Der vorliegende Artikel *"Anfrageoptimierung in Datenbanksystemen — Ein Überblick über drei DFG–Projekte"* von Andreas Heuer, Joachim Kröger, Astrid Lubinski und Holger Meyer ist 1998 im

Forschungsbericht der Universität Rostock 1994 – 1996 Fakultät für Ingenieurwissenschaften

im Abschnitt

— Ausgewählte wissenschaftliche Beiträge —

auf den Seiten 41–51 erschienen.

Anfrageoptimierung in Datenbanksystemen — Ein Überblick über drei DFG–Projekte

Andreas Heuer, Joachim Kröger, Astrid Lubinski, Holger Meyer

In diesem Beitrag soll ein Überblick über drei DFG–Projekte gegeben werden, die am Lehrstuhl Datenbank– und Informationssysteme der Universität Rostock in den letzten drei Jahren bearbeitet wurden bzw. noch werden. Verbindendes Element der drei Projekte sind Probleme der Anfragebearbeitung und –optmimierung in modernen Datenbanksystemen. Insbesondere handelt es sich um Projekte im Bereich verteilter (relationaler) Datenbanken, im Bereich des "Mobile Computing" und im Bereich objektorientierter Datenbanken.

1 Einführung und Motivation

Die Bearbeitung und insbesondere Optimierung von Anfragen ist bei Datenbanksystemen schon seit Jahrzehnten nicht nur eine zentrale und für den effektiven Einsatz des Systems zentrale Aufgabe, sondern auch ein hartes Problem. Gute Anfrageoptimierer sind diejenigen Komponenten, die selbst in ausgereiften Datenbanksystemen erst in den letzten Versionen integriert wurden.

Der Lehrstuhl Datenbank– und Informationssysteme hat in drei DFG–Projekten den Prozeß der Anfragebearbeitung und –optimierung eingehend untersucht. In diesen Projekten ging es insbesondere um neue Anforderungen, die sich aus dem Datenbankmodell oder der Art der Verteilung der Daten und Systemmodule ergeben.

Im Projekt $\mathsf{HE_AD}$ wird die Anfrageoptimierung in einem verteilten relationalen Datenbanksystem untersucht. Ein Ziel dieses Projektes war insbesondere die Lastverteilung zwischen den verschiedenen beteiligten Rechnern. Das Projekt wurde nach 4 Jahren Laufzeit im Herbst 1996 abgeschlossen.

Im Projekt MoVi wird der Zugriff auf Datenbestände von einem mobilen Rechner aus untersucht. In einer mobilen Umgebung muß der Anfragebearbeitungsprozeß adaptiv und offen gestaltet werden, eine bestehende Anfrage aufgrund von verschiedenen Kontexteinflüssen angepaßt werden. Das MoVi–Projekt wurde Ende 1994 gestartet und läuft noch bis Ende 1998.

Im Projekt CROQUE wird die Anfrageoptimierung für ein objektorientiertes Datenbanksystem untersucht. Objektorientierte Datenbanksysteme zeichnen sich bisher durch navigierende Datenmanipulationssprachen aus, die weder eine deskriptive Beschreibung noch eine systemgesteuerte Optimierung der Anfrage ermöglichen. In diesem Projekt wurde ein regel– und kostenbasierter Optimierer für die Standard–Anfragesprache OQL entwickelt. Das CROQUE–Projekt wurde Ende 1994 gestartet und läuft noch bis Ende 1998.

Alle Projekte basieren auf kommerziellen Plattformen. So sind die in HE$_A$D eingesetzten Datenbanksysteme etwa Ingres und Postgres, die in MoVi verwendeten Informationsserver Ingres und O_2 und die Plattform für CROQUE die persistente C++–Version ObjectStore.

Die einzelnen Projekte werden jetzt in den folgenden drei Kapiteln beschrieben. Für nähere Informationen sei auf die zitierte und am Ende des Beitrags aufgeführte Literatur verwiesen.

2 Anfrageoptimierung in HE$_A$D

2.1 Zielstellung

Die wesentliche wissenschaftliche Zielsetzung des HE$_A$D–Projektes[1] besteht in der Erprobung und Verbesserung von Konzepten und Mechanismen zur Steuerung der Anfrageverarbeitung auf der Basis von Parallelität sowie Lastverteilung und –balancierung in verteilten Datenbankmanagementsystemen, speziell im HE$_A$D–Prototyp.

