Rostocker Informatik-Berichte (1994) 15

Guntram Flach
Uwe Langer
Holger Meyer

Kostenbasierte Anfrageoptimierung in HEAD†

ABSTRACT

The aim of the project HEAD are examinations on parallel query processing in heterogeneous distributed database management systems. Besides horizontal parallelism implicated by fragmentation and replication of data we focus our attention to pipelining and partitioning. In HEAD methods of load sharing and load balancing are used to control the multiprogramming level and allocation of free operators to achieve an optimal processing of complex queries in the distributed database management system. The optimization should result in the reduction of execution time and a speedup in general by employing non-loaded network nodes.

Keywords: heterogeneous distributed DBMS, parallel processing, pipelining, partitioning, query optimization, join optimization, tree shapes, cost model

1. Einführung

Das Projekt **HEAD** (**H**eterogeneous **E**xtensible **a**nd **D**istributed Database Management System) ist der Prototyp eines heterogenen und verteilten Datenbankverwaltungssystems. Ziel der Entwicklungsarbeiten sind Untersuchungen zur parallelen Anfragebearbeitung in heterogen verteilten Datenbanksystemen und Integration der Analyseergebnisse in den HEAD-Prototypen [Lang93].

Es wird vorausgesetzt, daß Anfragen vorwiegend im Dialogbetrieb an das System gestellt werden. Um zu gewährleisten, daß Anfragen von unterschiedlichen Rechnerknoten aus an das System gerichtet werden können, wird auf allen Rechnerknoten eine einheitliche Sprachschnittstelle zur Verfügung gestellt, die auf einem eingeschränkten SQL-Sprachumfang basiert.

Zur Minimierung der Antwortzeit wird in HEAD von der Möglichkeit der parallelen Abarbeitung von Teilen der Anfrage, z.B. den Operationen SCAN, JOIN und SORT, Gebrauch ge-

† Die Entwicklungsarbeit wurde unterstützt durch die DFG, Kennzeichen: Ge 677/1-1

macht. Grundlegende Techniken, wie z.B. die Parallelverarbeitung von Anfragen, Pipelining [Mikk88] und Lastverteilung [Scha92], sollen die Reaktion des Systems auf interaktive Nutzeranfragen unterhalb bestimmter Zeitschranken gewährleisten. Grundlegendes Entwurfsziel ist eine Verteilungstransparenz, d.h., daß die Verteilung der Daten für den Nutzer unsichtbar bleibt.

2. Systemkomponenten

Die Rechnerknoten in einer heterogenen Umgebung weisen zum Teil eine sehr differenzierte Leistungsfähigkeit auf. Aus diesem Grund ist es nicht sinnvoll, auf allen Rechnern den vollen Funktionsumfang zur Verfügung zu stellen. Dabei wird angestrebt, die verschiedenen Rechner ihrem Leistungsvermögen entsprechend in die Anfrageverarbeitung einzubeziehen.

Als Lösungansatz wird in HE$_A$D die Zerlegung des Systems in relativ eigenständige Funktionseinheiten gewählt, welche den einzelnen Rechnerknoten unter Berücksichtigung der Kapazität und Auslastung zugeordnet werden können. Es werden

- die Funktionen zur Anfrageübersetzung und -optimierung
- und die Funktionen zur Abarbeitung der Ausführungspläne

zu relativ eigenständigen Systemkomponenten zusammengefaßt.

Die Funktionen zur Abarbeitung der Ausführungspläne werden wiederum unterteilt in Funktionen zur Abbildung physischer Datenstrukturen auf Mengen von Tupeln, welche somit die Schnittstelle zur jeweiligen lokalen Datenbank bilden (Local Data Interface, *LDI*) und in Funktionen zur mengenorientierten, datenflußgesteuerten Weiterverarbeitung der eingelesenen Tupel (Siehe Abb. 1).

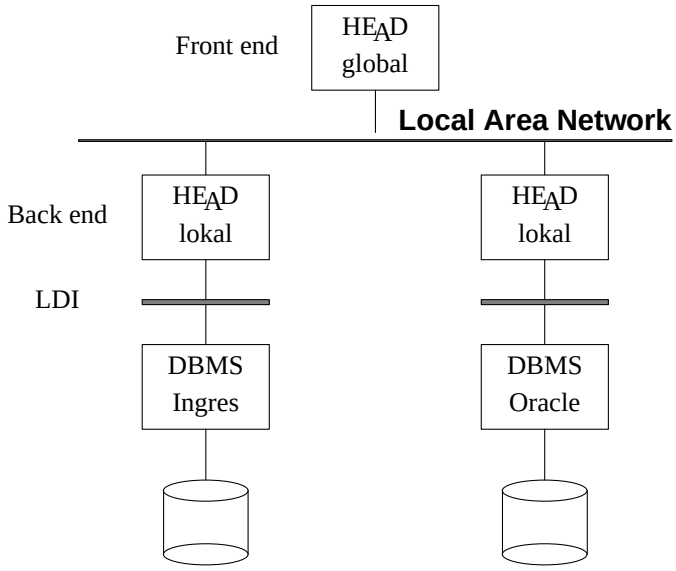

Abb. 1: LDI - Schnittstelle zu lokalen DBMS

Die funktionsorientierte Zerlegung des Systems hat den Vorteil, daß Rechnerknoten der unteren Leistungsklasse einbezogen werden können, andererseits keine zentrale Instanz

existieren muß, die zum Engpaß werden könnte.

Die Funktionen zur mengenorientierten Abarbeitung der Ausführungspläne werden auf allen Rechnerknoten implementiert, welche über die erforderliche Rechenleistung verfügen. Auf diese Weise ist ein beträchtlicher Teil der Anfrageverarbeitung nicht an die Rechnerknoten gebunden, auf denen die in der jeweiligen Anfrage referenzierten Daten gespeichert sind. So ist es möglich, stark ausgelastete Rechenanlagen zu entlasten und andererseits die Rechenkapazität leistungsfähiger Workstations besser auszunutzen.

2.1. Funktionelle Architektur von HE$_A$D

Die funktionelle Architektur von HE$_A$D (siehe Abb. 2) wird bestimmt durch die Zerlegung des Systems in eigenständige Funktionseinheiten sowie durch die Anwendung von Pipelining und Datenflußsteuerung.

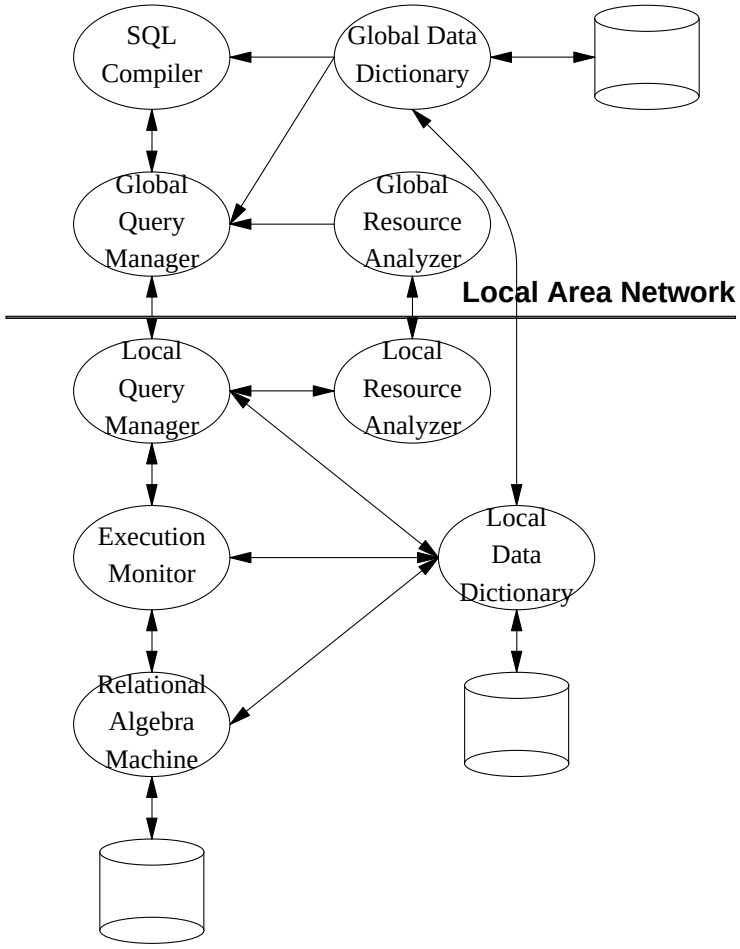

Abb. 2: Funktionelle Architektur von HE$_A$D

Die Komponenten zur Übersetzung, Optimierung und Zerlegung der Anfragen sowie jene zur Aufbereitung und Ausgabe der Ergebnisse sind im SQL-Compiler und Global Query Manager

(*GQM*) zusammengefaßt.

