwer zu skalieren, das bedeutet, daß die Größe der Datenbank nicht ohne weiteres zu verändern ist. Ein weiterer Nachteil von empirischen Daten ist, daß es beinahe unmöglich ist, Anfragen mit bestimmten Selektivitätsfaktoren oder einer bestimmten Anzahl an Ergebnistupeln zu stellen. Wie dem Wisconsin Benchmark liegen auch den anderen, in diesem Kapitel vorgestellten Benchmarks, dem AS^{3}AP- und dem Set Query Benchmark, synthetische Datenbanken zugrunde.

Die Relationen Die Datenbasis für den Original-Benchmark besteht aus drei Relationen, eine mit 1000 Tupeln (ONEKTUP) und 2 weitere mit je 10000 Tupeln (TENKTUP1 und TENKTUP2). Jede dieser Relationen besteht aus dreizehn Integerattributen und drei 52-Byte Stringattributen. Mit den schnellen Fortschritten in der Computertechnik zeigte sich jedoch, daß die Relationengrößen für die immer schneller werdenden Rechner zu klein gewählt war. Deshalb wurde die Größe der Relationen nicht auf 1000 bzw. 10000 beschränkt. Ein Beispiel für eine skalierbare Benchmark-Relation und eine Übersicht über die verwendeten Attribute gibt die folgende Tabelle 3.1. MAXTUPLES gibt hierbei die gewünschte Relationengröße an.

Die Anfragen Das Spektrum der Anfragen an die Benchmark-Datenbank wurde so gewählt, daß die Leistung aller relationalen Operatoren ermittelt wird. Dies sind:

Attributname	Wertebereich	Ordnung	Kommentar
unique1	0-(MAXTUPLES-1)	zufällig	unique
unique2	0-(MAXTUPLES-1)	sequentiell	unique
two	0-1	zufällig	(unique1 mod 2)
four	0-3	zufällig	(unique1 mod 4)
ten	0-9	zufällig	(unique1 mod 10)
twenty	0-19	zufällig	(unique1 mod 20)
onePercent	0-99	zufällig	(unique1 mod 100)
tenPercent	0-9	zufällig	(unique1 mod 10)
twentyPercent	0-4	zufällig	(unique1 mod 5)
fiftyPercent	0-1	zufällig	(unique1 mod 2)
unique3	0-(MAXTUPLES-1)	zufällig	unique1
evenOnePercent	0,2,4,...,198	zufällig	(onePercent * 2)
oddOnePercent	1,3,5,...,199	zufällig	(onepercent * 2)+1
stringu1	-	zufällig	Kandidatschlüssel
stringu2	-	zufällig	Kandidatschlüssel
string4	-	zyklisch	

Tabelle 3.1: Skalierbare Relation des Wisconsin Benchmark

1. Selektionen mit verschiedenen Selektivitätsfaktoren
2. Projektionen mit verschiedenen Duplikatanteilen
3. Einfach- und Mehrwegeverbunde
4. einfache Aggregate und Aggregatfunktionen
5. Einfüge-, Änderungs- und Löschoperationen

Für die meisten Anfragen enthält der Benchmark zwei Variationen: eine, die von einem geclusterten Primärindex auf dem Attribut `unique2` Gebrauch macht und eine weitere, die einen ungeclusterten Sekundärindex auf dem Attribut `unique1` benutzt. Für jede der 32 Anfragen des Wisconsin Benchmarks wird die Antwortzeit aus dem Durchschnitt der verstrichenen Zeit für mehrere "äquivalente" Anfragen berechnet. Diese Anfragen unterscheiden sich hauptsächlich durch die verwendeten Relationen und Prädikate. Normalerweise laufen für die Selektions-, Änderungs- und Löschtests zehn und für die Verbund-, Aggregat- und Projektionstests vier Anfragen.
Anfrage 1 des Wisconsin Benchmarks ist ein Selektionstest:

```
SELECT * FROM TENKTUP1 WHERE unique2 BETWEEN 0 AND 99
```

Die nächsten drei der neun Varianten der Anfrage könnten zum Beispiel lauten:

```
... FROM TENKTUP2 WHERE TENKTUP2.unique2 BETWEEN 1000 AND 1099
```

```
... FROM TENKTUP1 WHERE TENKTUP1.unique2 BETWEEN 200 AND 299
... FROM TENKTUP2 WHERE TENKTUP2.unique2 BETWEEN 120 AND 219
```

Die Metrik zur Leistungsbewertung eines DBS ist die verstrichene Zeit nach Abarbeitung aller 32 Tests.

3.3.2 Stärken und Schwächen

Obwohl der Wisconsin Benchmark heute nicht mehr viel Aufmerksamkeit erhält, wird er trotzdem noch von einer Anzahl von Datenbankherstellern als Teil der Standard-Qualitätstest verwendet, half er doch Leistungsanomalien der frühen DBS aufzudecken. Trotzdem gibt es einige gravierende Kritikpunkte. Der Wisconsin Benchmark ist ein Einzelnutzer-Benchmark, das heißt die Mehrnutzer-Performance, die für heutige DBS ausschlaggebend ist, wird nicht betrachtet. Weitere Defizite sind die Einfachheit der Aggregat- und Joinanfragen, Fehlen von Outer-Joinanfragen, Massen-Updates und Tests zum Laden und Löschen der Datenbank und die ausschließliche Verwendung von gleichverteilten Attributwerten.

3.4 AS³AP Benchmark

Der AS³AP (ANSI SQL Standard Scalable and Portable) Benchmark wurde mit dem Ziel entwickelt, einen allgemeinen Benchmark zu entwickeln, der es ermöglicht, Systeme mit unterschiedlicher Architektur und Leistungsfähigkeit zu vergleichen. Folgende Anforderung standen im Mittelpunkt des Designs:

- eine umfassende, aber trotzdem leicht kontrollierbare Testreihe zur Messung der Leistung einer Datenbank zu schaffen
- Portabilität und Skalierbarkeit zu gewährleisten, um auf möglichst vielen Plattformen lauffähig zu sein
- minimaler Aufwand, den Benchmark zu implementieren und anzuwenden
- Bereitstellen einer einheitlichen Metrik, des *äquivalenten Datenbankverhältnises*, zur eindeutigen Interpretation der Benchmark-Ergebnisse

Für ein bestimmtes DBS ermittelt der AS³AP Benchmark die *äquivalente Datenbankgröße*. Dies ist die maximale Größe der Datenbank, für die das DBMS die Tests des Benchmarks in weniger als 12 Stunden absolvieren kann. Auf Basis der äquivalenten Datenbankgröße läßt sich das Preis/Leistungs-Verhältnis eines DBMS bestimmen, indem man den Gesamtpreis des DBMS durch die äquivalente Datenbankgröße dividiert. Das *äquivalente Datenbankverhältnis* für zwei Systeme ist dann das Verhältnis ihrer äquivalenten Datenbankgrößen.

Die einzige Anforderung an die AS^{3}AP Testdatenbank ist, daß ihre logische Größe mindestens gleich der Hauptspeichergröße des Testrechners ist. Die logische Datenbankgröße berechnet sich wie folgt:

(Tupelanzahl pro Relation) * (100 Bytes/Tupel) * (4 Datenbankrelationen)

Diese Bedingung verhindert, daß Systeme mit hohen Kosten für die Abspeicherung der Relationen bei der Berechnung des äquivalenten Datenbankverhältnisses bevorzugt werden.

3.4.1 Die AS^{3}AP Datenbank

Die AS^{3}AP Datenbank besteht aus fünf Relationen. Eine davon, die Relation *tiny*, dient lediglich der Erfassung des Overhead. Die anderen vier werden in der gewünschten Größe, mit den folgenden Eigenschaften, generiert.

1. *uniques:* Relation, in der alle Attributwerte voneinander verschieden sind.

2. *hundred:* Relation, in der die meisten Attribute genau 100 voneinander verschiedene Werte haben und korrelieren. Dies ist notwendig, um absolute Selektivitäten von 100 und Projektionen, die exakt 100 Tupel liefern, zu erreichen.

3. *tenpct:* In dieser Relation haben die meisten Attribute eine Selektiviät von 10 Prozent.

4. *updates* Diese Relation ist speziell für Updates gedacht. Verschiedene Verteilungen und Indizes werden benutzt.