Die Verbindung der klassischen DB–Optimierung mit der Parallelisierung/Antwortzeitoptimierung und die Erprobung des Systems in normalen Lastsituationen bilden Schwerpunkte, die durch Konzeption, formale Untersuchung und Implementierung abgedeckt werden müssen [11, 21].

Schwerpunkte der Arbeit bildeten folgende Gebiete:

- Lastmessung und –verteilung in einer heterogenen Workstation–Umgebung,
- die Entwicklung eines Kostenmodells für die verteilte Bearbeitung komplexer Anfragen in einem verteilten, relationalen DBMS (Zwischenergebnisrelationsabschätzungen, Attributwertverteilungen, Intra–Transaktionsparallelität und die Pipeline–Ausführung von Operationen),
- konzeptionelle Vereinfachung und Optimierung verteilter, globaler Anfragen auf der Basis der Algebra qualifizierter Relationen (AQUAREL) (Unterstützung von Fragmentierung),
- optimale Parallelisierung verteilter komplexer Anfragen unter Nutzung von Intra–Transaktions–, Inter–Operator– und Pipeline–Parallelität mit dem Ziel der Verkürzung der Antwortzeit einzelner, komplexer Anfragen,
- Optimierung von lokal auszuführenden Teilanfrageplänen, Finden von Teilanfragen, die vollständig vom lokalen DBMS abzuarbeiten sind, und
- Bereitstellung eines Optimierer–Frameworks, in dem mit heuristischer und systematischer (kostenbasierter) Optimierung kombiniert gearbeitet werden kann, sowie Be-

[1] gefördert unter dem DFG–Kennzeichen Me 1346/1–2

reitstellung von Modulen für spezielle Optimierungsschritte (Common subexpression elimination CSE, Join–Folge–Optimierung [15]).

2.2 HE_AD–Systemübersicht

Das verteilte Datenbanksystem HE_AD basiert auf dem relationalen Datenbankmodell. Relationen des globalen Schemas können fragmentiert sein. Vom System werden horizontale, vertikale und abgeleitete horizontale Fragmentierung unterstützt. Aus Verfügbarkeits– und Performance–Gründen werden diese Fragmente repliziert, also über mehrere Rechnerknoten verteilt gespeichert. Die Basisrelationen und Replikate werden in lokalen Datenbanksystemen (CA Ingres, UCB Ingres, Postgres 4.2) verwaltet.

Anfragen an das verteilte DBMS werden in SQL gestellt, intern werden sie in eine erweiterte, relationale Algebra übersetzt. Die physischen Algebraoperationen werden, sofern sie komplett in einem lokalen DBMS abarbeitbar sind, von diesem ausgeführt. Algebraoperationen, die globale Verarbeitungsschritte realisieren, werden außerhalb von den lokalen DBMS in Query–Prozessoren ausgeführt.

Die Kommunikation zwischen den Algebraoperationen erfolgt über ein nachrichtenorientiertes Tupelprotokoll, das lokal auf Shared memory–Implementierungen (MMAP, Memory mapped files) und entfernt auf ein Protokoll (RDP) aus der TCP/IP–Protokollsuite abgebildet wird.

Anfragebearbeitung

Die Anfragebearbeitung im System HE_AD ist ein mehrstufiger Prozeß. Globale Anfragen bezüglich eines globalen verteilten Datenbankschemas werden in der relationalen Anfragesprache SQL gestellt. Die zugrundeliegende Schemaarchitektur im System geht davon aus, daß globale Relationen sowohl horizontal als auch vertikal fragmentiert sein können, und das diese Fragmente wiederum evtl. repliziert vorliegen.

Übersetzung: Im SQL–Compiler werden die SQL–Anfragen in die relationale Algebra AQUAREL übersetzt. Dabei werden die Informationen aus dem Fragmentationsschema (Globales Data Dictionary) in Form von Fragmentausdrücken (Komposition von globalen Relationen aus Fragmenten) und von qualifizierten Relationen als Operanden der Algebraoperationen mit eingebracht. Diese zusätzlichen Informationen können zur logischen Vereinfachung von Algebraausdrücken heranzgezogen werden, z.B. für die Optimierung des verteilten Joins. Ergebnis der Übersetzung ist ein globaler, fragmentierter Algebraausdruck in Form eines Anfragebaumes (globaler QEP, Query Evaluation Plan).