Der SQL-Compiler besteht aus Parser und Präoptimierer. Der Parser enthält die Funktionen zur syntaktischen und semantischen Analyse der Anfragen sowie die Funktionen zur Erzeugung eines der SQL-Anfrage äquivalenten Ausdrucks der Relationenalgebra (primäres Datenflußprogramm). Die primären Datenflußprogramme basieren auf dem globalen konzeptuellen Schema, d.h., daß der Speicherungsort der betroffenen Daten zunächst unberücksichtigt bleibt. Die Transformation von Ausdrücken des in HE$_A$D eingeschränkten SQL-Sprachumfanges in Ausdrücke der Relationenalgebra erfolgt weitestgehend nach dem in [Ceri85] vorgeschlagenen Verfahren.

Das vom Parser erzeugte primäre Datenflußprogramm wird durch den Präoptimierer mit dem Ziel der Verringerung der Antwortzeit umgeformt. Dabei kommen vorwiegend die aus der klassischen algebraischen Anfrageoptimierung bekannten Transformationen in semantisch äquivalente Datenflußprogramme zur Anwendung [Ozsu91]. Der Parser führt neben der eigentlichen Übersetzung auch eine Normalisierung durch. Dies geschieht auf der Grundlage von Informationen aus dem globalen Schema und speziell dem Fragmentationsschema.

Bei gegebener horizontaler oder vertikaler Fragmentierung werden die Selektionsoperationen über den Relationen des globalen Schemas in Selektionsoperationen über den Fragmenten aufgefächert und die entsprechenden Vereinigungs- (UNION) und/oder Verbund- (JOIN) Operationen in den Anfragebaum eingefügt.

Der Global Query Manager transformiert den vom SQL-Compiler gelieferten globalen Anfragebaum in parallel auf mehreren Rechnerknoten ausführbare Teilanfragebäume. Durch die Aufteilung der Teilanfragebäume auf geeignete Rechnerknoten unter Berücksichtigung der aktuellen Lastsituation im Gesamtsystem wird eine ausgeglichene Lastverteilung erreicht. Dabei wird neben der Parallelisierung der Anfragebäume eine Optimierung hinsichtlich des Antwortzeitverhaltens durchgeführt. Der GQM entscheidet aufgrund der Informationen aus dem Allokationsschema, der aktuellen Lastsituation sowie bestimmter Kostenberechnungen über die Zuordnung von Operatoren zu Rechnerknoten. Durch den GQM wird auf jedem betroffenen Rechnerknoten ein Local Query Manager (*LQM*) gestartet, der die auf dem Rechnerknoten lokal abzuarbeitende Teilanfrage entgegennimmt und Ausführungsmonitore (Execution Monitor, *EM*) aktiviert. Außerdem wird in Abhängigkeit von der momentanen Rechnerlast die Lastbalancierung organisiert. Der GQM stellt somit die Lastverteilungskomponente (Loadsharing†) dar, die neben der durch die Local Query Manager durchgeführten Lastbalancierungen (Loadbalancing) für eine Minimierung der Antwortzeit einer Anfrage sorgt. Der Global Query Manager besteht aus Optimierer (siehe Abschnitt Optimierer) und einer Dekompositionskomponente, die den parallel ausführbaren Anfragebäume in Teilbäume zergliedert und an die entsprechenden Rechnerknoten versendet.

Im Falle einer Überlastsituation organisiert der LQM gegebenenfalls die Umverlagerung von Algebraprozessen auf andere Rechnerknoten.

Der EM nimmt die auf dem jeweiligen Rechnerknoten abzuarbeitende Teilanfrage entgegen, analysiert diese und organisiert die Abarbeitung jeder Algebraoperation.

Die Relationale Algebra Maschine (Data driven Relational Algebra Machine, *DRAM*) stellt die Operationen der relationalen Algebra bereit und organisiert Pipelining- und Interprozeßkommunikationsmechanismen. Gleichzeitig wird durch die SCAN-Operation der Zugriff auf

† Im Rahmen von HE$_A$D wird eine statische Lastverteilung als Loadsharing bezeichnet.

lokale Datenbanken durchgeführt.

Notwendige Lastmessungen werden von den Lastmessungskomponenten, dem Global Resource Analyzer (*GRA*) sowie dem Local Resource Analyzer (*LRA*) durchgeführt.

Der Global Resource Analyzer stellt eine Schnittstelle zwischen den Local Resource Analyzers und dem Global Query Manager dar. Diese Komponente verwaltet die zyklisch von den einzelnen LRA gesendeten Lastinformationen (Lastdeskriptoren). Somit wird eine globale Sicht auf die momentane Last der Rechnerknoten erreicht und eine schnelle und kostengünstige Bereitstellung von Lastinformationen für die Lastverteilung und Lastbalancierung ermöglicht (siehe Abb. 3).

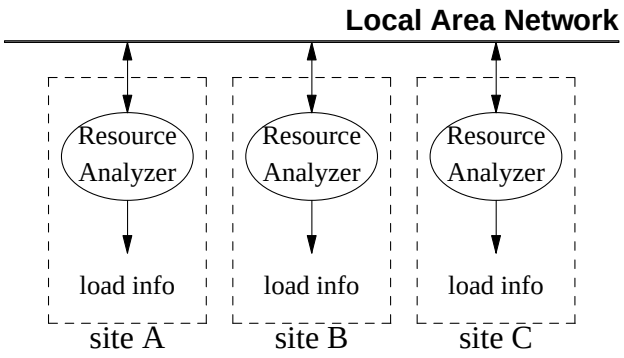

Abb. 3: Lastanalyse

Der Local Resource Analyzer führt auf jedem Rechnerknoten Lastmessungen durch, die Grundlage für eine effiziente Verteilung der Algebraprozesse sind. Die Entscheidungen hinsichtlich der Verteilung der Prozesse auf die verfügbaren Prozessoren wird in Abhängigkeit von aktuellen Lastparametern der einzelnen Prozessoren getroffen.

Die Aufgaben der Metadatenverwaltung auf globaler bzw. lokaler Ebene werden durch die Systemkomponenten Global Data Dictionary (*GDD*) und Local Data Dictionary (*LDD*) übernommen.

2.2. Anfrageverarbeitung in HE$_A$D

Die Anfrageverarbeitung in HE$_A$D ist in Übersetzungs- und Ausführungsphase gegliedert. In der Übersetzungsphase werden SQL-Anfragen in primäre Datenflußprogramme übersetzt, optimiert und in lokale Datenflußprogramme zerlegt. In der Ausführungsphase wird für jedes lokale Datenflußprogramm zunächst eine Hierarchie durch Pipelines verbundener DRAM-Prozesse installiert. Mit der Aktivierung dieser Prozesse beginnt dann die eigentliche datenflußgesteuerte Abarbeitung der lokalen Datenflußprogramme. Die Ergebnisse einer Anfrage werden zum Ursprungsknotens übertragen, wo sie aufbereitet und ausgegeben werden.