Jede der vier Relationen besitzt zehn Attribute mit denselben Namen und Datentypen und ein zusätzliches Attribut *fill*, um gegebenenfalls die Tupelgröße auf 100 Bytes aufzustocken. Die Typen der Attribute sind so gewählt, daß die am häufigsten verwendeten Datentypen, wie Integer- und Gleitkommazahlen, Strings mit fester und variabler Länge und weitere, abgedeckt sind. Den Aufbau der AS^{3}AP Relationen zeigt Tabelle 3.2.

Der Wertebereich eines Attributs wird durch den Typ und die Größe der Relation festgelegt. Für einige numerische Attribute existieren zwei Typen von Wertebereichen, *dense* und *sparse*. Bei einem *dense* Wertebereich, tritt jeder Integer-Wert genau einmal in der Relation auf. Bei einer Relationengröße von 10000, hat das Attribut *key* den Wertebereich von 0 bis 10000. Dabei wird bei jedem Attribut mit *dense*-Wertebereich der Wert 1 weggelassen um, *not-found-scans* zu ermöglichen. Mit diesen Scans kann die Geschwindigkeit zum Lesen und Vergleichen der Daten ermittelt werden.

Auf den Attributen können Zugriffsstrukturen erstellt werden. Der Benchmark erlaubt hier:

Attributname	Typ	Länge
key	integer	4
int	unsigned integer	4
signed	signed integer	4
float	floating point	4
double	double precision	8
decimal	exact decimal	18
date	datetime	8
code	alphanumeric	10
name	string	20
address	variable string	durchschnittlich 20
fill	string	nach Bedarf

Tabelle 3.2: Relation des AS³AP Benchmarks

- B/B*-Bäume
- Hashing

Auf dem Attribut *key* liegt typischerweise ein B*-Baumindex, auf den anderen Attributen Hash- oder B-Baumindizes.

3.4.2 Die Tests

Der Benchmark ist in zwei Module unterteilt, die Einzelnutzer- und Mehrnutzertests. Da manche DBMS keinen Mehrnutzerbetrieb ermöglichen, absolvieren diese nur den Einzelnutzertest. Das *äquivalente Datenbankverhältnis* wird dann nur auf dieser Basis berechnet.

Der Einzelnutzertest

Der Einzelnutzertest besteht aus dem Aufbau der Testdatenbank und aus den Anfragen. Zuerst werden die AS³AP Datenbank von einem externen Speichermedium eingespielt und die Zugriffsstrukturen auf den Relationen angelegt. Anschließend werden die Tabellenstatistiken auf Basis der geladenen Daten aufgebaut und die Relation *updates* auf Band oder ein anderes Medium gesichert.

Die Anfragen des Einzelnutzertests dienen der Untersuchung:

- des Overheads der verschiedenen Ausgabemodi, die eine Anfrage benutzen kann (Bildschirm, Datei, Relation)
- der Zugriffsstrukturen und Anfrageoptimierung anhand von Anfragen mit einfachen Selektionen, Projektionen, Joins und Aggregatfunktionen

- der Effizienz der Änderungsoperationen bei Einzel- und Massen-Updates
- der Integritätsprüfung der Änderungsoperationen durch Einfügen von doppelten Schlüsseln und Ändern von Fremdschlüsseln

Der Mehrnutzertest

Um den Benchmark in Hinsicht auf das Zeitlimit von 12 Stunden und die Metrik der *äquivalenten Datenbankgröße* aussagekräftig zu halten, gilt für die Nutzeranzahl folgende Formel:

$$\text{Nutzeranzahl} = \frac{\text{logische Datenbankgröße in bytes}}{4 \text{ MBytes}}$$

Für eine 40 MByte große Datenbank ist die Anzahl der Nutzer also 10.

Der Mehrnutzertest gliedert sich in vier Tests, die jeweils ein anderes Arbeitslastprofil modellieren. Dies sind folgende online-transaction-processing (OLTP) und information-retrieval (IR) Tests:

1. OLTP-Test, bei dem alle Nutzer ein Eintupelupdate auf einem zufällig gewählten Tupel derselben Relation durchführen.
2. IR-Test, wobei alle Nutzer eine Eintupelanfrage auf derselben Relation ausführen.
3. Gemischter IR-Test, wobei ein Nutzer einen Mix von 10 Updates und Anfragen ausführt und die anderen Nutzer dieselben Anfragen stellen wie in Test 2.
4. Gemischter OLTP-Test, wobei ein Nutzer einen Mix von 10 Updates und Anfragen ausführt und alle anderen Nutzer dieselben Updates machen wie in Test 1.

Die ersten beiden Tests messen den Durchsatz in Abhängigkeit von der Nutzeranzahl und der dritte das Absinken der Antwortzeiten für einen Querschnitt von Anfragen durch hohe Systemlast.

3.4.3 Stärken und Schwächen

Die Relationen des AS^{3}AP Benchmarks sind sehr gut skalierbar, d.h. es ist einfach, Relationen beliebiger Größe zu schaffen. Der Benchmark ist auf Grund der Verwendung des ANSI SQL Standards für den Datenbankaufbau und die Testanfragen einfach auf verschiedene Systeme zu portieren. Es werden im Gegensatz zum Wisconsin Benchmark Mehrnutzertests durchgeführt. Die Leistung eines DBS wird in einer leicht verständlichen und leicht vergleichbaren Maßeinheit, der *äquivalenten Datenbankgröße*, angegeben. Ein Manko des Benchmarks

ist, daß nicht alle in der Praxis anzutreffenden Typen von Anfragen, wie zum Beispiel Mehrwegejoins mit mehr als vier Relationen, abgedeckt werden.

3.5 Set Query Benchmark

Das Design des Set Query Benchmarks ist auf die Leistungsbewertung von Systemen ausgerichtet, die den strategischen Wert der Datenbestände von kommerziellen Unternehmen erforschen sollen. Dies sind zum Beispiel Data-Mining- oder Data-Warehouse-Applikationen, die Fragen wie: „Wer ist unser bester Kunde?“ oder „Wie ist der Verkaufstrend für Produkt X?“ beantworten sollen. Die Datenbankfunktionen, die hinter Anwendungen dieser Art, im folgenden SDA-Applikationen (*strategic data access*) genannt, stehen, sind *set queries*, Anfragen mit Mehrwegejoins mit mehr als 4 Relationen und Bereichsanfragen. Die Anfragen des Set Query Benchmark bilden einen Querschnitt aus den drei wichtigsten SDA-Anwendungen: Dokumentensuche, Direktmarketing und Entscheidungsunterstützung. Vier Schlüsselcharakteristika bestimmen den Benchmark.

1. *Portabilität:* Die Anfragen des Benchmark sind in SQL formuliert und die Relationen werden von einem einfachen Algorithmus generiert.

2. *Funktionale Vollständigkeit:* Die Menge der Anfragen ist für die wichtigsten SDA-Applikationen repräsentativ.

3. *Abdeckung der Selektivität:* In jedem Anfragetyp wird ein großes Spektrum von Selektivitätsfaktoren verwendet.

4. *Skalierbarkeit:* Die Datenbank besteht aus einer Relation, die eine Million Tupel oder, wenn benötigt, ein Vielfaches davon enthält.