Globale Optimierung/Parallelisierung: Globale Anfragepläne werden durch den Globalen Query Manager (GQM) normalisiert, vereinfacht und unter Zuhilfenahme der Informationen aus dem Replikationsschema, statistischer Informationen zu Relationsgrößen, Attributwerteverteilung, etc., der Beschreibung des Leistungspotentials der Rechnerknoten

und –netzes und der aktuellen Lastbeschreibungsdaten in lokal ausführbare Teilanfragen (lokale QEP) zerlegt.

Optimierungsziel ist die minimale Ausführungszeit der Einzelanfragen unter Nutzung des Parallelisierungspotentials des verteilten Systems. Die Parallelisierung erfolgt unter Nutzung von Intra–Transaktionsparallelität, speziell der horizontalen und vertikalen (Pipeline-) Inter–Operator–Parallelität.

Um verschiedene Optimierungstechniken kombinieren zu können, wird im System ein Optimiererbaukasten benutzt, der unterschiedliche Komponenten in sich vereint [13] (Rewriting–Modul mit unterschiedlichen spezifizierbaren Transformationsregelmengen, Module für spezielle Optimierungsschritte, z.B. Common subexpression elimination (CSE), Kostenmodellspezifikation [2], Lastmessungskomponente, Lösungs–Suchraum–Abbildungen und mehrdimensionale, systematische Optimierung mit entsprechendem Suchverfahren [14]).

Lokale Optimierung/Ausführung: Der Lokale Query Manager (LQM) steuert die Bearbeitung von Teilanfragen (lokale QEP's) auf einem Rechnerknoten. Unter Beachtung der aktuellen Rechnerbelastung ist er in der Lage, noch folgende lokale Optimierungen vorzunehmen:

- Bestimmte aufeinanderfolgende logische Algebraoperationen (z.B. Selektion, Projektion) können zu einem physischen Operator (Filter) zusammengefaßt werden.
- Physische Operatoren werden Ausführungseinheiten (Threads) zugeordnet. Mehrere zusammengehörige/aufeinanderfolgende Threads können zu einem Betriebssystemprozeß zusammengefaßt werden.
- Teilpläne, die vollständig vom lokalen DBMS abgearbeitet werden können, werden als ein Prozeß ausgeführt.

Operationen, die nicht im lokalen DBMS verarbeitet werden können, wie z.B. ein globaler Join zweier Fragmente, die in unterschiedlichen lokalen DBMS gespeichert sind, werden von separaten Anfrageprozessoren (Query Processor, QP) implementiert. Die Kommunikation zwischen den Operatoren/Threads erfolgt über ein spezielles Tupel–Protokoll, das lokal mit Memory mapped files (MMAP) und zwischen Rechnerknoten mit einem sicheren, verbindungsorientierten Netzprotokoll (RDP/IP) implementiert wurde.

Potentiale der Optimierung im verteilten DBMS

Die Shared nothing–Architektur des verteilten Systemes HEAD und die existierenden Workstation–Clusters bieten günstige Voraussetzungen für die Optimierung des verteilten Datenbanksystems. Die Heterogenität der zugrundeliegendenden Hardware erschwert zwar die Optimierung, bietet dafür aber ein großes Optimierungspotential in sich. Die Optimierung findet in zwei getrennten Komponenten statt: algebraische Optimierung und Optimierung paralleler Anfragepläne.

Die algebraische Optimierung weist folgende Potentiale auf:

- Normalisierung und Vereinfachung globaler Anfragen,
- Umformung der globalen Anfrage in fragmentierte Anfragen über Fragmenten einer Relation,
- Entfernung gleicher Teilpläne und nicht benötigter Fragmente (CSE) und
- Bestimmung der optimalen Bearbeitungsreihenfolge der Teiloperationen.

Während die Normalisierung, Fragmentierung und CSE in der Literatur bereits vielfältig analysiert wurden, kommt der in $\mathsf{HE_AD}$ vorgeschlagenen Bestimmung der optimalen Bearbeitungsreihenfolge der Teiloperationen eine besondere Bedeutung bei [13].