3. Der Optimierer

Die vom GQM erzeugten parallelen Verarbeitungspläne müssen hinsichtlich der Antwortzeit unter Berücksichtigung von Prozessor-, Kommunikations-, I/0- und anderen Kosten optimiert werden. In die Bearbeitung einer Datenbankanfrage werden all jene Rechnerknoten einbezogen, die (tendenziell) gering oder nicht belastet sind und weiterhin bestimmte Voraus-

setzungen für die Bearbeitung einer Teilanfrage besitzen wie hohe Leistungsfähigkeit u.a. Der Optimierer besteht aus drei folgenden Hauptkomponenten:

- Baumumformkomponente (Rewriter)
- unabhängiger Optimierungsalgorithmus
- Kostenkomponente

Der Rewriter führt die vom Optimierungsalgorithmus vorgeschlagenen Umformungen der Anfragebaumstruktur durch, die mit Hilfe der Kostenkomponente bewertet werden. Dabei müssen die abstrakten Darstellungen des Optimierers in konkrete QEP-Strukturen umgeformt werden. Der modulare Aufbau des Optimierers hat den Vorteil, daß der Optimierungsalgorithmus leicht gegen andere bzw. parallele Verfahren ausgetauscht werden kann. In HE$_A$D kommt derzeit ein approximatives Optimierungsverfahren zum Einsatz, welches auf Simulated Annealing beruht (ASA, [Ingb89]). Dieses Verfahren bietet den Vorteil, daß es die Optimierung multidimensionaler Suchräume unterstützt.

3.1. Optimierungsziel

In [Scha92] wird die Hypothese aufgestellt, daß das größte Potential für eine Leistungssteigerung in einem verteilten System in einer Gleichverteilung der Rechenlast der aktiven Prozesse s_i auf alle verfügbaren Rechnerknoten liegt (siehe Abb. 4).

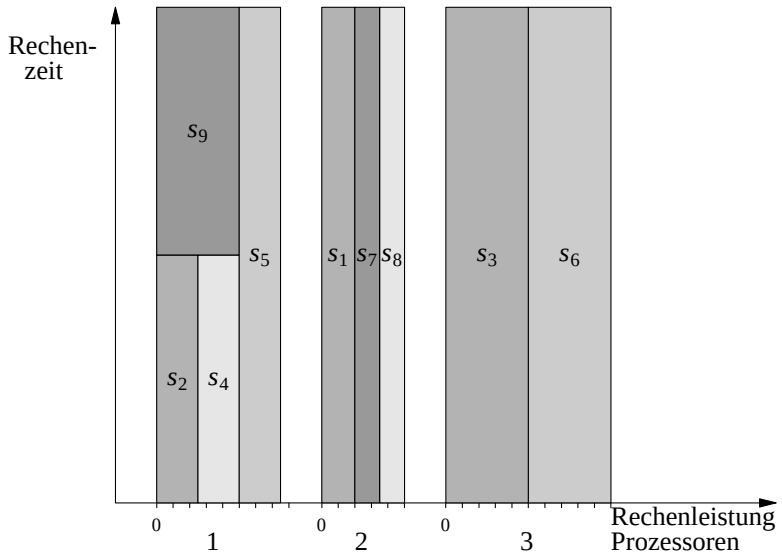

Abb. 4: Ideale Prozeßaufteilung

Da in HE$_A$D keine globale Sicht angestrebt wird, das heißt, bei unabhängig voneinander gestarteten Datenbankanfragen wird keine aufeinander abgestimmte Optimierung durch die zugehörigen HE$_A$D-Komponenten durchgeführt, ist eine faire Aufteilung der Ressourcen bzgl. der konkurrierenden DB-Anfragen nicht möglich. Eine durchsatzorientierte Optimierung könnte daher nur bedingt erfolgen. Optimiert wird deshalb die Antwortzeit einer einzelnen DB-Anfrage. Die Teilanfragebäume sind dabei so auf die zur Verfügung stehenden Rechner-

knoten aufzuteilen, daß ein Minimum hinsichtlich der Antwortzeit der gesamten Datenbankanfrage an das verteilte System erreicht wird.

3.2. Abbildung eines Anfragebearbeitungsplanes auf ein Rechnernetz

3.2.1. Darstellung des Anfragebaumes

Eine Anfrage an das verteilte Datenbanksystem wird intern als ein Bearbeitungsplan (QEP) dargestellt. Dieser Bearbeitungsplan läßt sich graphisch repräsentieren, indem die Knoten relationale Algebraoperatoren und die gewichteten Kanten die Kommunikationsverbindungen d_i in einem gerichteten Graphen darstellen. (siehe Abb. 5).

3.2.2. Darstellung des Rechnernetzes

Es existiert ein lokales Netz von Rechnerknoten, wobei sich das verteilte System im einfachsten Falle auf ein Subnetz mit n Rechnerknoten beschränkt. Das Rechnernetz kann ebenfalls als Graph betrachtet werden. Dabei wird jeder Rechnerknoten als Knoten und jede Kommunikationsverbindung zwischen den Rechnerknoten als gerichtete Kante dargestellt (siehe Abb. 6).

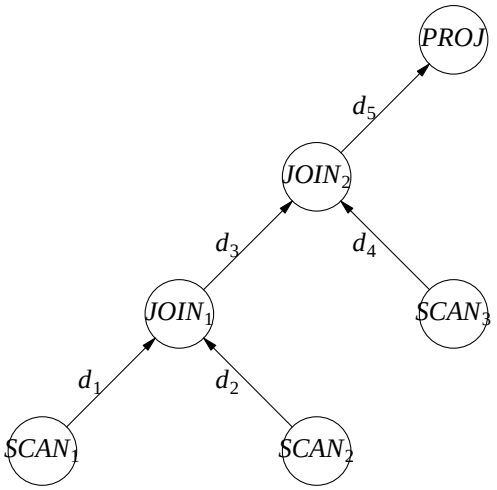

Abb. 5: Einfacher Bearbeitungsplan

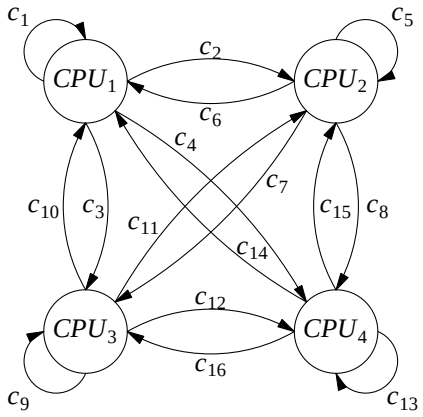

Abb. 6: Lokales Subnetz

Zwischen den n Rechnerknoten existieren l Kommunikationsverbindungen mit $l \leq n(n-1)$. (Die lokalen Kommunikationsverbindungen auf den einzelnen Rechnerknoten sind dabei vernachlässigt.) Prinzipiell müssen nicht alle Kanten unbedingt vorhanden sein, das heißt, der Graph muß nicht vollständig sein. Es reicht bereits aus, wenn der Graph zusammenhängend ist und jeder Knoten von jedem anderen Knoten aus erreicht werden kann.

3.2.3. Abbildung des QEP auf die Rechnerknoten

Bildet man den Graphen des Bearbeitungsplanes auf den Graphen der Rechnerknoten ab, erzeugt man einen neuen Graphen, der alle Möglichkeiten der Ausführung aller freien Operatoren auf jedem verfügbaren Rechnerknoten enthält.

Die gebundenen Operatoren (SCAN) können in Abhängigkeit der Lokation der Replikate ebenfalls auf verschiedene Rechnerknoten abgebildet werden (siehe Abb. 7).

