Mehrstufige Anfrageoptimierung in HEAD†

Uwe Langer‡
Holger Meyer

Universität Rostock
Fachbereich Informatik
Lehrstuhl Datenbank- und Informationssysteme
18051 Rostock

ABSTRACT

Am Beispiel verteilter Datenbanksysteme wird gezeigt, daß die Optimierung ein vielschichtiger Prozeß ist. Die Beschränkung auf eine Technologie wie z.B. einen regelbasierten Optimierer oder Tiefensuche erscheint nicht sinnvoll oder praktikabel. In HEAD wird davon ausgegangen, daß mehrere Optimierungsschritte aufeinander folgen können, die den Einsatz verschiedener Techniken erfordern.

Um zu untersuchen, inwieweit die einzelnen Optimierungsphasen unabhängig voneinander sind, wird in HEAD ein Werkzeugkasten von Optimiererbausteinen benutzt. Dieser Baukasten integriert existierende Techniken, z.B. Transformationsregelmengen, Finden gleicher Teilausdrücke (CSE), Kostenmodelle und verschiedene ein- und mehrdimensionale Suchalgorithmen (Adaptive simulated annealing, Genetic algorithms).

1. Einleitung

Im Rahmen des Projektes HEAD [FLM93] wird der Prototyp eines relationalen, verteilten Datenbanksystems (VDBS) implementiert. Hauptgegenstand der Untersuchungen sind Algorithmen zur optimal verteilten Bearbeitung globaler Anfragen im VDBS.

Bei der Zerlegung globaler Anfragepläne in lokal abarbeitbare Ausführungspläne werden folgende Teilprobleme bearbeitet:

- Normalisierung, Vereinfachung globaler Anfragen,
- Umformung in (fragmentierte) Anfragen über den Fragmenten einer Relation,
- Entfernung gleicher Teilpläne, nicht benötigter Fragmente,
- Bestimmung der optimalen Bearbeitungsreihenfolge der Teiloperationen,
- Auswahl entsprechender Implementierungen für die Operatoren,

† Die Arbeiten wurden teilweise durch die Deutsche Forschungsgemeinschaft (DFG) unter dem Aktenzeichen Me-1346/1-2 unterstützt.

‡ Die Autoren sind zu erreichen: `Vor.Nachname@Informatik.Uni-Rostock.DE`.

- Finden der optimalen Operator-Rechnerknoten-Zuordnung und
- Auswahl des optimal zu benutzenden Replikates.

Würde man diese Probleme systematisch lösen wollen, würde dies in einem mehrdimensionalen Optimierungsproblem mit entsprechender Komplexität münden. Eine mögliche Vorgehensweise wäre die Zergliederung in obige Teilprobleme, das Bestimmen der jeweiligen optimalen Teillösung und deren Kombination. Voraussetzung dafür wäre, daß sich die Teilprobleme nicht gegenseitig beeinflussen.

Eine weitere Voraussetzung für die systematische Lösung der Probleme ist, daß entsprechende Zielfunktionen, die die optimale Lösung beschreiben, angebbar sein müssen. Ist dieses nicht möglich, kann nur mittels Heuristiken eine Lösung gesucht werden.

In HEAD wird versucht, Teilprobleme zu isolieren, getrennt zu beschreiben, deren Korrelation zu untersuchen und als Ergebnis einen gut strukturierten, effizienten Optimierer zu erhalten, der in der Lage ist, in angemessener Zeit optimale Anfragepläne zu erzeugen.

Regelbasierte Anfrageoptimierung wurde bereits in [Lohm88, FG89, PHH92, DB94] untersucht, systematische Optimierung zuerst in System R [Cham81]. Für Anfrageumschreibung (Query rewriting) bzw. Pattern matching in Bäumen gibt es z.B. aus dem Compilerbau mit Werkzeugen wie TWIG [Tjia86] entsprechende Unterstützung. Speziell für die Implementierung von Datenbankoptimierern wurde VOLCANO Optgen [Grae90] geschaffen. Effiziente Suchalgorithmen für kostenbasierte Optimierer wurden in [Rose91, DLS92, IR92] betrachtet.

Im weiteren werden die Optimierungsprobleme in verteilten Datenbanksystemen dargestellt, und es wird gezeigt, welche Beschreibungsmittel und Bausteine HEAD für die Konstruktion des Optimierers bereitstellt, um verschiedene Konzepte zu integrieren. Auf Probleme der Implementierung und Stand der Arbeiten wird im Vortrag eingegangen.