Die Optimierung paralleler Anfragepläne muß sowohl eine horizontale als auch eine vertikale Parallelität in Form von Pipelining unterstützen. Die Shared nothing–Architektur erleichert maßgeblich eine horizontale Parallelität. Die Potentiale paralleler Anfragepläne werden genauer in [14] erläutert.

3 Adaptive Anfragebearbeitungsprozesse in mobilen Umgebungen — MoVi

Weltweit sind eine große Menge von multimedialen Informationen elektronisch verfügbar. Wir nennen dies das Informationsuniversum (Infoverse), in dem sich die Nutzer auf Datenautobahnen bewegen. In der Zukunft wird es von zunehmender Bedeutung sein, die heterogenen Strukturen des Infoverse transparent zu machen und andererseits an unterschiedlichen Orten und in verschiedenen Umgebungen das Infoverse zu betreten. Das DFG–Projekt Mobile Visualisierung[2](MoVi) hat sich zum Ziel gesetzt, beliebige Informationen des Infoverse für einen mobil agierenden Nutzer zu visualisieren. Mobilität des Nutzers bedeutet hierbei, daß ein Nutzer bei mobiler Arbeit, die nicht für die Arbeit mit Computern geeignet scheint, unterstützt wird. Dies ist z.B. der Fall, wenn die Wiedereinrichter des Landesforstamtes M–V den aktuellen Waldbestand aufnehmen. Diese Tätigkeiten werden durch sehr kleine, transportable Geräte, PDA's (Personal Digital Assistant) unterstützt. Eine andere Art der mobilen Arbeit mit Computern ist die, daß der mobile arbeitende Mensch unterschiedliche, gerade zur Verfügung stehende Geräte und Kommunikationsnetze nutzt. So könnte mit einem Notebook ein Festnetzanschluß in einem Hotel ebenso genutzt werden wie ein öffentliches Multimedia–Kommunikationsterminal, vergleichbar mit den heutigen Telefonzellen.

Mobilität beinhaltet alle Probleme stationärer, verteilter Arbeit, stellt jedoch darüber hinaus neue Anforderungen [9]. Diese Anforderungen resultieren aus der dynamischen Arbeitsumgebung, der *mobilen Umgebung*. Sie soll **mobiler Kontext** heißen und beinhaltet als

[2]gefördert unter den DFG–Kennzeichen Schu 887/3–1 und Schu 887/3–2

Kern den sich ändernden Aufenthaltsort des Nutzers. Wesentlicher Kontext sind außerdem Informationen über die zur Verfügung stehenden Ressourcen, da mobile Geräte oftmals sehr ressourcenbeschränkt (betreffs Speicher, Bildschirmparametern und Energie) sind bzw. ein Wechsel der Geräte auch einen Wechsel der verfügbaren Ressourcen nach sich zieht. Einen Engpaß stellt außerdem die Bandbreite eines Funknetzes dar. In Relation zu diesen Ressourcen steht die Menge und Größe der Informationen des Infoverse, die auf dem benutzten Gerät visualisiert und bearbeitet werden sollen. So ist ein Video nicht auf PDA's übertragbar, da dieses Medium die Kapazitäten eines PDA übersteigt. Informationen alternativer Medien müssen es ersetzen.

Nicht allein die Informationen, sondern der gesamte Anfragebearbeitungsprozeß muß sich dem mobilen Kontext anpassen indem die jeweils passenden alternativen Komponenten ausgewählt werden [10]. Für die Adaption wurde ein Kontextmanager [20] geschaffen, der die laufenden Anfragebearbeitungsprozesse überwacht und der für die Beschaffung, Aktualisierung und Verteilung der jeweils zutreffenden Kontexte sorgt. Alle Teile des Anfragebearbeitungsprozesses, die Anfragezerlegung, –optimierung, –bearbeitung und –aggregation können von Kontexten beeinflußt werden.