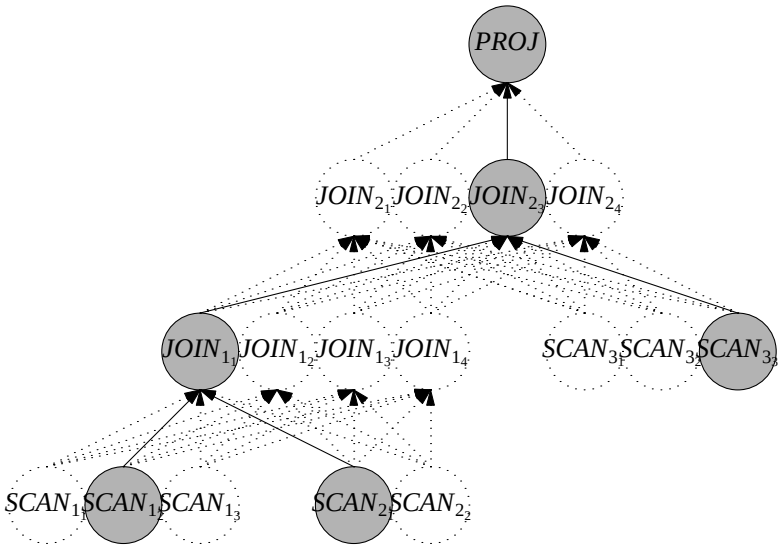

Abb. 7: Abbildung freier und gebundener Operatoren eines QEP auf ein Rechnernetz

Bei der Abbildung des QEP auf das Rechnernetz entsteht eine regelmäßige Struktur, die relativ einfach eineindeutig auf ein Paar numerischer Werte projiziert werden kann, wobei ein Wert eine Rechnerknotenbelegung durch freie Operatoren und der andere Wert die Rechnerknotenbelegung durch die gebundenen Operatoren darstellt. Beide Wertebereiche sind ganzzahlig kontinuierlich innerhalb der zulässigen Intervallgrenzen im Intervall $[1 \cdots \mathit{Alternativenanzahl}]$. Da die Abbildung eineindeutig ist, läßt sich aus dem Wertepaar ebenfalls der Anfragegraph rekonstruieren. Diese Eigenschaft ist bei der Optimierung der QEP sinnvoll einsetzbar. Mit der Abbildung der freien und gebundenen Operatoren auf das Rechnernetz ist jedoch der Suchraum für den zu optimierenden Anfragebaum noch nicht vollständig beschrieben. Es gibt weitere Bestandteile des QEP, die ebenfalls optimiert werden müssen.

3.3. Der Suchraum

Die Optimierung der parallelen QEP besteht in der Auswahl des optimalen parallelen QEP aus der Menge aller möglichen QEP, die den Suchraum darstellt. Die Suchraumgröße ist abhängig von der Menge aller zu optimierenden Kriterien des QEP, die nachfolgend erläutert werden. Die zu optimierenden Kriterien des QEP sind:

- Abbildung der freien Operatoren auf die Menge der Rechnerknoten (Allokation der freien Operatoren)
- Abbildung der gebundenen Operatoren auf die Menge der replikatführenden Rechnerknoten (Allokation der gebundenen Operatoren)
- Abbildung der verfügbaren Joinoperatortypen auf die Joinoperationen (Joinselektion)
- Formen der Parallelität (Tree shapes)
- Grad der Multiprogrammierung des Anfrageplanes
- Grad der Intraoperatorparallelität

3.3.1. Abbildung der freien Operatoren auf die Menge der Rechnerknoten

Bildet man die Menge der freien Operatoren auf die Menge der verfügbaren Rechnerknoten ab, so erhält man k^m verschiedene mögliche Pfade, die alle eine Teillösung darstellen, wobei k die Menge der freien Rechnerknoten und m die Menge der freien Operatoren darstellt.

3.3.2. Abbildung der gebundenen Operatoren auf die Menge der replikatführenden Rechnerknoten

Durch die Replikate der SCAN-Operatoren entstehen zusätzliche Auswahlmöglichkeiten für die Allokation der SCAN-Operatoren. Es gibt für n SCAN-Operatoren $\prod_{i=1}^{n} |Replikate_{SCAN_i}|$ verschiedene Kombinationen von SCAN-Operatoren. Durch die in HE$_A$D verwendete Datenflußsteuerung besitzt die Verteilung der SCAN-Operatoren eine besondere Bedeutung, da eine von den SCAN-Operatoren vorgegebene Flußgeschwindigkeit nicht mehr erhöht werden kann. Das bedeutet, daß diese initiale Flußgeschwindigkeit direkt die Antwortzeit des Systems beeinflussen kann.

3.3.3. Abbildung der Joinoperatortypen auf die Joinoperatoren

In HE$_A$D kommen verschiedene Joinverfahren zum Einsatz. In Abhängigkeit der Anordnung des Join, des Joinoperatortyps (Sort merge, Nested loop, Hash) und anderen Faktoren (Sortierung, Index) wird der Datenfluß zeitweise gestoppt bzw. ein Datenfluß bedingt ermöglicht. In einem System, welches auf Datenfluß basiert, kommt letzterem eine besondere Bedeutung zu. Besonders Hash joins werden eingesetzt, um einen weitgehend ungehinderten Datenfluß zu gewährleisten. Inwieweit dies möglich ist, hängt in starken Maße vom verfügbaren Internspeicher ab. Um einen gleichmäßigen Datenfluß zu erreichen, wird für alle auf einem Rechnerknoten allokierte Hash joins gleichzeitig der Speicherbereich für die Hashtabelle der inneren Relation angefordert. Die Folge könnte ein großer Aufwand für ständiges Ein- und Auslagern von Seiten vom Intern- auf den Externspeicher und umgekehrt sein, wodurch sich die Antwortzeit verschlechtert. Eine Unterbrechung des Datenflusses durch einen Join (Sort merge) dagegen kann den Speicherbedarf erheblich reduzieren. Bildet man die Menge der Joinoperatortypen auf die Menge der Joinoperationen ab, so erhält man j^l verschiedene Joinkombinationen, wobei j die Anzahl der Joinoperationen und l die Anzahl der Joinoperatortypen darstellt.

3.3.4. Formen der Parallelität (Tree shapes)

In HEAD werden vier Formen der Parallelität betrachtet: *horizontale Parallelität, Pipeline-Parallelität, Intra-Operator-Parallelität* und die *Bushy-Tree-Parallelität* [Grae90]. Die Verteilung der Datenbank in Fragmente und Replikate erzwingt eine Aufteilung des QEP, der eine Datenbankanfrage für das konzeptuelle Schema repräsentiert, in mehrere parallel bearbeitbare Sub-QEP's, die den Zugriff auf lokale Datenbanken beschreiben. Diese Parallelität stellt eine **horizontale Parallelität** dar. Durch die Verwendung einer Datenflußsteuerung und aufeinander abgestimmter Algorithmen ist eine parallele Ausführung logisch aufeinanderfolgender Operationen möglich [Naka88]. Daraus ergibt sich eine Antwortzeitverkürzung. Diese Art der Parallelität wird als **Pipeline-** oder auch **vertikale Parallelität** bezeichnet.

Einige Operationen (insbesondere Join und Sortierung) stoppen den Datenfluß aus Gründen der Komplexität bzw. der zugrundeliegenden Algorithmen. Dadurch wird die vertikale Parallelität eingeschränkt. Durch Partitionierung des Datenstromes in mehrere Teilströme **(Intra-Operator** oder **Datenparallelität)** und der damit verbundenen Volumenreduzierung der Teildatenströme kann eine Reaktionszeitverbesserung erzielt und der Grad der vertikalen Parallelität erhöht werden.

Eine weitere Möglichkeit der Erhöhung der horizontalen Parallelität ist die Reduktion der Baumhöhe des QEP. Diese Form der Parallelität, die **Bushy-Tree-Parallelität,** wird durch eine äquivalente Umformung des QEP erzielt. Ein typisches Beispiel ist das Rewriting von JOIN-Listen [Lanz91].