3.5.1 Die Benchmark-Relation

Die einzige Tabelle des Set Query Benchmarks BENCH hat 13 indizierte Attribute, die bei der normalen Tupelanzahl von 1 Million `KSEQ, K500K, K250K, K100K, K40K, K10K, K1K, K100, K25, K10, K5, K4` und `K2` genannt werden. Zusätzlich zu den indizierten Attributen gibt es noch mehrere Attribute vom Typ String `S1`,.., `S8`, um die Tupelgröße auf 200 Bytes aufzufüllen. Bis auf `KSEQ` und `S1,..,S8` sind alle Attribute gleichförmig zufällig verteilt. Die Werte von `S1,..,S8` sind so gewählt, daß die Tupel nicht unter 200 Bytes gepackt werden können, falls das DBS Datenkompression unterstützt. Einen Überblick über den Aufbau der Relation gibt Tabelle 3.3. Falls eine größere Relation BENCH gefordert ist, werden die beiden Attribute mit den höchsten Kardinalitäten, `K500K` und `K250K` umbenannt. Bei einer Relation mit 5 Millionen Tupel würden diese

Attributname	Typ	Wertebereich	Verteilung
KSEQ	integer	1..1000000	sequentiell
K500K	integer	1..500000	gleichförmig zufällig
K250K	integer	1..250000	gleichförmig zufällig
K100K	integer	1..100000	gleichförmig zufällig
K40K	integer	1..40000	gleichförmig zufällig
K10k	integer	1..10000	gleichförmig zufällig
K1K	integer	1..1000	gleichförmig zufällig
K100	integer	1..100	gleichförmig zufällig
K25	integer	1..25	gleichförmig zufällig
K10	integer	1..10	gleichförmig zufällig
K5	integer	1..5	gleichförmig zufällig
K4	integer	1..4	gleichförmig zufällig
K2	integer	1..2	gleichförmig zufällig
S1..S8	string	-	-

Tabelle 3.3: Relation des Set Query Benchmarks

beiden Attribute dann `K2500K` und `K1250K` heißen, die anderen aber unverändert bleiben. `KSEQ` hätte dann die Werte 1,2,...,5000000. Die neuen Attributnamen würden die normalen Attributnamen in den Testanfragen ersetzen.

3.5.2 Die Anfragen

Die Anfragen des Benchmarks sollen eine repräsentative Auswahl der in SDA-Anwendungen auftretenden Anfragen darstellen. Als Anwendungen kommen Dokumentensuche, Direktmarketing und Entscheidungsunterstützung in Frage. Die Metrik für die Angabe der Performance eines DBS ist $/query/minute. Zusätzlich lassen sich die Anfragen bei der Berechnung der Performance wichten.

Dokumentensuche In dieser Art von Anwendung spezifiziert der Nutzer das gewünschte Dokument, läßt sich die Anzahl der gefundenen Dokumente via `COUNT` anzeigen und fügt bei Bedarf neue Einschränkungen hinzu. Auffallend dabei ist die häufige Verwendung der `COUNT`-Aggregatfunktion zur Ergebniseinschränkung bis zu dem Punkt, an dem die gewünschten Tupel selektiert werden. Aus diesem Grund bestehen die Testanfragen aus einer Reihe von Anfragen mit der `COUNT`-Funktion mit immer stärkeren Einschränkungen und einer anschließenden Selektion mit mehreren Bedingungen.

Direktmarketing Das Ziel bei Direktmarketing-Applikationen besteht darin, die Zielgruppe für ein Produkt zu ermitteln. Diese Aufgabe besteht aus zwei Phasen.

1. Finden der Selektionskriterien für die Zielgruppe

2. Auswahl und Beschaffung der Daten der Zielgruppe

Eine typische Anfrage der Phase 1 wäre, alle Haushalte aus einer Menge von Postleitzahlen, mit einem Jahreseinkommen über 100000 DM und einem Auto eines bestimmten Herstellers/Jahres für eine Briefwerbeaktion zu zählen. Falls der ermittelte Wert unter oder über der gewünschten Anzahl an Haushalten liegt, muß eine neue Anfrage spezifiziert werden. Wenn die Zielgruppe eingegrenzt wurde, können die Daten der Haushalte in Phase 2 aus der Datenbank selektiert werden. Die Testanfragen teilen sich in Anfragen mit Verwendung der `COUNT`-Aggregatfunktion für Phase 1 und Selektionsanfragen mit mehreren Selektionsbedingungen für Phase 2.

Entscheidungsunterstützung Diese Klasse von Anwendungen soll bei strategischen Entscheidungen helfen. Ein mögliches Szenario wäre, das Kaufverhalten der Kunden zu speichern und mögliche Schlüsse für zukünftiges Kaufverhalten zu ziehen. Wenn man zum Beispiel beobachtet, daß Kunden, die Reisegepäck kaufen, oft kurz danach Mäntel einkaufen, könnte man Kunden, die gerade Gepäck gekauft haben in eine Werbeaktion für Mäntel einschließen. Um derartige Korrelationen zu entdecken, enthält der Benchmark `GROUP BY`- Anfragen, die für zwei Produkte die Verkaufszahlen ermitteln.

```
SELECT K10, K25, COUNT(*) FROM BENCH GROUP BY K10, K25;
```

Um die Kunden zu finden, die sich für zwei verschiedene Produkte interessieren, stellt der Set Query Benchmark Selfjoinanfragen bereit.

```
SELECT COUNT(*) FROM BENCH B1, BENCH B2
WHERE B1.K100K = 50 AND B1.K250K = B2.K500K;
```

3.5.3 Stärken und Schwächen

Ein großer Vorteil des Set Query Benchmark ist die Verwendung von „realistischen Anfragen“, die teilweise wirklich in realen Anwendungen vorzufinden sind. Ein Kritikpunkt ist, daß der Benchmark nur Einzelnutzertests beinhaltet, obwohl die Entwickler behaupten, aus der Einzelnutzerleistung lasse sich auch auf die Mehrnutzerleistung schließen. Weitere Schwächen liegen in der Verwendung nur weniger Attributtypen (integer, string), nur einer Relation und somit im Fehlen von Verbundanfragen über mehrere, verschiedene Tabellen.

Kriterium	Wisconsin BM	AS^{3}AP BM	Set Query BM
Relevanz	$\ominus$	$\odot$	$\odot$
Portabilität	$\odot$	$\odot$	$\odot$
Skalierbarkeit	$\ominus$	$\oplus$	$\odot$
Einfachheit	$\oplus$	$\odot$	$\odot$

Erläuterung der verwendeten Symbole:

$\ominus$ Kriterium schlecht oder überhaupt nicht erfüllt

$\odot$ Kriterium erfüllt

$\oplus$ Kriterium sehr gut erfüllt

Tabelle 3.4: Bewertung der Benchmarks im Überblick

3.6 Zusammenfassung

In diesem Abschnitt werden die vorgestellten Benchmarks gemeinsam hinsichtlich der in Abschnitt 3.2 definierten Kriterien bewertet und die Eignung von Benchmarks im Allgemeinen zur Lösung der in dieser Arbeit adressierten Probleme untersucht.

3.6.1 Tabellarischer Vergleich der Benchmarks

Die Tabelle 3.4 zeigt wie die drei vorgestellten Benchmarks [2] die Kriterien *Relevanz*, *Portabilität*, *Skalierbarkeit* und *Einfachheit* erfüllen. Die folgenden Anmerkungen dienen der Erklärung einzelner Bewertungspunkte. Obwohl der AS^{3}AP Benchmark eine sehr praktikable Metrik zur Bewertung von DBS anbietet, wird er im Kriterium *Relevanz* nicht mit 'sehr gut' bewertet, weil sein Repertoire an Anfragen zu beschränkt ist. Im Gegensatz dazu bietet der Set Query Benchmark eine sehr praxisbezogene Auswahl an Anfragen, jedoch verhindert das Fehlen von Mehrnutzertests eine Bewertung mit 'sehr gut' im Kriterium *Relevanz*. Die Eignung der vergestellten Benchmarks hängt teilweise auch von dem jeweiligen Optimierungsziel ab. Falls eine Antwortzeitoptimierung gefordert wird, ist der Set Query Benchmark gut geeignet. Bei der Durchsatzoptimierung ist dieser Benchmark jedoch nicht anwendbar, da die Mehrnutzertest fehlen. Der AS^{3}AP Benchmark wäre für diesen Anwendungsfall besser geeignet.

Abschließend kann festgestellt werden, daß keiner der vorgestellten Benchmarks alle genannten Kriterien zufriedenstellend erfüllt. Wünschenswert wäre, die Vorteile des AS^{3}AP Benchmark bezüglich den Kriterien *Portabilität* und *Skalierbarkeit*, mit der Anfragevielfalt des Set Query Benchmark zu kombinieren.

[2] In der Tablelle aus Platzgründen mit BM abgekürzt.