Nach der Anfragegenerierung durch den Nutzer oder aufgrund einer Aktion des Nutzers auf dem Bildschirm wird die Anfrage zerlegt und zunächst, wenn nötig, an den aktuellen Ort und Zeit angepaßt. Eine Informationsanforderung des Wiedereinrichters wird beispielsweise auf das Revier bezogen, in dem er sich aufhält. Die sich anschließende Optimierung weicht von der "normalen" Optimierung in verteilten, stationären Datenbanken ab. Die Parameter, die in die Kostenfunktion einbezogen werden, sind nicht allein Datenbankparameter, sondern können alle Ressourcenkontexte umfassen, und können sehr dynamisch sein. Das Ziel der Optimierung unterscheidet sich ebenfalls von denen der traditionellen Optimierung. Während es in stationärer Umgebung darauf ankommt, die Antwortzeiten zu verkürzen, ist dieses Ziel in mobiler Umgebung unwichtig, da häufige Verbindungsunterbrechungen die Antwortzeit unkalkulierbar machen. Dafür ist die Minimierung der Übertragungskosten über teure Funknetze von größerer Bedeutung. Ebenso kann die Schonung der Energiereserven bei der Anfragebearbeitung bei geringer Akkukapazität eines mobilen Gerätes wichtiger sein als die Erhöhung des Durchsatzes. Auch die Optimierungsziele sind wie die Kostenparameter im Gegensatz zur stationären Verarbeitung dynamisch.

Der Anfragebaum wird entsprechend transformiert und gegebenenfalls verteilt abgearbeitet. Dabei spielen wiederum die geltenden Kontexte eine Rolle, da zunächst versucht wird, die Anfrage lokal zu bearbeiten. Ist dies nicht möglich, kann die Anfrage derart umgeformt werden, daß die fehlenden Informationen von einem entfernten Server aus dem *Infoverse* geholt werden. Nach der Abarbeitung werden die Ergebnisse aggregiert. Dabei wird das Ergebnis so umgeformt, daß es den Anforderungen der aktuellen mobilen Kontexte entspricht. Dies kann eine erneute Selektion oder Projektion auf das Ergebnis oder eine gewichtete Sortierung oder bei stark veränderterm Kontext gar eine völlig neue Anfragebearbeitung zur Folge haben.

So ist der gesamte Anfragebearbeitungsprozeß sehr dynamisch und wird immer wieder neu nach den jeweils geltenden Kontexten gestaltet.

4 Anfrageoptimierung in objektorientierten Datenbanken – CROQUE

Wesentliche Kriterien für den zukünftigen Erfolg oder Mißerfolg objektorientierter Datenbanksysteme werden die *verfügbaren Konzepte* und die *Performance* solcher Systeme sein. Die zur Zeit in kommerziellen Systemen verfügbaren Konzepte vernachlässigen beispielsweise Freiheitsgrade auf der physischen Ebene; zur Leistungssteigerung vorhandener Systeme ist es notwendig, das vorhandene Optimierungspotential aller Komponenten voll auszuschöpfen.

Das DFG–Projekt[3] CROQUE[4] umfaßt den Entwurf und die Realisierung eines objektorientierten Anfragebearbeitungssystems und Anfrageoptimierers auf Basis von ODMG–OQL [1] und ObjectStore [16]. Dazu werden *Plattformen* wie objektorientierte Datenmodelle und deskriptive Anfragesprachen, *Optimierungsansätze in objektorientierten Datenbankmanagementsystemen* (OODBMS), *Optimierungstechniken* und *unterstützende Maßnahmen* wie Monoid–Kalküle, Speicherstrukturen, Suchstrategien sowie Kostenmodelle betrachtet und ihre Integration in einen einheitlichen Ansatz im Rahmen eines Forschungsprototypen untersucht [17].

Im Mittelpunkt der Arbeiten stehen dabei der Entwurf und die Realisierung des Anfrageoptimierers [7, 8]. Die zur Evaluierung notwendigen Laufzeitkomponenten werden soweit als möglich aus verfügbaren Produkten und Prototypen übernommen. Dies schließt insbesondere das Speichersubsystem ein, bei dem das kommerziell verfügbare OODBMS ObjectStore verwendet wird (dies erhöht die Stabilität gegenüber Forschungsprototypen und erleichtert eine spätere Nutzung in Anwendungsprojekten). Weitere Plattformen können Prototypen und eventuell auch modernere SQL–Datenbanksysteme sein. Der Optimierer wurde zunächst als statischer Optimierer konzipiert, d.h. alle Entscheidungen werden zur Übersetzungszeit der Anfrage getroffen. Spätere Erweiterungen hin zu dynamischen Ausführungsplänen, die einige Entscheidungen auf den Ausführungszeitpunkt verschieben, sind vorgesehen, aber nicht Kernbestandteil dieses Vorhabens. Ausgehend von den Optimierungsansätzen aus OSCAR[5] und COCOON[6], die beide auf algebraischen Transformationsregeln basieren, wird derzeit unter Verwendung der funktionalen Sprache SML die zentrale Komponente (der Optimierer) erstellt.