Die Anordnung der Joinoperationen entscheidet wesentlich darüber, welche Form der Parallelität vorliegt. Zwei Extremformen der Parallelität werden dabei durch Left deep trees und Right deep trees repräsentiert, die eine maximale bzw. eine minimale vertikale Parallelität ermöglichen. Der "balancierte" Bushy tree ist eine dritte Extremform. Er unterstützt zwar nur bedingt die vertikale Parallelität, dafür aber optimal die horizontale (echte) Parallelität. Zwischen den drei genannten Extremformen gibt es Übergangsformen, deren Anzahl von der Anzahl der Joinoperationen im entsprechenden Teilbaum beeinflußt wird. Bei der verwendeten Form der Parallelität muß wie bei der Wahl der Joinoperatortypen ein Kompromiß zwischen einem gleichmäßigen Datenfluß und der Größe des verfügbaren Externspeichers getroffen werden.

3.3.5. Grad derMultiprogrammierung des Anfrageplanes

In HEAD gibt es prinzipiell vier Möglichkeiten, eine relationale Algebraoperation bearbeiten zu lassen:

- als jeweils ein separater Prozeß auf einem Rechnerknoten unter Nutzung globaler Kommunikationsmedien
- zusammen mit anderen relationalen Operatoren auf einem Rechnerknoten als separate Prozesse unter Nutzung lokaler und globaler Kommunikationsmedien
- zusammen mit anderen relationalen Operatoren als Leichtgewichtsprozeß (thread) innerhalb eines Prozeßkontextes unter Nutzung lokaler und globaler Kommunikationsmedien
- als Teilanfrage, die einem lokalen DBMS übergeben wird

Diese vier Formen können miteinander kombiniert werden. Falls Algebra-Operatoren als Leichtgewichtsprozesse zusammengefaßt werden, wird auf jeden Fall Rechenzeit für

kostenintensive Prozeßumschaltungen eingespart. Da sich die CPU selbst in stark belasteten Systemen noch einen Großteil der Zeit im Wartezustand befindet, dagegen die meisten Prozesse auf periphere Geräte warten müssen, zeichnet sich ab, welches Potential zur Leistungssteigerung in Threads liegt. Gerade derartige Zusammenfassungen von Algebraoperationen mit einem gleichmäßigen Datenfluß nutzen fast ausschließlich die CPU und den Primärspeicher. Im Gegensatz zu eigenständig auf unterschiedlichen Rechnerknoten ausgeführten Algebraoperationen ist der Zugriff auf andere Ressourcen sehr gering. Nachteilig wirkt sich dabei jedoch aus, daß der Rahmenprozeß die Ressourcen "fair" zugeordnet bekommt, ohne jedoch zu berücksichtigen, daß dieser Prozeß mehrere Prozesse repräsentiert.

3.3.6. Grad der Intraoperatorparallelität

Besonders aufwendige Operationen können selbst wiederum parallel ausgeführt werden. Dadurch ist es möglich, Operationen, die den Datenfluß extrem verlangsamen, zu beschleunigen und somit einen Engpaß zu beseitigen.

3.4. Abhängigkeiten der Optimierungskriterien

Die im letzten Absatz genannten Optimierungskriterien beeinflussen sich zum Teil gegenseitig. Dies wird in Tabelle 1 verdeutlicht.

Optimierungskriterien						
Kriterium	AFO	AGO	JNS	TRS	MPL	IOP
AFO	-	0	1	1	2	0
AGO	0	-	0	0	2	0
JNS	0	0	-	1	0	0†
TRS	0	0	0	-	2	1
MPL	0	0	0	0	-	1
IOP	0	0	0	0	0	-

AFO Allokation der freien Operatoren
AGO Allokation der gebundenen Operatoren
JNS Joinselektion
TRS Formen der Parallelität (Tree shapes)
MPL Grad der Multiprogrammierung (Multiprogramming level)
IOP Intraoperatorparallelität

Tab. 1: Abhängigkeiten der Optimierungskriterien

Starke Abhängigkeiten bestehen in der Form, daß bei einer Änderung eines Kriteriums während der Optimierung zwangsweise ein andere Kriterium ebenfalls geändert werden muß.‡ Diese sind in der Tabelle mit 2 gekennzeichnet. Falls eine schwache Abhängigkeit vorliegt, ist sie in der Tabelle mit 1 gekennzeichnet, sonst mit 0. Schwache Abhängigkeiten können entweder ignoriert werden bzw. sie können mittels Modifikation des Kostenwertes des jeweiligen Anfrageplanes ausgeglichen werden. Reicht zum Beispiel der vorhandene Intern-

† Es wird vorausgesetzt, daß alle Joinverfahren intraparallel bearbeitet werden können.

‡ Die genannten Abhängigkeiten sind nur zur Optimierungszeit von Bedeutung.

speicher nicht aus, sind die Kosten des Anfrageplanes zu erhöhen.

Indirekte Abhängigkeiten, sofern sie nur eine Lastveränderung auf den betroffenen Rechnerknoten erzeugen, werden nicht betrachtet. Anschließend werden die Abhängigkeiten der einzelnen Kriterien erläutert.

- AFO/JNS: Es sollte getestet werden, ob auf den betroffenen Rechnerknoten genügend freier Internspeicher existiert.
- AFO/TRS: siehe AFO/JNS
- AFO und AGO/MPL: Bei der Verlagerung Operatoren auf andere Rechnerknoten kann der momentane Grad der Multiprogrammierung unter Umständen nicht beibehalten werden, falls der betroffene Operator bereits als Leichtgewichtsprozeß vorgesehen ist.
- JNS/TRS: Der Datenfluß wird möglicherweise erheblich gestört. Außerdem siehe AFO/JNS.
- TRS/MPL: Eine Veränderung der Form der Parallelität verändert möglicherweise die Reihenfolge und die Verbindungen zwischen den Operatoren im Operatorbaum. Bereits als Leichtgewichtsprozesse vorgesehene Operationen verhindern daher die Umformung des Anfragebaumes.
- TRS/IOP: Intraoperatorparallelität möglicherweise gegenstandslos
- MPL/IOP: siehe TRS/IOP

Aus der Tabelle wird ersichtlich, daß eine Optimierung des Grades der Multiprogrammierung (MPL) zusammen mit den anderen Optimierungskriterien schwierig zu realisieren ist. Aus diesem Grund wird dieses Kriterium im Zusammenhang mit der Optimierung des parallelen Anfragebaumes vorerst nicht weiter berücksichtigt.

4. Kostenmodell

4.1. Begriffsbestimmung

Wenn ein verteiltes Datenbanksystem (*VDBS*) aus der Benutzersicht betrachtet wird, entsprechen die Kosten zur Bearbeitung einer Anfrage der Wartezeit von der Formulierung der Anfrage bis zur Bereitstellung des Ergebnisses. Folgende zusätzliche Aspekte, die sich aus dem Vorhandensein mehrerer autonomer, heterogener Rechner im VDBS ergeben, die durch ein Kommmunikationssystem miteinander verbunden sind, müssen beachtet werden:

- *Kommunikations- und Datenübertragungszeit* zwischen den Rechnerknoten
- Berücksichtigung neuer Möglichkeiten der *Parallelverarbeitung einer Anfrage*
- Berücksichtigung der *Partitionierung* oder *Replikation* von Daten

Definition Kostenmodell:

> Modell, das die bei der Bewertung semantisch äquivalenter Ausführungspläne durch den Optimierungsalgorithmus berücksichtigten Kostenkomponenten und ihre Wichtung widerspiegelt.

Der Begriff *Kosten* bezieht sich hier jeweils auf die benötigte Zeit zur Verarbeitung einer Anfrage.

Im System HEAD werden neben herkömmlichen Kostenabschätzungsstrategien [Pram88, Mikk88] neue Aspekte in die Kostenanalyse aufgenommen, die mögliche differenzierte

Einflußfaktoren im VDBS berücksichtigen.