3.6.2 Benchmarks zur Optimierungsbewertung

Benchmarks sind nützliche Werkzeuge zur Bewertung von DBMS. Sie reduzieren die Leistung eines DBMS auf eine oder mehrere Zahlen in einer bestimmten Metrik z.B. (Transaktionen/sec), die einen einfachen Vergleich ermöglichen. Für die Bewertung von Optimierungen sind sie aber weniger geeignet. Aus der ermittelten Performance für das DBMS lassen sich weder Aussagen über die Qualität einzelner Optimierungen, noch über die Qualität des Anfrageoptimierers treffen. Dies ist der Tatsache geschuldet, daß Benchmarks ein DBS als Ganzes testen und nicht einzelne Komponenten, wie z.B. den Anfrageoptimierer. Obwohl der Optimierer selbstverständlich das Ergebnis des Benchmarktest beeinflußt, können keine Schlüsse auf dessen Qualität gezogen werden.. Es wäre vorstellbar, daß schlecht optimierte Anfragen auf Grund einer sehr guten Pufferverwaltung oder optimierter Implementierung der physischen Operatoren doch noch sehr gute Testergebnisse liefern. Weiterhin wird statt der Laufzeit für die einzelne Anfrage die Zeit für die Gesamtheit aller Testanfragen betrachtet. Notwendig wäre die Performance der einzelnen Anfrage mit der optimal möglichen Performance zu vergleichen. Ansätze, die versuchen diesen Weg zu gehen, werden im nächsten Kapitel vorgestellt.

Kapitel 4

Methoden zur Optimierungsbewertung

In diesem Kapitel werden drei Möglichkeiten zur Bewertung konkreter Optimierungen für eine fest vorgegebene Anfrage beschrieben, die bisher noch nicht tiefergehende Betrachtung gefunden haben, und ihre praktische Anwendbarkeit bewertet. Für die Bewertung der Optimierung einer konkreten Anfrage bieten sich folgende Ansätze an. Man kann versuchen, die Bewertung sofort nach der Abarbeitung der Anfrage vorzunehmen (online) oder zu einem späteren Zeitpunkt (offline). Falls die Bewertung sofort vorgenommen wird, ist es auf Grund der begrenzten Ressourcen nicht sinnvoll, exhaustiv nach besseren Anfrageplänen zu suchen, da dies dann auch während der Optimierung hätte getan werden können. Stattdessen müssen Abschätzungen für die Qualität der Optimierung getroffen werden.

4.1 Vorstellung der Methoden

Für die im folgenden vorgestellten Bewertungsansätze soll das Optimierungsziel auf die minimale Antwortzeit für eine Anfrage festgelegt sein. Die in Abschnitt 4.1.1 und 4.1.2 vorgestellten Methoden versuchen sofort (online) eine Einordung der Optimierungsqualität vorzunehmen. Die dritte Methode in Abschnitt 4.1.3 nimmt eine exhaustive Bewertung zu einem Zeitpunkt mit niedriger Systemlast (offline) vor.

4.1.1 Fiktive Extrempunkte

Dieser Ansatz kann eine grobe Einordnung der Qualität einer Optimierung sofort nach Ausführung einer Anfrage liefern. Es wird versucht für die Bearbeitungszeit der jeweiligen Anfrage *fiktive Extrempunkte* zu finden. Diese sind einerseits ein Minimum, das von keinem realen System unterboten werden kann und anderer-

seits ein Maximum, das die größtmögliche Bearbeitungszeit festlegt. Die reale Ausführungszeit einer Anfrage wird mit den Extrempunkten verglichen, wodurch eine Einordnung vorgenommen werden kann. In einem heterogenen, verteilten DBS können die maximale und die minimale Ausführungszeit für jede Anfrage durch folgende fiktive Systemkonfigurationen mit Hilfe des zugrundeliegenden Kostenmodells berechnet werden:

1. *Minimum:*
   (a) Verwendung von sovielen Rechnerknoten, wie für die Anfragebearbeitung notwendig sind
   (b) alle Rechnerknoten seien so schnell wie der schnellste Knoten
   (c) Vernachlässigung von Kommunikationskosten
2. *Maximum:*
   (a) nur ein Rechnerknoten für die Bearbeitung der Anfrage
   (b) dieser Knoten sei der langsamste Knoten
   (c) Kommunikationkosten werden nicht vernachlässigt

Die reale Ausführungszeit liegt zwischen diesen Extrempunkten. Wäre dies nicht der Fall, so ließe dies auf eine fehlerhafte Optimierung schließen. Da die Extrempunkte unter Umständen weit auseinander liegen, könnte man sich weitere Extrempunkte suchen, indem man die einzelnen Parameter der fiktiven Systeme variiert. Denkbar wäre ein System mit einem Rechnerknoten, dem schnellsten, ohne Kommunikationskosten. Natürlich bietet dieser Ansatz nur eine sehr vage Einordnung der Optimierungsqualität, weil die Ausführungszeit mit dem fiktiven Optimum und nicht mit dem realen Optimum verglichen wird.

4.1.2 Statistiken aufbauen

Bei dieser Methode werden statistische Informationen für jede Anfrage gesammelt. Dies sind im wesentlichen die Anfrage selbst, oder ihre Interndarstellung nach der Übersetzung, der ermittelte Anfrageplan und die für die Ausführung benötigten Kosten. Die Speicherung dieser Informationen kann entweder in einer internen Datenstruktur oder etwa in einer Datenbankrelation erfolgen. Die letztere Variante hat den Vorteil, daß die Funktionen einer DB-Anfragesprache zur Auswertung genutzt werden können. Weiterhin müssen zu jeder Anfrage die wichtigsten Systemparameter festgehalten werden, um festzustellen, ob die für gleiche Anfragen ermittelten Kosten überhaupt vergleichbar sind. Falls der Systemzustand des VDBS zur Ausführungszeit der Anfragen ähnlich war, kann festgestellt werden, ob die Optimierung der Anfrage das erste Mal besser oder schlechter gewesen ist. Da die Anfragepläne bekannt sind, kann analysiert werden, was die

Gründe einer möglichen Abweichung der Ausführungszeiten gewesen sein könnten. Je häufiger dieselbe Anfrage an das DBS gestellt wird, umso besser lassen sich Schlußfolgerungen bezüglich der Optimierungsqualität ziehen.

4.1.3 Auswertung aller Pläne

Prinzip Diese Methode, die in [ML86a] und [ML86b] zur Bewertung der Systeme R/R^* benutzt wurde, nimmt die Bewertung einer Optimierung nicht sofort nach dem Anfrageoptimierungsprozeß vor, sondern zu einem Zeitpunkt, zu dem das DBS für längere Zeit [1] unbelastet ist. Die typischen (oder alle) Anfragen, die während des Normalbetriebs an das System gestellt werden, werden abgespeichert. Für jede dieser Anfragen werden alle möglichen Anfragepläne ermittelt, mit Kosten bewertet und ausgeführt. Nach der Ausführung jedes Anfrageplans werden die benutzten Systemresourcen ermittelt und daraus die realen Kosten berechnet. Da die die geschätzten und realen Kosten für jeden Anfrageplan bekannt sind, kann man feststellen, ob der Optimierer wirklich das Optimum gefunden hat. Weiterhin lassen sich daraus Schlüsse auf die Qualität des Kostenmodells ziehen. Diese Idee wird in Abschnitt 5.2.1 weiter vertieft.

Der folgende Algorithmus in einer Pseudonotation soll das Prinzip des in diesem Abschnitt beschriebenen Ansatzes verdeutlichen.

Algorithmus

for all plans P for query Q **do**
  force optimizer to plan # P;
  estimate cost $Cost_{est}$ for query Q;
  flush all buffer pages;
  execute query Q;
  get consumed Resources R;
  evaluate real cost $Cost_{real}$ from R;
  store(SysState, Q, P, $Cost_{est}$, $Cost_{real}$);
end for

Empfehlenswert ist, die Pseudoprozedur `store` am Ende des Algorithmus als `INSERT`-Statement in SQL zu implementieren. Dadurch lassen sich komfortabel weitere Analysen durchführen wie z.B.:

1. Ermittlung des geschätzten und realen Optimums.
2. Ermittlung der realen maximalen Ausführungszeit und deren Verhältnis zum realen Optimum.
3. ...

[1] Am besten zu einer Zeit, wenn keine oder nur wenige Nutzeranfragen zu erwarten sind.

Variationen Der in der vorgestellten Methode benutzte Algorithmus, der alle Pläne bewertet und ausführt, benötigt unter Umständen sehr viel Zeit. Deshalb sollen in diesem Abschnitt einige Variationen des Algorithmus erläutert werden.