Eingabe für den **Anfrageoptimierer** ist ein Anfrageplan, der mittels eines Parsers aus einer vom Typchecker auf Korrektheit überprüften OQL–Anfrage generiert wurde. Ausgabe des Anfrageoptimierers ist ein ausgewählter ("optimaler") physischer Ausführungsplan, der vom Anfrageprozessor verarbeitet, d.h. in ObjectStore–Quellcode übersetzt und auf ObjectStore ausgewertet werden kann. Der Anfrageoptimierer arbeitet dabei wie folgt: der **initiale Anfrageplan** ist eine **hybride Kombination** aus **kalkül**artigem Ausdruck

[3] gefördert unter den DFG–Kennzeichen Scho 554/1-1 und He 1768/5-1 bzw. Scho 554/1-2 und He 1768/5-2

[4] **C**ost– and **R**ule–based **O**ptimization of object–oriented **QUE**ries — Arbeitsname des Projektes

[5] **O**bject management **S**ystem for **C**omplex **A**pplications, approach: **R**elational [6, 12]

[6] **CO**coon. . . **C**omplex **O**bject **O**rientation based on **N**ested Relations [18, 19]

(*monoid comprehension* – [5]) und **logischer Algebra**, in der mittels *pattern matching* gewisse Muster identifiziert werden, die in der Algebra nicht auf den ersten Blick erkennbar sind, aber eine effiziente algebraische Umsetzung besitzen. Diese Muster werden schrittweise durch ihre algebraischen Gegenstücke ersetzt und der gesamte Ausdruck somit in eine reine Algebradarstellung überführt [3].

Aus dem so erzeugten **Ausdruck der logischen Algebra** und unter Verwendung von **Transformationsregeln** (d.h. gerichteten Rewritingregeln) läßt das Steuerungsmodul vom Plangenerator mittels *Rewriting* zunächst eine Menge äquivalenter Ausdrücke erzeugen. Anschließend wählt wiederum das Steuerungsmodul aus der vorliegenden Menge äquivalenter Pläne und der vorhandenen Regelmenge einen oder mehrere Pläne und eine oder mehrere Regeln aus, die dem Plangenerator als Grundlage für ein erneutes Rewriting dienen. Dieser **Zyklus** von Plan– und Regelauswahl und Plangenerierung mittels **Rewriting** wird mehrfach durchlaufen: zunächst werden im ersten Durchlauf *inverted queries* eingesetzt, d.h. Informationen über die tatsächliche, ggf. redundante, fragmentierte Speicherung werden optimal ausgenutzt und *physische Substitutionen* aufgelöst, d.h. erkannte physisch materialisierte Datenbankvariablen werden durch ihre Definition ersetzt, also z.B. materialisierte Sichten verwendet [4]. Es folgt eine durch eine vordefinierte maximale Schranke begrenzte Anzahl an Durchläufen zum eigentlichen Rewriting, die weitere **äquivalente Ausdrücke** der logischen Algebra erzeugen.

Die **Steuerung** des Rewritings besteht aus verschiedenen Entscheidungen: zunächst wird das Rewriting beendet, sobald eine vorgegebene maximale Anzahl an Zyklen erreicht oder eine maximale Zeitschranke überschritten wurde. Durch diese Schranken wird die *Terminierung* in jedem Falle sichergestellt. Ein weiteres Endkriterium ist erfüllt, sobald keine Regel mehr auf irgendeinen Plan anwendbar ist. Die heuristische Planauswahl verwirft alle diejenigen Kombinationen aus Plan und Regel sofort, die gemäß einer mitgeführten *Hashtabelle* bereits zuvor behandelt wurden.