Durch entsprechende Untersuchungen [Flac93] wurde nachgewiesen, daß die Integration der aktuellen Netzbelastung sowie der aktuellen Prozessorauslastung in das Kostenmodell notwendig ist. Durch eine statische Lastanalyse der Lastkomponente des HE$_A$D-Systems werden verschiedene Meßgrößen zu einem Netzbelastungsfaktor bzw. Prozessorbelastungsfaktor zusammengefaßt, die dann in die Berechnung innerhalb der Kostenabschätzung eines Anfragegraphen einbezogen werden. Durch diese Einbeziehung der aktuellen Lastsituation im VDBS wird eine höhere Exaktheit der Kostenanalyse erreicht und Veränderungen der Leistungsfähigkeit einzelner Rechnerknoten im verteilten System beachtet.

Die bei der Abarbeitung eines Ausführungsplanes auftretenden Kosten lassen sich in folgende Komponenten aufgliedern:

- Übertragungskosten
- Externspeicherzugriffskosten
- CPU-Kosten
- aus Konkurrenzsituationen resultierende Kosten (Warten auf Ressourcen)
- Folgekosten

Durch die Wichtung der einzelnen Kostenkomponenten können die spezifischen Leistungsmerkmale des verteilten Systems und ihre Bedeutung für die Kostenanalyse mit berücksichtigt werden. Für das HE$_A$D-System wird von der in Abb. 8 dargestellten Wichtung der einzelnen Kostenkomponenten ausgegangen.

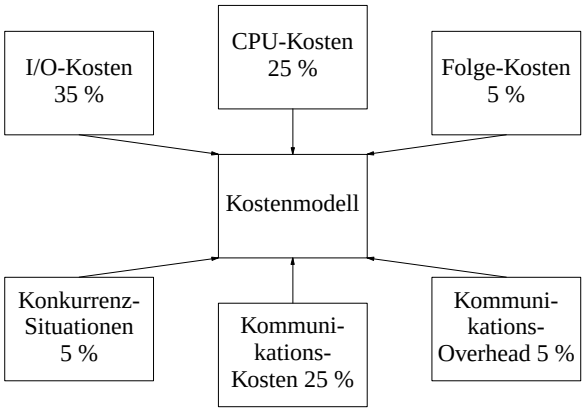

Abb. 8: Wichtung der Kostenkomponenten

Die dargestellten Wichtungsverhältnisse stellen eine Zusammenfassung verschiedener in der Literatur verwendeter Lösungsansätze dar und wurden speziell auf die Charakteristiken des HE$_A$D-Systems angepaßt.

4.2. Kostenfunktionen

Zur Ermittlung des optimalen Ausführungsplanes können alle möglichen, semantisch äquivalenten Ausführungspläne generiert und der Reihe nach auf der Basis des Kostenmodells bewertet werden. Da bei einer großen Anzahl von möglichen Ausführungsplänen diese Vorgehensweise selbst sehr kostenintensiv werden kann, wird im HEAD-System die Entwicklung geeigneter Heuristiken vorgenommen. Es kann aus einer Klasse äquivalenter Ausdrücke derjenige Ausdruck als optimal angesehen werden, dem bzgl. einer Kostenfunktion ein minimaler Wert zugeordnet wird.

Parameter		**Einheit**
$C_{cpu}(i)$	CPU-Geschwindigkeit auf dem Rechner i	ms/inst
$C_{io}(i)$	I/O-Zugriffsgeschwindigkeit auf dem Rechner i	ms/byte
$C_{tr}(i, j)$	Zeit die benötigt wird, um ein Byte zwischen den Rechnern i und j zu übertragen	ms/byte
$C_{msg}(i, j)$	Kommunikations-Overhead zur Übertragung einer Nachricht zwischen den Rechnern i und j	ms/msg
C_{comm}	Kommunikationskosten	ms
C_{local}	lokale Verarbeitungskosten	ms
$C_{load}(i)$	CPU-Belastung	
C_{net}	Netz-Belastung	
ω_i	Operator i	
$S(\omega_i)$	Rechner auf dem ω_i ausgeführt wird	
ρ_i	temporäre Relation i, (HEAD-Tupelstrom)	
$l(\rho_i)$	Größe von ρ_i	bytes
$S(\rho_i)$	Rechner auf dem Zwischenergebnisrelation ρ_i vorliegt	
$P(i, j)$	1, wenn ρ_i nicht auf dem Rechner vorliegt, auf dem Operator ω_i ausgeführt wird, 0 anderenfalls	
$l_{\max}$	maximale Datenpaketlänge (Ethernet)	bytes
$ceil(x)$	Rundungsfunktion	
N_{io}	Anzahl der Dateneinheiten (Byte), die zwischen Externspeicher und Hauptspeicher übertragen werden	
N_{inst}	Anzahl der Instruktionen in Abhängigkeit von der relationalen Operation (pro Tupelpaket	
q	Anzahl der Zwischenergebnisrelationen, die benötigt werden	
k	Anzahl der Operationen des Anfragegraphen	
$card(\rho_i)$	Anzahl der Tupel in der Relation ρ_i	
$size(\rho_i)$	Größe der Tupel	bytes
$val(A[\rho_i])$	Anzahl unterschiedlicher Werte des Attributes A der Relation ρ_i	
τ	Tupelanzahl pro Tupelpaket	

Tab. 2: Parameter des Kostenmodells

Die Bestimmung der Kommunikationskosten zwischen zwei Rechnerknoten ist abhängig von der Größe der Relation, die übertragen wird. Die Abschätzung der Größe der Ausgaberelation stellt sich dar als eine Funktion der ausgeführten Operation, die abhängig ist von der Größe der Eingaberelation sowie den Attributeigenschaften. Über die Metadatenverwaltung können Informationen über die Anzahl der Tupel einer Relation, die Länge jedes Attributes, des Attributwertbereiches und die Zahl der unterschiedlichen Werte innerhalb des Wertebereiches des Attributes gewonnen werden.

Zielfunktion für die Kommunikationskosten:

$$C_{comm} = \sum_{j=1}^{k} \sum_{i=1}^{q} (C_{msg}(S(\rho_i), S(\omega_j)) * ceil(l(\rho_i)/l_{\max}) + C_{tr}(S(\rho_i), S(\omega_j)) * l(\rho_i)) * P(i, j) \quad (1)$$

Die Bestimmung der lokalen Verarbeitungskosten einer Teilanfrage ist abhängig von der Art der algebraischen Operation (Projektion,Selektion,..). Die Größe der Zwischenergebnisrelation, als Eingangsgröße des Operators, kann durch spezifische Berechnungsvorschriften [Ceri85] abgeschätzt werden. Die CPU-Geschwindigkeit $C_{cpu}(i)$ und die Externspeicherzugriffsgeschwindigkeit $C_{io}(i)$ gehen als rechnerknotenabhängige Größen in die Kostenfunktion ein. Auf Grund des starken Einflusses von Prozessorlastschwankungen (Swapping) auf die Leistungsfähigkeit eines Rechnerknoten und der damit verbundenen Erhöhung der lokalen Verarbeitungskosten wird ein Prozessorbelastungsfaktor $C_{load}(i)$ in die Abschätzungsfunktion integriert.