1. Eine Möglichkeit, die Laufzeit zu verkürzen, ist, aus der Menge aller Anfragepläne Stichproben zu entnehmen. Je nach Größe und Beschaffenheit des Suchraums werden dazu in regelmäßigen Intervallen (z.B. jeder zehnte oder hundertste Plan) oder stochastisch Pläne ausgewählt und dem Bewertungsalgorithmus übergeben. Die Einschränkung auf wenige Pläne reduziert die Dauer der Optimierungsbewertung zwar drastisch, birgt aber das Risiko, nicht den besten oder schlechtesten Anfrageplan zu finden.

2. Eine weiterer Ansatz, die Bewertung aller Pläne zu vermeiden, ist die Parameter des vom Optimierer ermittelten besten Planes im nachhinein zu verändern und den so generierten neuen Plan noch einmal auszuführen.

3. Lohnenswert wäre weiterhin, häufiger auftretende Teilanfragen zu betrachten und zu bewerten, da eine Optimierung einen hohen Nutzen/Kosten-Faktor erwarten läßt.

4. Für eine Einordnung des schlechtesten und besten Anfrageplanes kann man die in Kapitel 4.1.1 beschriebenen fiktiven Extrempunkte heranziehen.

Obwohl die hier genannten Variationen weiterer Untersuchungen bedürfen, soll darauf im Weiteren nicht mehr eingegangen werden, da dies den Rahmen der Studienarbeit sprengen würde. Die vorgestellten Ansätze müssten implementiert werden, um auswertbare Ergebnisse zu erhalten, auf dessen Basis diese Ansätze vertieft werden können.

4.2 Bewertung der Methoden

In diesem Abschnitt werden die bisher vorgestellten Methoden zur Bewertung von Optimierungen kurz anhand ihrer Praktikabilität bewertet. Als Bewertungskriterien für die Optimierungsbewertung bieten sich der Zeitpunkt und die Genauigkeit der Bewertung und der Implementationsaufwand an.

4.2.1 Fiktive Extrempunkte

Der Vorteil dieser Methode ist, daß die Bewertung der Optimierung sofort nach dem Optimierungsvorgang erfolgen kann, weil die Berechnung der fiktiven Extrempunkte für die jeweilige Anfrage vernachlässigbar wenig Zeit in Anspruch nimmt. Der Implementationsaufwand ist als sehr gering anzusehen, da der Optimierer für jeden Extrempunkt nur um eine abgewandelte Kostenfunktion mit veränderten Parametern erweitert werden muß.

Die Tatsache, daß die Extrempunkte nur eine obere und eine untere Grenze für die Optimierungskosten festlegen, ermöglicht nur eine sehr grobe Einordnung der Qualität der Anfrageoptimierung. Es wären praktische Untersuchungen notwendig, um herauszufinden, ob weitere Extrempunkte, wie in Abschnitt 4.1.1 beschrieben, die Genauigkeit der Bewertung verbessern.

4.2.2 Statistiken aufbauen

Der Statistikansatz versucht, die Bewertung ebenfalls sofort nach dem Optimierungsvorgang durchzuführen. Dies ist aber überhaupt erst möglich, nachdem die Anfrage ein zweites Mal gestellt wurde, damit ein Vergleich erfolgen kann. Ein weiteres Problem stellt die Vergleichbarkeit der Optimierungen bei unterschiedlichen Systemzuständen dar. Falls die Systemlast bei gleichen Anfragen ähnlich ist, könnte diese Methode jedoch ebenso schnell, aber bessere Ergebnisse als der Extrempunktansatz liefern. Der Implementationsaufwand dieser Methode ist also relativ hoch einzuschätzen. Es müssen Statistiken etabliert und Anfrage gespeichert und verglichen werden würden.

4.2.3 Auswertung aller Pläne

Die Auswertung aller Anfragepläne stellt die genaueste der drei vorgestellten Methoden zur Optimierungsbewertung dar, da hierbei der reale optimale Anfrageplan ermittelt wird.

Ein Nachteil des Ansatzes ist, daß eine Bewertung einer Optimierung direkt nach dem Optimierungsprozess nicht möglich ist. Das heißt, eine sofortige Adaption des Optimierungsprozesses ist nicht realisierbar, weil die Optimierungsbewertung erst viel später vorgenommen wird. Die Ausführung aller Pläne einer Anfrage könnte mitunter Tage oder Wochen dauern. Deshalb scheint es lohnenswert, die in Abschnitt 4.1.3 beschriebenen Variationen näher zu untersuchen.

Ein weiterer Schwachpunkt der Methode ist die Vernachlässigung des Systemzustandes bei der Optimierung einer Anfrage. Falls der Anfrageoptimierer die aktuelle Systemlast bei der Optimierung berücksichtigt, besteht die Möglichkeit, daß in der Bewertungsphase, während das DBS unbelastet ist, ein anderer optimaler Plan gefunden wird, als während des Normalbetriebs. Dieser kann aber durchaus der beste Anfrageplan für die aktuelle Lastsituation gewesen sein. Um dieses Problem zu beheben, wäre vorstellbar, die aktuelle Systemlast bei jeder Optimierung einer Anfrage abzuspeichern und dieselbe Last während der Optimierungsbewertung zu simulieren wie es im HEaD-Prototypen realisiert ist. Da dabei aber Resourcen verschenkt würden, wäre es für diese Anwendung sinnvoller die Lastsimulation über veränderte Parameter im Kostenmodell zu bewerkstelligen.

Kapitel 5

Bewertung von Anfrageoptimierern

Dieses Kapitel gibt eine kurze Beschreibung des Aufbaus eines klassischen Anfrageoptimierers und der Funktion seiner Komponenten. Der Einfluß der Komponenten auf die Güte der Anfrageoptimierung wird verdeutlicht und Ansätze zur Untersuchung ihrer Qualität werden vorgestellt. Das Kapitel wird mit einer Einschätzung der praktischen Anwendbarkeit dieser Ansätze abgeschlossen.

5.1 Optimiererkomponenten

Nachdem in Kapitel 2 eine allgemeine Einführung in die Anfrageverarbeitung gegeben wurde, soll nun der Aufbau des Anfrageoptimierers genauer betrachtet werden. Wie in Abbildung 5.1 gezeigt, ist dieser in ein **Rewritingmodul** und in ein **Suchmodul** unterteilt. Das Rewritingmodul hat die Aufgabe, die Anfragerestrukturierung, auch algebraische Optimierung genannt, durchzuführen. *Regeln* beschreiben die Umformungsmöglichkeiten für den Anfragegraphen. Die *Regelmengen* legen die Anwendungsreihenfolge für die Regeln fest. Die Nacheinanderanwendung einer oder mehrerer Regelmengen bildet den *Rewriting Plan*. Ergebnis der algebraischen Optimierung ist ein optimierter Anfragegraph. Dieser dient als Ausgangspunkt für die Anfragetransformation, oder auch kostenbasierte Optimierung, die Aufgabe des Suchmoduls ist. Wie auch in Abbildung 5.1 veranschaulicht, lässt sich das Suchmodul sinnvollerweise in zwei Einheiten aufteilen, die ***Plansuche*** und die ***Kostenabschätzung***. Das Suchmodul generiert auf Basis des optimierten Anfragegraphen mögliche Ausführungspläne. Dies kann beispielsweise durch Variation der Joinfolgen und der Methoden für die Planoperatoren oder in verteilten Systemen durch Zuordnung von Operatoren zu verschiedenen Rechnerknoten erfolgen. Die Gesamtheit aller möglichen äquivalenten Anfragepläne spannt den *Suchraum* auf. Aufgabe des *Suchalgorithmus'* ist es, Anfragepläne aus dem Suchraum auszuwählen. Zu jedem dieser Pläne werden