Verwendete **Heuristiken** [17] sind zum einen die *Bevorzugung von Plänen*, etwa derjenigen Pläne, die durch die Anwendung einer großen *Anzahl von Rewritingschritten* entstanden sind vor solchen, die durch weniger Regelanwendungen entstanden sind; eine Verfeinerung dieser Strategie betrachtet nicht die Anzahl der angewendeten Regeln, sondern deren Qualität (das *Optimierungspotential* der verwendeten Regeln). Diese letztgenannte Bevorzugung erfolgt zum Teil bereits implizit dadurch, daß Regeln mit einem hohen Optimierungspotential generell gegenüber solchen mir einem niedrigeren Potential bevorzugt werden. Zum anderen kann durch eine *selektive Regelanwendung* die Größe des aufgespannten Suchraumes äquivalenter Pläne von vornherein reduziert werden. Dazu werden (bei vordefiniertem n) immer nur die *n–best*en der anwendbaren Regeln tatsächlich auf einen ausgewählten Plan angewendet. Mit Ende dieses Rewriting–Zyklus liegt eine Menge (oder eine Liste von Listen) von äquivalenten logischen Anfrageplänen vor, die die erste Dimension (bzw. die ersten beiden Dimensionen) eines Suchraumes aufspannen. Aus diesem Suchraum können nun **Suchstrategien** (zur Zeit sind vier verschiedene in CROQUE umgesetzt – [17]) Pläne herausgreifen, die mittels des Plangenerators je nach Anforderung nacheinander in alle sie implementierenden physische Ausführungspläne übersetzt werden (dies bildet zugleich die nächste Dimension des Suchraumes, die allerdings nur *selektiv*

aufgespannt wird).

Jeder erzeugte physische Plan kann mittels des CROQUE–**Kostenmodell**s [17] bewertet werden. Diese Bewertung dient der jeweils verwendeten Suchstrategie als Basis für den Vergleich zweier Pläne, der letztendlich wiederum die weitere Suche in aller Regel beeinflußt (ausgenommen ist hier die Suchstrategie Random Walk). Die Suchstrategie kann auf diese Weise den ("optimalen") tatsächlich **auszuführenden physischen Plan** bestimmen.

Literatur

[1] R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93, Release 1.2.* Morgan-Kaufmann, San Mateo, CA, 1996.

[2] G. Flach and H. Meyer. Integration of Load Measurement Parameters into the Cost Evaluation of Database Queries. In *Proc. 14th British National Conf. on Databases (BNCOD)*, Edinburgh, Juli 1996.

[3] T. Grust, J. Kröger, D. Gluche, A. Heuer, M. H. Scholl : Query Evaluation in CROQUE - Calculus and Algebra Coincide - *Accepted for BNCOD'15.* March 1997.

[4] D. Gluche and M. H. Scholl. Physical Design in OODBMS. In *8. Workshop Foundations of Databases.* Preprint Nr. 3, 1996, University of Magdeburg, CS Faculty, May 1996.

[5] T. Grust and M. H. Scholl. Translating OQL into Monoid Comprehensions – Stuck with Nested Loops? Technical Report 03a/1996 (*Konstanzer Schriften in Mathematik und Informatik, Nr. 3a)*, Department of Mathematics and Computer Science, Database Research Group, University of Constance, September 1996. Revised Version.

[6] A. Heuer, J. Fuchs, and U. Wiebking. OSCAR: An object-oriented database system with a nested relational kernel. In *Proc. of the 9th Int. Conf. on Entity-Relationship Approach, Lausanne*, pages 95–110. Elsevier, October 1990.

[7] A. Heuer and J. Kröger. Query Optimization in the CROQUE Project. In: R.R. Wagner and H. Thoma, editors, *Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA '96)*, Zurich, Switzerland, September 1996; volume 1134 of LNCS, pages 489-499, Springer Verlag, 1996

[8] A. Heuer and J. Kröger. Query Optimization in the CROQUE Project (extended version). *Preprint CS–05–96, CS Dept., University of Rostock*, June 1996.

[9] A. Heuer, A. Lubinski. Mobile Information Access, Challenges and Possible Solutions. In: *Proceedings of the IMC'96*, Rostock, Februar 1996.

[10] A. Heuer, A. Lubinski. Database Access in Mobile Environments. In: R.R. Wagner and H. Thoma, editors, *Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA '96)*, Zurich, Switzerland, September 1996; volume 1134 of LNCS, pages 544-553, Springer Verlag, 1996

[11] R. Krishnamurthy, S. Ganguly, and W. Hasan. Query Optimization for Parallel Execution. In *SIGMOD Record*, volume 12, June 1992.