2. Anfrageoptimierung in verteilten Datenbanksystemen

Anfragen im relationalen, verteilten Datenbanksystem HEAD werden bezüglich eines globalen konzeptuellen Schemas gestellt. SQL-Anfragen werden nach bekannten Techniken [CG85] in eine erweiterte relationale Algebra (AQUAREL) übersetzt.

```
ABT(NUM,NAME,LEITER)
PERS(NUM,NAME,ANSCHRIFT,ABT,GEHALT)

ABT := UNION(ABT.1, UNION(ABT.2, ABT.3));
ABT.1 := FRAG(ABT, NUM >= 1 AND NUM < 4);
ABT.2 := FRAG(ABT, NUM >= 4 AND NUM < 8);
ABT.3 := FRAG(ABT, NUM >= 8);
PERS := PERS.1;

ABT.1 := { ABT.1.1@ithaka, ABT.1.2@syros };
ABT.2 := { ABT.2.1@ithaka, ABT.2.2@syros };
ABT.3 := { ABT.3.1@marathon };
PERS.1 := { PERS.1.1@marathon };
```

Abb. 1: Beispielschema

Durch Anwendung der Kompositionsregeln aus dem Fragmentationsschema, die beschreiben, wie sich globale Relationen aus Fragmenten zusammensetzen, wird die globale Anfrage in eine bezüglich des Fragmentationsschemas übersetzt. Operanden sind jetzt nicht mehr globale Relationen, sondern Fragmente mit entsprechenden Informationen bezüglich der Fragmentation.

Bei horizontaler Fragmentation werden die einzelnen Fragmente über die Vereinigung zusammengesetzt und bilden so die globale Relation. Bei vertikalen Fragmenten erfolgt dies über den natür-

lichen Verbund. Durch das Umschreiben globaler in fragmentierte Anfragen werden also neue Operationen und Operanden eingefügt.

Die Optimierungen globaler bzw. fragmentierter Anfragen erfolgen durch die Anwendung von Heuristiken unter Benutztung von Transformationsregeln, die auf algebraische Äquivalenzen beruhen. Wie z.B. das Verschieben von Selektion und Projektion zu den Blattknoten des Algebrabaumes, um Zwischenergebnisrelationsgrößen zu minimieren. Benutzt werden dafür z.B. Idempotenz unärer Operatoren, Kommutativ- und Distributivgesetz (Bsp. 1).

Bsp. 1: Anfrageumformung

```
/* globale Anfrage:
 * Name des Leiters der Abteilung 1
 */
PROJ(
    SEL(
        JOIN(
            ABT,
            PERS,
            ABT.LEITER = PERS.NUM
        ),
        ABT.NUM = 1
    ),
    PERS.NAME
)

/* fragmentierte Anfrage */
PROJ(
    JOIN(
        PROJ(
            SEL(
                REN(ABT.1, ABT),
                ABT.NUM = 1
            ),
            ABT.LEITER
        ),
        PROJ(
            PERS,
            (PERS.NUM, PERS.NAME)
        ),
        ABT.LEITER = PERS.NUM
    ),
    PERS.NAME
)
```

Eine weitere Technik, um Mehrfachberechnungen zu vermeiden, ist die Eliminierung von gemeinsamen Teilausdrücken (Common subexpression elimination), dazu wird der Algebrabaum Buttom up nach gleichen Mustern durchsucht.

Fragmentierte Anfragen bieten weitere Möglichkeiten der Vereinfachung von Algebraausdrücken. Durch Hinzunahme der Fragmentationsinformation zu den Fragmenten erhält man qualifizierte Relationen, für die die bekannten Äquivalenzen weiterhin gelten und neue hinzufügbar sind. Diese neuen Äquivalenzen gestatten es, Teilausdrücke zu entfernen, die Operationen über nicht benötigten horizontalen Fragmenten enthalten (Bsp. 1).

Die bisher genannten Anfrageumformungen basieren auf Heuristiken, die besagen, daß bei äquivalenten Algebraausdrücken ein bestimmter Ausdruck zu bevorzugen ist. Um systematisch den besten Anfrageplan zu finden, müßten durch uneingeschränkte Anwendung aller angebbaren Transformationsregeln auf einen initialen Plan alle äquivalenten Pläne aufgezählt werden und aus diesen der kostenoptimale Plan ausgewählt werden. Dazu ist die Angabe eines Kostenmodells für alle Operationen der erweiterten Algebra notwendig.