Zielfunktion für die lokalen Verarbeitungskosten:

$$C_{local} = N_{io}(\omega_i) * C_{io}(i) + N_{inst}(\omega_i) * C_{cpu}(i) * \frac{1}{1 - C_{load}(i)} \quad (2)$$

Da im Rahmen des HEAD-Systems das Verfahren der Lastbalancierung [Link94] angewendet wird, ist es notwendig, basierend auf einer gemessenen Last, eine Abschätzung der entstehenden Kosten für die Verlagerung relationaler Operationen auf andere Rechnerknoten durchzuführen. Durch das Pipelining (vertikale Parallelität) kommt es zu einer wechselseitigen Beeinflussung mehrerer Operatoren auf einem gemeinsamen Rechnerknoten in bezug auf die momentane CPU-Belastung. Deshalb ist es notwendig, diese Laständerung in die Kostenfunktionen des Kostenmodells zu integrieren. Die Instruktionsanzahl N_{inst} wurde auf der Grundlage von Benchmark-Testergebnissen an den Systemen POSTGRES und INGRES als vorläufige Richtwerte ermittelt.

Zielfunktion für die Laständerung:

$$C_{load}'(i) = \frac{C_{load}(i) * \Delta t * C_{cpu}(i) + N_{inst}}{\Delta t * C_{cpu}(i)} \quad (3)$$

4.3. Abschätzung der Zwischenergebnisrelationen

Grundlage für die Aktualisierung (Berechnung) der Metadaten nach dem Ausführen einer relationalen Operation ω_i sind die Berechnungsvorschriften nach [Ceri85]. Ozsu und Valduriez [Ozsu91] weisen insbesondere auf die Notwendigkeit einer exakten Bestimmung der Metadaten nach der Ausführung einer algebraischen Operation hin.

Die Grundlage für die Gültigkeit dieser Berechnungsvorschriften ist die Annahme, daß die Werte aller Attribute einer Gleichverteilung unterliegen. Für eine genaue Abschätzung der Ergebnisse nach der Ausführung einer relationalen Algebraoperation müßte zuerst eine Verteilungsfunktion für die Werte der einzelnen Attribute ermittelt und auf dieser Grundlage das zu erwartende Ergebnis abgeschätzt werden.

Da diese Vorgehensweise jedoch zu einem hohen Berechnungsoverhead führen würde und mit vertretbarem Aufwand nicht zu realisieren ist, wird in vielen Datenbanksystemen von einer Gleichverteilung der Werte ausgegangen.

Keine Berechnungsvorschriften oder Näherungsformeln existieren für die relationale Algebraoperation Selektion der Form: *Attr v_op Wert*, *mit v_op* = ($<, \leq, >, \geq$). Für eine derartige Abschätzung sind Kenntnisse über die Wertebereiche der einzelnen Attribute notwendig, d.h., daß für jedes Attribut in den Metadaten der jeweils größte und kleinste Attributwert bekannt sein müßte. Unter Annahme der Gleichverteilung könnten dann für die Berechnung der Kardinalität der Ergebnisrelation Näherungsformeln ermittelt werden.

Nachfolgend wird ein Berechnungsansatz aufgezeigt, bei dem die Metadaten um die jeweiligen Grenzwerte der Wertebereiche erweitert werden:

Prädikat	**Selektivitätsfaktor**	**card(T)**	**size(T)**
$Attr = value$	$\frac{1}{val(A[R])}$	$\frac{card(R)}{val(A[R])}$	$size(T) = size(R)$
$Attr > value$	$\frac{(max-value)}{(max-min)}$	$\frac{((max-value)*card(R))}{(max-min)}$	$size(T) = size(R)$
$Attr < value$	$\frac{(value-min)}{(max-min)}$	$\frac{((value-min)*card(R))}{(max-min)}$	$size(T) = size(R)$

Tab. 3: Selektivitätsfaktoren

In der weiteren Entwicklungsarbeit wird im HE$_A$D-System zur Verbesserung der Abschätzungsergebnisse eine Statistik-Komponente eingesetzt, mit deren Hilfe unter Verwendung von Histogrammen Aussagen über die Verteilung der Attributwerte einer Relation möglich sind. Ziel der eingesetzten Histogrammtechnik ist die Datenverteilung so genau wie möglich widerzuspiegeln, um eine akkurate Abschätzung der Ergebnistupelanzahl zu erreichen. Somit kann während der Optimierungsphase durch die Bereitstellung der Informationen über die Selektivität der *where*-Bedingung eine effiziente Umformung der Anfragegraphen erfolgen.

Da das Erstellen von Statistiken ein zeitintensiver Prozeß ist, der das Einzulesen und Bewerten großer Datenmengen erfordert, müssen Einschränkungen bzgl. des Zeitpunktes und

des Umfanges der Statistikerstellung vorgenommen werden. Nachfolgend sind wesentliche Kriterien aufgeführt:

- die gesamte Tupelmenge der gegebenen Relation wird eingelesen und bewertet
- eine Teilmenge der Tupel der Relation wird untersucht
- Erstellung eines Histogrammes für den gesamten Wertebereich des betreffenden Attributes
- es werden nur Minimum -und Maximumwerte betrachtet (siehe Tab. 3)
- Beschränkung der Statistikerstellung auf die in den Qualifikationen verwendeten Attribute (*where*-Bedingung)
- Attribute in Join-Bedingungen
- Attribute mit definiertem Index bzw. Schlüsselattribute

Um eine exakte Näherung der Attributwertverteilung einer gegebenen Relation zu erreichen, wird eine Zerlegung des Attributwertbereiches in endlich viele aneinander grenzende Intervalle $\Delta_1,..,\Delta_k$ durchgeführt. Dann wird ermittelt, wieviele Attributwerte innerhalb der Intervalle ($i \leq j \leq k$) liegen. Über Δ_j wird ein Rechteck der Höhe m_j/n gezeichnet (relative Klassenhäufigkeiten). Das so entstehende Stufenbild (Histogramm) gibt somit Auskunft über die Verteilung der Attributwerte (siehe Abb. 9).

Die Effizienz der eingesetzten Verfahren, Histogrammtechnik bzw. Abschätzung der Zwischenergebnisrelationen unter Annahme der Gleichverteilung der Attributwerte muß in weiteren Arbeiten durch Vergleiche der Abschätzungsergebnisse mit Zeitmeßergebnissen an verschiedenen Anfragemustern untersucht werden.

4.4. Abschätzungsstrategie

Anhand des nachfolgenden vereinfachten Operatorgraphen unter Verwendung der für das Kostenmodell notwendigen Parameter wird die grundsätzliche Vorgehensweise der Kostenabschätzung eines Anfragegraphen dargestellt (siehe Abb. 10).

Die Knoten des Graphen repräsentieren die algebraischen Operationen ω_i auf den jeweiligen Rechnerknoten S_i.

Die Kanten werden durch die Zwischenergebnisrelationen ρ_i jeweils nach Ausführung einer algebraischen Operation markiert. Die Zwischenergebnisrelationen ρ_i sind wiederum Ausgangsgrößen für die nachfolgenden Operationen, gekennzeichnet durch Kardinalität und Tupelgröße der Relation ($card(\rho_i)$, $size(\rho_i)$).

Nach Ausführung der Projektions-Operationen ρ_1 und ρ_2 auf den Rechnerknoten S_1 und S_2 sind Datenübertragungen (Kommunikationskosten) zur Ausführung der Join-Operation auf dem Rechner S_3 erforderlich.