[12] T. Klaustal. Das OSCAR–Projekt. *Informatik-Bericht 92/2, Technische Universität Clausthal*, Institut für Informatik, 1992.

[13] U. Langer and H. Meyer. Mehrstufige Anfrageoptimierung in HEaD. In *Proc. 7. GI-Workshop "Grundlagen von Datenbanken"*, Bad Salzdetfurth, Dez. 1995.

[14] U. Langer and H. Meyer. Join Sequence Optimization in Parallel Query Plans. In *7th Int. Workshop on DEXA*, pages 506–513, Zurich, Sept. 1996.

[15] P. Legato, G. Paletta, and L. Palopoli. Optimization of Join Strategies in Distributed Databases. *Information Systems*, 16(4):363–374, 1991.

[16] Object Design, Inc., Burlington, MA. *ObjectStore C++ -Handbücher, ObjectStore Release 4.0*, June 1995.

[17] M. H. Scholl, D. Gluche, T. Grust, A. Heuer, and J. Kröger : Optimierung von generischen Operationen und Speicherstrukturen in Objekt-Datenbanken (Arbeitsbericht) *Preprint CS-05-97, Fachbereich Informatik, Universität Rostock.* Gleichzeitig erschienen als: Technical Report 32/1997 (*Konstanzer Schriften in Mathematik und Informatik, Nr. 32*), Fakultät für Mathematik und Informatik, Universität Konstanz. April 1997

[18] M. H. Scholl, C. Laasch, C. Rich, M. Tresch, and H.-J. Schek. The COCOON object model. *ETH Zürich, Dept. of Computer Science, Technical Report 192*, December 1992.

[19] M. H. Scholl and H.-J. Schek. Survey of the COCOON project. In R. Bayer, T. Härder, and P.C. Lockemann, editors, *Objektbanken für Experten*, Informatik Aktuell, pages 243–253. Springer Verlag, 1992.

[20] J. Waterstraat. *Definition und Nutzung von Kontexten in mobiler, verteilter Datenbankumgebung.* Diplomarbeit, Universität Rostock, Fachbereich Informatik, 1996.

[21] M. Ziane, M. Zait, and H. H. Quang. The Impact of Parallelism on Query Optimization. In *Proc. of Fifth Workshop on Foundations of Models and Languages for Data and Objects*, pages 127–138, Sept. 1993.

Autoren:

Prof. Dr. Andreas Heuer, Universität Rostock, Fachbereich Informatik, Institut für Praktische Informatik, Lehrstuhl Datenbank- und Informationssysteme, Albert–Einstein–Str. 21, D–18059 Rostock, Tel.: 0381/498–3401, Fax: 381/498–3426, E–mail: heuer@informatik.uni-rostock.de, WWW: http://wwwdb.informatik.uni-rostock.de

Dipl.–Inf. Joachim Kröger, Universität Rostock, Fachbereich Informatik, Institut für Praktische Informatik, Lehrstuhl Datenbank- und Informationssysteme, Albert–Einstein–Str. 21, D–18059 Rostock, Tel.: 0381/498–3406, Fax: 381/498–3426, E–mail: jo@informatik.uni-rostock.de, WWW: http://wwwdb.informatik.uni-rostock.de

Dipl.–Ing. Astrid Lubinski, Universität Rostock, Fachbereich Informatik, Institut für Praktische Informatik, Lehrstuhl Datenbank- und Informationssysteme, Albert–Einstein–Str. 21, D–18059 Rostock, Tel.: 0381/498–3405, Fax: 381/498–3426, E–mail: lubinski@informatik.uni-rostock.de, WWW: http://wwwdb.informatik.uni-rostock.de

Dr.–Ing. Holger Meyer, Universität Rostock, Fachbereich Informatik, Institut für Praktische Informatik, Lehrstuhl Datenbank- und Informationssysteme, Albert–Einstein–Str. 21, D–18059 Rostock, Tel.: 0381/498–3402, Fax: 381/498–3426, E–mail: hme@informatik.uni-rostock.de, WWW: http://wwwdb.informatik.uni-rostock.de