Solch ein Kostenmodell basiert auf den von der Ausführung eines Operators erzeugten Kosten (verbrauchten Ressourcen) und der Abschätzung von Zwischenergebnissen, die wiederum Parameter der Operatorausführung bilden. Weitere Parameter sind z.B. die Wahl des Algorithmus zur Implementierung eines Operators, beim Verbund z.B. Hash join-Algorithmus, Merge join-Algorithmus etc. [Flac93]

Die Wahl von bestimmten Implementierungen kann an Eigenschaften der Zwischenergebnisse geknüpft sein. Zum Erzwingen dieser Eigenschaften werden wiederum neue Operationen in den Anfrageplan eingefügt, z.B. explizites Sortieren der Zwischenrelation vor einem

Merge join.†

Ein weiteres Problem in verteilten Datenbanksystemen ist die optimale Wahl von Replikaten, d.h. falls ein Fragment mehrfach auf unterschiedlichen Rechnerknoten gespeichert ist, jenes zu wählen, auf das am schnellsten zugegriffen werden kann.

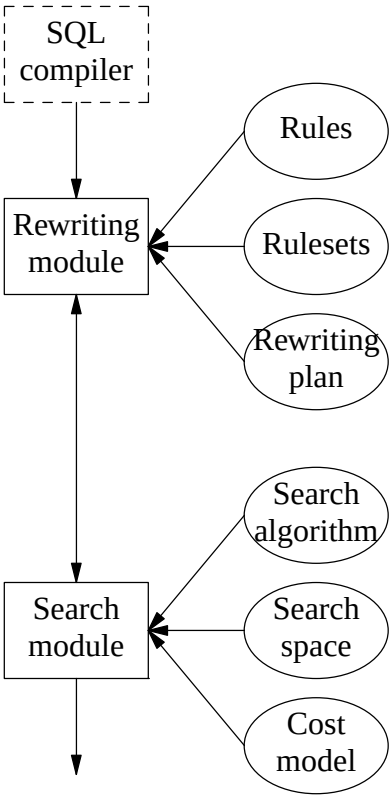

Abb. 2: HEAD-Optimierer

Hiermit verbunden ist weiterhin die Zuordnung von Teilanfragen und Operationen zu Rechnerknoten. Lösungen hierfür basieren auf einer kostenbasierten Optimierung. Der Einsatz von Suchverfahren für einen mehrdimensionalen Lösungsraum ist angebracht. Verfahren, die ohne vollständiges Absuchen auskommen, sind einsetztbar.

† Der Optimierergenerator aus VOLCANO z.B. benutzt hierfür sogenannte Enforcer.

3. Struktur des HEAD-Optimierers

Der Optimierer (Abb. 2) besteht aus zwei getrennten Modulen, die wahlweise oder zusammen eingesetzt werden können:

Bsp. 2: Regelmengendefinition

```
rules {
p1:
      PROJ(SEL(rel, expr), attrs) ->
            SEL(PROJ(rel, attrs), expr);
s1:
      {
            SEL(rel, expr AND expr) ->
                  SEL(SEL(rel, expr), expr);
            SEL(NJOIN(rel, rel), expr AND expr) ->
                  NJOIN(SEL(rel, expr), SEL(rel, expr));
      }
s2:
      SEL(SEL(rel, expr), expr) ->
            SEL(rel, expr AND expr);
q1:
      {
            SEL(FRAG(rel, exp1), exp2) ->
                  FRAG(SEL(rel, exp2), exp1 AND exp2);
      }
q2:
      {
            FRAG(rel, expr) -> rel;
            FRAG(rel, FALSE) -> NULL;
      }
      ...
}

rulesets {
r1:
      { p1, s1, j1, q1 }
r2:
      { p1, s2, j4, q2 }
}

rewrite {
      apply r1;
      apply r2;
}
```

- **Rewriting module**: mit einer eigenen Spezifikationssprache, die in Anlehung an reguläre Grammatiken entworfen wurde, werden die Algebra, Transformationsregeln und Regelmengen definiert. Transformationsregeln können zu Regelmengen zusammengefaßt werden. Die Reihenfolge der Anwendung von Regelmengen wird gesteuert.
- **Search module**: aus einer Menge äquivalenter Pläne wird über einen zu wählenden Suchalgorithmus der

kostenoptimale Plan gesucht. Funktionen zum Aufspannen des Lösungsraumes, der Generierung einer konkreten Lösung und der Ermittlung der Kosten für den gewählten Plan sind bereitzustellen.