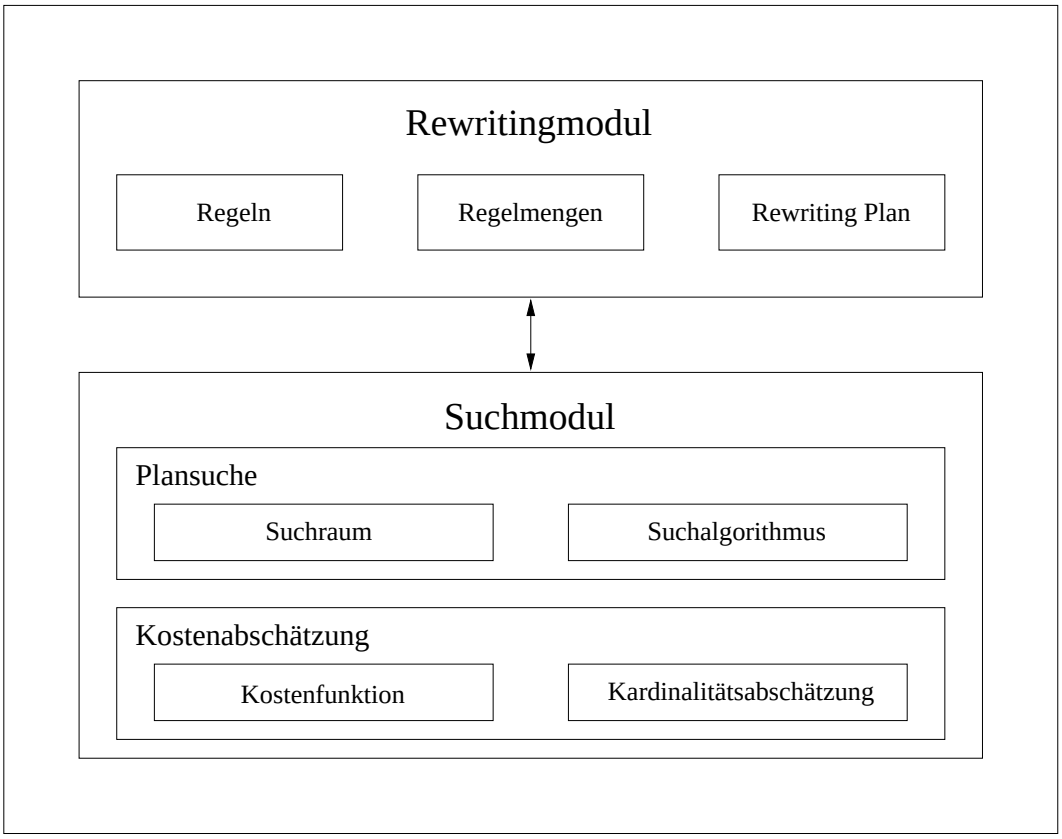

Abbildung 5.1: Komponenten eines Anfrageoptimierers

von der *Kostenfunktion* die Ausführungskosten berechnet. Voraussetzung dafür sind:

- *Selektivitätssabschätzung* für Basisrelationen und *Kardinalitätsabschätzung* für Zwischenergebnisse
- Kenntnis der Bearbeitungskosten der Planoperatoren

Jede einzelne Komponente des Anfrageoptimierers beeinflußt selbstverständlich die Güte der Optimierung einer Anfrage. Falsche Heuristiken im Rewritingmodul, eine schlecht gewählte Suchstrategie oder fehlerhafte Abschätzungen der Zwischenergebnisse während der Kostenberechnung führen mit hoher Wahrscheinlichkeit dazu, daß der erzeugte Anfrageplan nicht der Optimale ist. Deshalb ist es von Interesse, Methoden zu entwickeln, die die Qualität der Bestandteile eines Anfrageoptimierers untersuchen. Für das Suchmodul sollen im folgenden Abschnitt einige Ansätze dazu vorgestellt werden. Das Rewritingmodul wird aus folgenden Gründen nicht betrachtet:

1. Die Umformung des Anfragegraphen basiert auf Heuristiken, Annahmen, die zu einer Verbesserung des Anfragegraphen beitragen sollen.

2. Die algebraische Optimierung ist nicht kostenbewertet, daher können auch nicht ohne weiteres Schlußfolgerungen über deren Qualität getroffen werden.

5.2 Ansätze zur Bewertung der Komponenten

Die in den nächsten Abschnitten vorgestellten Ansätze sollen Schwächen oder Fehler im Suchmodul eines Anfrageoptimierers aufdecken, bzw. dessen korrekte Arbeitsweise bestätigen. Dabei ist zu beachten, daß die einzelnen Komponenten des Suchmoduls in einer bestimmten Reihenfolge zu untersuchen sind, um mögliche Fehlerquellen genau zu lokalisieren.

1. Kostenberechnung
   (a) Kardinalitätsabschätzungen
   (b) Kostenfunktion
2. Suchraum und Suchalgorithmus

Die Kardinalitätsabschätzungen müssen zu Beginn getestet werden, weil diese eine der wichtigsten Parameter für die Kostenberechnung darstellen. Nur wenn die Ergebnisabschätzung innerhalb gewisser Grenzen gute Resultate liefert, können die restlichen Parameter der Kostenfunktion und deren Wichtung untersucht werden. Nachdem festgestellt wurde, daß die Kostenabschätzung verlässliche Ergebnisse liefert, werden Suchraum und Suchstrategie überprüft.

5.2.1 Kostenabschätzung

Gemäß der obigen Vorbetrachtungen werden im folgenden erst Ansätze zur Untersuchung der Kardinalitätsabschätzungen und im Anschluß Möglichkeiten zur Bewertung der Kostenfunktion vorgestellt.

Kardinalitätsabschätzung

Die Kardinalitätsabschätzung dient zur Schätzung der Größe der Ergebnisrelation einer Operation. Dabei unterscheidet man zwischen Operationen auf Basisrelationen, im allgemeinen der Tabellenzugriff, und Operationen auf Zwischenrelationen. Die Schätzung der Ergebnistupelanzahl einer Operation ist eine wesentliche Grundlage für die Kostenberechnung.

Abschätzungen für den Zugriff auf Basisrelationen Für die Abschätzung der Tupelanzahl einer Selektion werden Statistiken zur Hilfe genommen. Im allgemeinen werden sowohl Relationenstatistiken, als auch Attributstatistiken geführt.

Darin werden die wichtigsten Kenngrößen für das jeweilige Speichermodell zusammengefaßt. Natürlich müssen die Statistiken so aktuell wie möglich gehalten werden. Es existiert aber auch ein Zusammenhang zwischen der Genauigkeit einer Statistik und den dazu notwendigen Aktualisierungskosten. Eine direkte Aktualisierung der Statistiken nach jeder Änderung wäre bei sich häufig ändernden Daten viel zu aufwendig. Die Alternative wäre eine Neuberechnung der Statistiken in periodischen Abständen. Auch dies kann im allgemeinen recht teuer werden. Eine deutliche Verbesserung kann erzielt werden, wenn die Statistiken anstatt aus allen Tupeln nur aus einer Stichprobe berechnet werden. Allerdings mußen dabei wieder geringe Ungenauigkeiten in Kauf genommen werden.

Mit Hilfe der Statistiken für Relationen und Attribute können nun Selektivitätsabschätzungen getroffen werden. Die *Selektivität s(p)*[1] eines Prädikats p hinsichtlich eines gegebenen Objektstroms gibt den erwarteten Anteil an Elementen des Objektstroms an, die das Prädikat erfüllen [Mit95]. Die Anzahl c der von Prädikat p ausgewählten Tupel aus Relation N läßt sich durch:

$$c = s(p) * card(N)$$

abschätzen[2] [3]. Diese Abschätzung ist aber nur richtig, wenn die in Abschnitt 2.2.3 genannten Grundannahmen zutreffen. Dies ist einmal die Annahme, daß die Werte eines Attributs einer Gleichverteilung unterliegen und andererseits, daß die Attribute einer Relation voneinander unabhängig sind[4]. Um die Selektivitätsabschätzung für die Basisrelationen zu validieren, müssen diese Annahmen, wie im folgenden beschrieben, überprüft werden.

1. *Gleichverteilung:* Die Verteilung der Werte eines Attributes läßt sich durch mathematische Tests, Verteilungstests, ermitteln. Falls eine Gleichverteilung vorliegt, sollte die Ergebnisabschätzung nach der obigen Formel hinreichend genau sein. Es ist vorstellbar, daß man mit Hilfe der Verteilungstests eine andere Verteilung, z.B. Normalverteilung, nachweisen kann. In diesem Fall muß die Formel zur Berechnung der Selektivität eines Attributes,*s(p)*, für die jeweilige Verteilungsfunktion angepaßt werden. Sollte es nicht möglich, sein eine Verteilung für die Attributwerte zu ermitteln, muß entweder auf die Histogrammtechnik zurückgegriffen oder eine schlechte Kardinalitätsabschätzung in Kauf genommen werden.