Auf Grund der Überführung der verschiedenen Kostenkomponenten des Kostenmodells in eine einheitliche Einheit (*ms*), ist eine Transformation der CPU-Leistung C_{cpu} in Millisekunden/Instruktion als auch der Netzwerkübertragungsgeschwindigkeit $C_{tr}(i, j)$ in Millisekunden/Byte, notwendig. Um zunächst vereinfachte Berechnungen durchführen zu können, wird als Richtwert die Übertragungsgeschwindigkeit des lokalen Netzes (IEEE 802.3) verwendet. Da diese Übertragungsgeschwindigkeit von 10 MBit/s für Ethernet ein Maximum darstellt, das unter realen Verhältnissen nur selten erreicht wird, müssen weiterführende Zeitmeßreihen durchgeführt werden, um eventuelle Korrekturen vornehmen zu

Beispiel:

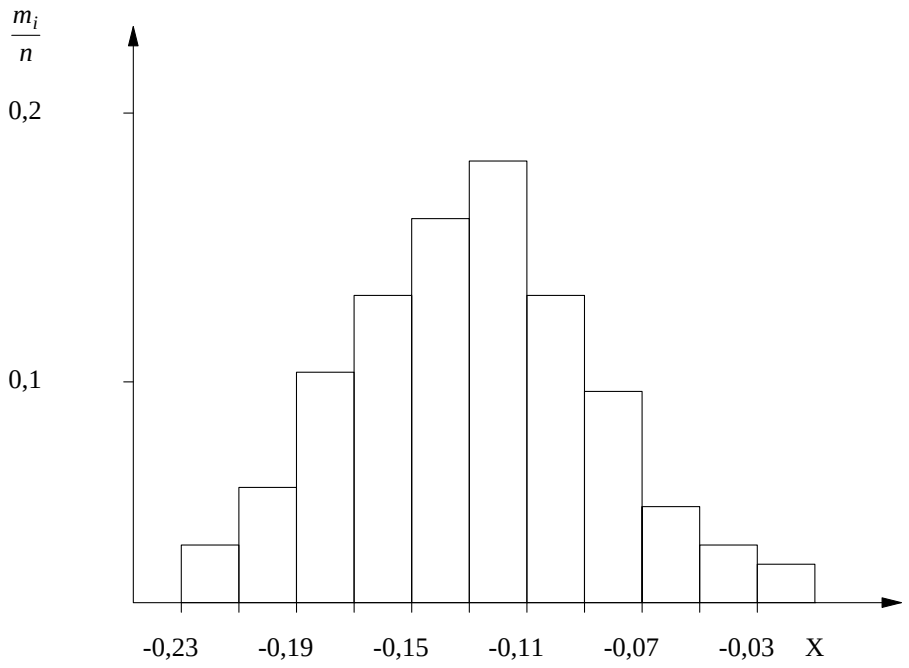

Abb. 9: Histogramm bzgl. eines Attributes vom Type *real*

können.

Die Aktualisierung der Metadaten nach dem Ausführen der jeweiligen relationalen Operation ist eine Annäherung an die Größe der realen Zwischenergebnistupelmenge. Folglich kann die berechnete lokale Verarbeitungszeit einer realen Operation, die auf diesen Ergebnisdatenmengen beruht, im ungünstigeren Fall in Richtung des Maximums bzw. Minimums abweichen. Die reale Bearbeitungszeit liegt demnach im folgenden Bereich:

Minimale Bearbeitungszeit < *reale Bearbeitungszeit* < *Maximale Bearbeitungszeit*

Da zum gegenwärtigen Zeitpunkt der Entwicklungsarbeit am HEAD-System die Berücksichtigung der vertikalen Parallelisierung durch das Formelsystem des Kostenmodells noch nicht möglich ist, wird das im nachfolgenden berechnete Abschätzungsergebnis jeweils in Richtung der maximalen Bearbeitungszeit abweichen.

5. Zusammenfassung und Ausblick

Es wurde die Arbeitsweise des Prototypen des verteilten, heterogenen Datenbanksystems HEAD dargestellt. Dabei wurde speziell auf die Optimierung eingegangen, dessen Kern die Kostenanalyse darstellt.

In zukünftigen Arbeiten werden parallele Approximationsverfahren untersucht werden, die den Zeitanteil der Optimierung bzgl. der Gesamtzeit zur Anfragebearbeitung reduzieren sollen. Des weiteren wird das Kostenmodell schrittweise ausgebaut, so daß eine umfassendere Bewertung der optimierbaren Bestandteile eines Anfrageplanes möglich wird.

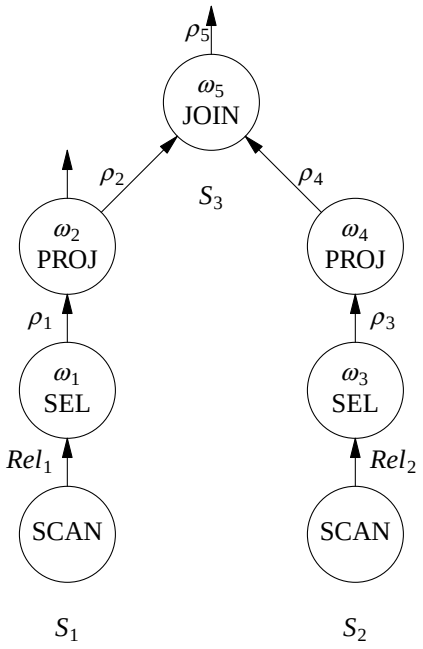

Abb. 10: Abbildung der Kostenmodellparameter auf einen Operatorgraphen

Außerdem gilt es, adaptive Regelmechanismen zu entwerfen und in HE$_A$D zu integrieren, die es dem System gestatten, sich optimal an geänderte Bedingungen anzupassen. Nicht zuletzt müssen Untersuchungen durchgeführt werden, wie und unter welchen Bedingungen der Multiprogramming level gezielt geändert werden kann.

Literaturverzeichnis

Ceri85.
Ceri, S. and Pelegatti, G., "Distributed Databases: Principles and Systems," *McGraw-Hill*, New York (1985).

Flac93.
Flach, G., "Konzeption eines Kostenmodells für ein verteiltes Datenbanksystem," *Diplomarbeit*, University of Rostock, Dept. of Computer Science, Database Research Group, Rostock (Mai 1993).

Grae90.
Graefe, G., "Encapsulation of Parallelism in the Volcano Query Processing System" in *Proceedings of the ACM SIGMOD Conference*, Atlantic City, NJ (May 1990).

Ingb89.
Ingber, L., "Adaptive Simulated Annealing (ASA)," [ftp.caltech.edu: /pub/ingber/ASA−shar.Z, ASA.tar.gz], Lester Ingber Research, McLean, VA (1993).

Lang93.
Langer, U. and Meyer, H., “Load Sharing and Distributing Relational Algebra Operations in the HEaD-Prototype” in *5th International Workshop on Foundations of Models and Languages for Data and Objects “Optimization in Databases”,* Aigen (Austria) (Sept. 1993).

Lanz91.
Rosana S. G. Lanzelotte and Patrick Valduriez, “Extending the Search Strategy in a Query Optimizer” in *Proceedings of the 17th VLDB Conference,* pp. 363-373, Barcelona, Spain (1991).

Link94.
Linke, A., “Lastmessungund -verteilung in heterogenen, verteilten Datenbanksystemen,” *Diplomarbeit*, Universität Rostock, Fachbereich Informatik (Mai 1994).

Mikk88.
Mikkilineni, K. P. and Su, S. Y. W., “An Evaluation of Relational Join Algorithms in a Pipelined Query Processing Environment,” *IEEE Trans. on Software Eng.,* 14, 6nd, pp. 838-848 (Jun. 1988).

Naka88.
Nakayama, M., Kitsuregawa, M., and Takagi, M., “Hash-Partitioned Join Method Using Dynamic Destaging Strategy” in *Proceedings of the 14th VLDB Conference,* pp. 468-478, Los Angeles, California (1988).

Ozsu91.
Özsu, M. T. and Valduriez, P., “Distributed database systems: Where Are We Now?,” *IEEE,* 24, 8, pp. 68-78 (Aug. 1991).

Pram88.
Pramanik, S. and Vineyard, D., “Optimizing Join Queries in Distributed Databases,” *IEEE Transaction on Software Engineering,* 14, 9 (September 1988).

Scha92.
Schabernack, J., “Lastausgleichsverfahren in verteilten Systemen - Überblick und Klassifikation,” *Informationstechnik,* 34, 5, pp. 280-295 (1992).

Adresse der Autoren:

Guntram Flach
Uwe Langer
Holger Meyer
Universität Rostock
Fachbereich Informatik
18051 Rostock

e-mail hme@informatik.uni-rostock.de