Die Spezifikation (siehe Bsp. 2) wird z.Z. in C++ bzw. TWIG-Spezifikationen übersetzt. Die Prozeduren für das Search module sind mit entsprechenden C++-Methoden zu implementieren.

Im Suchmodul sind verschiedene Suchdimensionen angebbar, für die Abbildung des Lösungsraumes auf einen ganzzahligen Suchraum sind zwei Methoden zur Ermittlung der maximalen Anzahl der Lösungen und zur Generierung einer konkreten Lösung zu implementieren.

4. Stand der Arbeiten

Aus der Arbeit mit verschiedenen Optimiererwerkzeugen heraus enstand der Wunsch, diese unter einer einheitlichen Oberfläche zu integrieren. Die Optimiererspezifikation sollte deskriptiv und Änderungen leicht ausführbar sein. Insbesondere sollte die Verwendung verschiedener Transformationsregelmengen ermöglicht werden, heuristische und exakte Optimierung kombinierbar sein.

Momentan sind eine Reihe von Modulen für den Optimierer realisiert: Implementierung von Transformationsregeln mittels TWIG, CSE, Kostenmodell und Einbindung Suchalgorithmen. Für die mehrdimensionale Suche wird momentan ein Adaptive simulated annealing-Algorithmus [Ingb89] benutzt. Weitere Suchverfahren sollten leicht integrierbar sein.

Als Anwendung dient die Optimierung von verteilten Anfragen in der erweiterten relationalen Algebra AQUAREL und deren Parallelisierung für die verteilte Bearbeitung. Die Spezifikationssprache ist entworfen und wird implementiert.

References

CG85.
Ceri, S. and Gottlob, G., “Translating SQL into Relational Algebra: Optimization, Semantics and Equivalence of SQL Queries,” *IEEE Trans. S.E.*, pp. 324-245 (April 1985).

Cham81.
Chamberlin, D. D. and et al., “Support for repititive transactions and ad-hoc queries in System R,” *ACM ToDS*, 6, 1, pp. 70-94 (1981).

DB94.
Das, D. and Batory, D., “Prairie: A Rule Specification Framework for Query Optimizer,” TR94-16, Dept. of CS, Univ. of Texas, Austin.

DLS92.
Diekmann, R., Lulïng, R., and Simon, J., “Problem Independent Distributed Simulated Annealing and its Applications” in *Lecture Nodes in Economics and Mathematical Systems*, Springer (1992).

FLM93.
Flach, G., Langer, U., and Meyer, H., “Das Projekt HEaD,” *Forschungsbericht (DFG)*, Universität Rostock, Fachbereich Informatik, Arbeitsgruppe Datenbanken, Rostock (1993).

Flac93.
Flach, G., “Konzeption eines Kostenmodells für ein verteiltes Datenbanksystem,” *Diplomarbeit*, Universität Rostock, Fachbereich Informatik, Rostock (Mai 1993).

FG89.
Johann Ch. Freytag and Nathan Goodman, “On the Translation of Relational Queries into Iterative Programs,” *ACM ToDS*, 14, 1, pp. 1-27 (Mar. 1989).

Grae90.
Goetz Graefe, “Volcano, an Extensible

and Parallel Query Evaluation System," *Computer Science Technical Report*, 481, University of Colorado at Boulder, Boulder, Colorado (Jul. 1990).

Ingb89.
Ingber, L. and Rosen, B., "Very Fast Simulated Re-annealing," *Mathematical Computer Modelling,* 12, 8, pp. 967-973 (1989).

IR92.
Ingber, L and Rosen, B. E., "Genetic Algorithms and Very Fast Simulated Reannealing: A Comparison," *Mathematical Computer Modelling*, 16, 11, pp. 87-100 (1992).

Lohm88.
Lohman, G. M., "Grammar-like Functional Rules for Representing Query Optimization Alternatives," *ACM ToDS,* 13, 6, pp. 18-27 (1988).

PHH92.
Pirahesh, H., Hellerstein, J. M., and Hasan, W., "Extensible/Rule Based Query Rewrite Optimization in Starburst," *Proc. ACM SIGMOD*, pp. 39-48 (1992).

Rose91.
Rosen, B. E., "GA's and Very Fast Simulated Reannealing," *Genetic Algorithms Digest*, 5, 36 (1991).

Tjia86.
Tjiang, S. W. K., "Twig Reference Manual," Computing Science Technical Report #120, AT&T Bell Laboratories (1986).