2. *Unabhängigkeit:* Die Abhängigkeit zwischen zwei Attribute kann durch eine Regressionsanalyse ermittelt werden. Deren Ergebnis sind zwei Funktionen, die die Abhängigkeit jeweils eines Attributes von dem anderen beschreiben.

[1] Der Wert von *s(p)* liegt dabei im Intervall [0,1]

[2] card(N) gibt dabei die Gesamtanzahl der Tupel in Relation N an

[3] Die Berechnung von *s(p)* für eine Gleichverteilung kann in [Mit95] nachgelesen werden.

[4] Damit sind indirekte Abhängigkeiten gemeint, nicht *functional dependencies*, funktionale Abhängigkeiten.

Der Grad der Abhängigkeit läßt sich durch eine Korrelationsanalyse bestimmen. Nimmt der ermittelte Korrelationsfaktor[5] den Wert -1 oder 1 an, liegt eine proportionale Abhängigkeit zwischen den beiden betrachteten Attributen vor.

Ergebnisabschätzungen für Zwischenrelationen Die für Basisrelationen vorgeschlagene Technik ist nicht auf Zwischen- oder Ergebnisrelationen anwendbar. Der Aufwand zur Bestimmung der Verteilung eines Attributes wäre nicht gerechtfertigt, da während der Optimierung einer Anfrage viele Zwischenrelationen erstellt werden und diese dynamisch sind, d.h. nur einmal benötigt werden. Die folgenden Bewertungsansätze sind daher vergleichender Natur.

- Eine Möglichkeit besteht darin, die geschätzten Kardinalitäten der Zwischenrelationen mit der realen Kardinalität, die nach Anfrageausführung feststeht, zu vergleichen. Die Differenz zwischen geschätzter und realer Tupelanzahl wäre eine Kriterium für die Güte der Abschätzung.

- Für einzelne oder alle Anfragepläne werden die Kardinalitätsabschätzungen verglichen. Aus den Abschätzungen lassen sich Minimum, Maximum und Durchschnitt bestimmen. Bei zu großen Differenzen zwischen minimaler und maximaler Kardinalität kann die Kardinalitätsabschätzung fehlerhaft sein. Nach Anfrageausführung können diese Werte wieder mit der tatsächlichen Ergebnistupelanzahl verglichen werden.

Für beide Ansätze wäre vorstellbar, aus den Soll- und Ist-Werten der Abschätzungen einen Wichtungsfaktor zu bestimmen, der in die nächste Optimierung einbezogen wird. Das bedeutet, daß die Kardinalitätsabschätzung sich zu einem gewissen Grad selbst korrigiert.

Kostenfunktion

Die Kostenfunktion dient der Berechnung der Kosten für einen Anfrageplan. Auf Basis der Kardinalitätsabschätzungen und der bekannten Bearbeitungskosten der Planoperatoren lassen sich die verschiedenen Kostenanteile abschätzen. Dies sind hauptsächlich:

- CPU-Kosten (lokale Verarbeitungskosten: pro Tupel, Tupelblock, Relation)
- Kommunikationskosten (Nachrichten und Datentransfer in VDBS)
- E/A-Kosten (pro Datenbankseite, Datei)
- Speicherkosten (benutzter Hauptspeicher)

[5] Der Korrelationsfaktor nimmt Werte im Intervall [-1,1] an

- Overhead (Prozeßverwaltung, Pufferverwaltung etc.)

Diese Kostenanteile gehen in einer bestimmten Wichtung, die abhängig von den Leistungsparametern des jeweiligen Systems ist, wie folgt in die Kostenfunktion[6] ein:

$$cost = W_{CPU} * (\#instr) + W_{I/O} * (\#io) + W_{msg} * (\#msg) + W_{trans} * (\#bytes)$$

Im folgenden sei angenommen, daß die Kardinalitätsabschätzungen und die Kosten für die Planoperatoren hinreichend genau bekannt sind. Auf Basis dieser Annahme ist dann für die Validierung der Kostenfunktion die Wichtung der Parameter, hier: W_{CPU}, $W_{I/O}$, W_{msg} und W_{trans}, zu untersuchen.

Vorgehensweise Die Kostenberechnung wird für einen repräsentativen Querschnitt von Anfragen untersucht. Für jede dieser Anfragen werden die Schätzwerte der Bearbeitungskosten und der zu benutzenden Resourcen abgespeichert. Während der Ausführung der optimierten Anfragen werden die realen Bearbeitungskosten und die wirklich benutzten Resourcen, zum Beispiel durch einen im Hintergrund laufenden *Load-Daemon*, wie beispielsweise im HEaD-Prototypen, gemessen. Die dadurch gewonnenen Meßdaten lassen sich auf verschiedene Weise auswerten.

1. *Berechnung der Gewichte der Kostenfunktion*: Dazu werden für mehrere, verschiedene Anfragen die realen Kosten gemessen. Da die Abschätzungen für die Kardinalitäten bereits validiert wurden und somit verlässlich sind, lassen sich auf deren Basis die jeweiligen CPU-, E/A- und Kommunikationskosten berechnen. Die Gesamtkosten (*cost*) und die Werte für die verschiedenen Kostenanteile werden dann in die Kostenfunktion eingesetzt. Es ergibt sich ein Gleichungssystem, aus dem sich die Gewichte W_{CPU}, $W_{I/O}$, W_{msg} und W_{trans} berechnen lassen.

2. *Vergleich: geschätzte und benutzte Resourcen:* Für jede Anfrage können die geschätzten und die real benötigten Resourcen miteinander verglichen werden. Falls grobe Unterschiede auftreten, liegt die Vermutung nahe, das entweder die Kardinalitätsabschätzung fehlerhaft ist, oder die Bearbeitungskosten für einen oder mehrere Planoperatoren falsch veranschlagt wurden.

3. *Ausführung aller Pläne zu einer Anfrage:* Diese Möglichkeit der Auswertung hat gewisse Ähnlichkeit mit der in Abschnitt 4.1.3 dargestellten Methode. Zu jedem möglichen Ausführungsplan der betrachteten Anfrage werden die realen Kosten gemessen. Zusammen mit den geschätzten Kosten können diese, wie in Abbildung 5.2, graphisch dargestellt werden. Im linken Teil der Abbildung wird eine gute Kostenabschätzung gezeigt. Die Kurve für

[6] W_{CPU}, $W_{I/O}$, W_{msg} und W_{trans} stellen dabei jeweils die Gewichte für die Kostenanteile CPU-Kosten, E/A-Kosten, Kosten für Nachrichten und Kosten für Datentransfers dar.

die geschätzten Kosten hat ihre Extreme etwa bei denselben Plänen, wie die Kurve der realen Kosten. Ein Suchalgorithmus wäre mit einer derartigen Kostenabschätzung immer noch in der Lage, den optimalen Anfrageplan zu finden. Im Gegensatz dazu steht die Kostenabschätzung im rechten Teil von Abbildung 5.2. Die lokalen Minima und Maxima der Kurven für reale und geschätzte Kosten liegen nicht bei denselben Anfrageplänen. Einem korrekt implementierten Suchalgorithmus wäre es also auf Grund der schlechten Kostenbewertung nicht möglich, einen (sub-)optimalen Ausführungsplan zu finden.

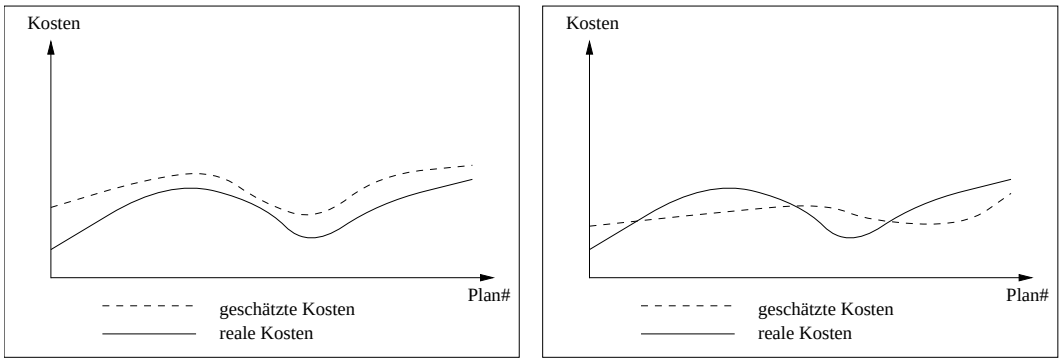

Abbildung 5.2: Beispiel für eine gute und eine schlechte Kostenabschätzung

5.2.2 Suchraum und Suchstrategie

Die Suchstrategie bestimmt, welche Pläne aus dem Suchraum ausgewählt und kostenbewertet werden. Ausgehend von einem Suchraum, hängt die Wahl des Suchverfahrens von folgenden Gegebenheiten ab[7]:

- Größe des Suchraumes
- Verteilung der Lösungen im Suchraum
- jeweiliges Optimierungsziel:
  - Finden des globalen Optimums
  - Finden eines lokalen Optimums
  - Vermeiden des schlechtesten Falles

[7] Die folgenden Ausführungen zur Auswahl einer Suchstrategie sind zum größten Teil [Lan97] S.28 ff. entnommen

Es sei vorrausgesetzt, daß die Suchverfahren korrekt und effizient implementiert seien, um die Vergleichbarkeit der Verfahren zu gewährleisten.

Die Größe des Suchraums und damit die Komplexität des Suchproblems spielt eine große Rolle bei der Entscheidung über die Verwendung eines deterministischen oder nichtdeterministischen Suchverfahrens. Bei kleineren Suchräumen sind deterministische Verfahren ausreichend.

Weiterhin wichtig für die Wahl des Suchverfahrens ist die Verteilung der lokalen Optima. Für sehr große Suchräume mit sehr guten lokalen Optima zwischen vielen Plateaus mit hohen Kosten ist eine lokale Suche angebracht, um schnell ein lokales Optimum zu finden. In einem großen, stark zerklüfteten Suchraum benötigt man einen Suchalgorithmus, der schnell den Suchraum durchwandert, um viele Optima zu untersuchen, beispielsweise ein zufallsbasiertes Verfahren. Letztendlich gibt es aber für jedes Suchverfahren einen Suchraum, für den es schlecht arbeitet oder kein Optimum findet.

Für die endgültige Wahl des Suchverfahrens ist die jeweilige Anwendung entscheidend. Bei vorübersetzten Anfragen, die oft ausgeführt werden, ist es unter Umständen lohnenswert, nach dem globalen Optimum zu suchen, während für interaktive Anfrage die Antwortzeit möglichst gering zu halten ist, wodurch die exhaustive Suche die bei nichttrivialen Suchräumen von vornherein auszuschließen ist.

Die bisherigen Ausführungen zeigen, daß die Wahl eines konkreten Suchverfahrens für einen Suchraum kompliziert ist und nicht vollständig dem Computer überlassen werden kann. Um für eine Anwendung das „optimale“ Suchverfahren zu bestimmen, ist eine Suchraumanalyse notwendig. Dafür müssen die Suchräume für unterschiedlichste Anfragen untersucht werden. Verschiedene Anfrage können unterschiedliche Suchräume aufspannen. Daher muß bei der Auswahl des Suchverfahrens ein Kompromiß gefunden werden.

Vorgehensweise: Die Suchräume von repräsentativen Anfragen einer gegebenen Anwendung werden mit unterschiedlichen Suchverfahren getestet. Das erfolgreichste Verfahren kann als Standardverfahren für die Anwendung ausgewählt werden. Falls sich das Anfrageprofil oder der Datenumfang erheblich verändert, muß der Test wiederholt und gegebenenfalls ein anderes Suchverfahren gewählt werden.

Kapitel 6

Zusammenfassung und Ausblick

Dieses Kapitel befaßt sich mit der Zusammenfassung der Schwerpunkte und der Ergebnisse dieser Arbeit und einem Ausblick auf die weitere Verwendbarkeit dieser Arbeit.

6.1 Zusammenfassung

Die Veranschaulichung der Problematik der Arbeit und den möglichen Nutzen für ausgewählte Forschungsprojekte lieferte die Motivation. Anschließend wurden die Grundlagen der Anfrageoptimierung erklärt. Die Untersuchung des Wisconsin-, AS^3AP- und Set Query Benchmarks auf ihre Eignung zur Bewertung von Anfrageoptimierungen, führte zu einem negativen Ergebnis.

Daraufhin wurde versucht Bewertungsmethoden für Anfrageoptimierungen zu finden. Die drei folgenden Ansätze wurden näher erläutert und auf ihre praktische Anwendbarkeit untersucht.

- Vergleich der Ausführungskosten einer Anfrage mittels fiktiver Extrema
- Führen von Statistiken für häufig auftretende Anfragen
- Exhaustive Untersuchung aller Anfragepläne für eine Anfrage

Auf Basis der Annahme, daß die Qualität der Bestandteile eines Anfrageoptimierers die Güte der Optimierung beeinflußt, wurden dessen Komponenten wie folgt identifiziert und Möglichkeiten zur Untersuchung deren Effizienz gezeigt.

- Kardinalitätsabschätzung
- Kostenfunktion
- Suchraum und Suchalgorithmus

6.2 Ausblick

Die praktische Weiterführung dieser Arbeit ist vorstellbar. Es wäre lohnenswert, einen oder mehrere der in Kapitel 4 und 5 vorgeschlagenen Ansätze testweise zu implementieren und damit deren praktische Anwendbarkeit zu überprüfen. Dabei sollten auch andere Optimierungsziele als die Minimierung der Ausführungszeit betrachtet werden, wie Antwortzeitoptimierung und maximale Resourcenauslastung.

Falls sich eine der Methoden zur Optimierungsbewertung als praktikabel erweisen sollte, ist die Integration dieser Methoden in einen Optimierer als Feedback, mit dem Ziel einer adaptiven Optimierung des Anfrageverarbeitungsprozesses wünschenswert.

Literaturverzeichnis

[ASK80] Astrahan, M.; Schkolnick, M.; Kim, W.: Performance of the System R Access Path Selection Mechanism. *Information Processing*, Jahrgang 1980, S. 487–491, 1980.

[Gra91] Gray, J.: *The Benchmark Handbook for Database and Transaction Processing Systems.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

[HK96] Heuer, A.; Kröger, J.: Query Optimization in the Croque Project. In: *Proc. of the Database and Expert Systems Applications (DEXA'96)*, 1996.

[HKLM98] Heuer, A.; Kröger, J.; Lubinski, A.; Meyer, H.: Anfrageoptimierung in Datenbanksystemen - Ein Überblick ber drei DFG-Projekte. Technischer Bericht, Universität Rostock, Fakultät für Ingenieurwissenschaften, 1998.

[HL96] Heuer, A.; Lubinski, A.: Database Access in Mobile Environments. In: *Proc. of the Database and Expert Systems Applications (DEXA'96)*, 1996.

[IP95] Ioannidis, Y.; Poosal, V.: Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In: *SIGMOD Conference 1995*, 1995.

[Lan97] Langer, U.: *Parallelisierung und Optimierung von Anfrageplänen im heterogenen verteilten, relationalen Datenbanksystem HEaD.* Dissertation, Universität Rostock, 1997.

[Mit95] Mitschang, B.: *Anfrageverarbeitung in Datenbanksystemen.* Vieweg, Braunschweig/Wiesbaden, 1995.

[ML86a] Mackert, L. F.; Lohman, G. M.: R* Optimizer Validation and Performance Evaluation for Local Queries. Technischer Bericht, IBM Almaden Research Center, San Jose, CA, Januar 1986.

[ML86b] Mackert, L. F.; Lohman, G. M.: R* Optimizer Validation and Performance Evaluation for Distributed Queries. Technischer Bericht, IBM Almaden Research Center, San Jose, CA, August 1986.

[SF95] Stillger, M.; Freytag, J.-C.: Testing the Quality of a Query Optimizer. *Data Engineering Bullettin 18(3)*, Jahrgang 1995, S. 41–48, 1995.

Abbildungsverzeichnis

Tabellenverzeichnis