Seminar: Data Streams

Multi-Query-Optimierung

Julia Thiele

j_thiele@informatik.uni-kl.de

Gliederung

- Multi-Query-Optimierung
- GAG-Repräsentation von Anfragen
- Verfahren der Multi-Query-Optimierung
  - Volcano
  - Volcano-SH
  - Volcono-RU
  - Greedy-Ansatz
  - NiagaraCQ
- Zusammenfassung
- Ausblick

Multi-Query-Optimierung

- Problematik:
  - Datenströme sind meistens unendlich
  - kontinuierliche Anfragen werden häufig ausgewertet

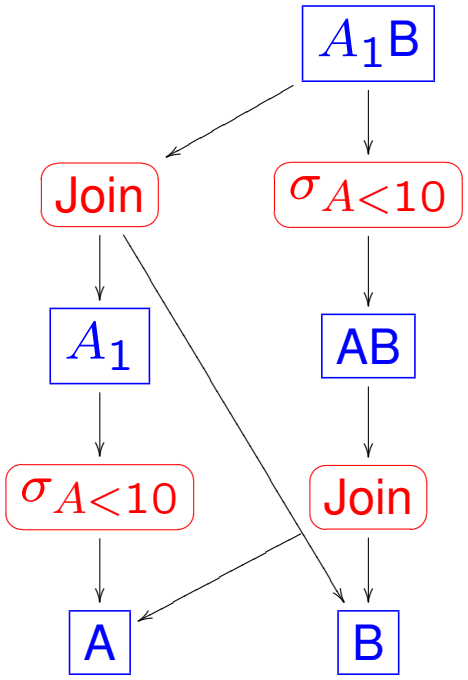
    $\Rightarrow$ effiziente Auswertung von Anfragemengen mit gleichen Unteranfragen

- Aufgaben:
  - Reduzierung der Auswertungskosten
  - Modifikation bzw. Verbinden der Anfragen zu einem global optimalen Auswertungsplan aller Anfragen

AND-OR-GAG-Repräsentation

- GAG ist ein gerichteter azyklischer Graph
- besitzt AND- und OR-Knoten
  - AND-Knoten entsprechen algebraischen Operationen (Join, Select) $\Rightarrow$ Operationsknoten
  - OR-Knoten entsprechen logischen Ausdrücken, die die gleiche Ergebnismenge erzeugen $\Rightarrow$ Gleichheitsknoten
- expandierter GAG:
  verdeutlicht alle Möglichkeiten zur Erzeugung eines Ausdrucks im Graphen
- Beispiel: $(\sigma_{A<10}(A))$ *Join B*
- wird in den Verfahren *Volcano* und *Greedy* verwendet

Expandierter GAG:



Volcano-Ansatz

- Ziel:

  Bestimmung eines besten Ausführungsplanes für Anfragen

- Vorgehensweise:
  - GAG mittels Tiefensuche durchlaufen
  - Kosten für jeden Knoten berechnen

    $\Rightarrow$ Ergebnis: bester Ausführungsplan für diese Anfrage

- Vorteil:

  Kostenreduktion durch das *branch and bound pruning*-Verfahren und durch Speicherung der bisher besuchten Knoten

- Verbesserungsmöglichkeiten
  - Volcano-SH $\Rightarrow$ z. B. $\sigma_{A<5}(A)$ *Join B* und $\sigma_{A<10}(A)$ *Join B*
  - Volcano-RU $\Rightarrow$ z. B. *(A Join B) Join C* und *A Join (B Join D)*

Erstellung eines Ausführungsplanes anhand eines Beispiels

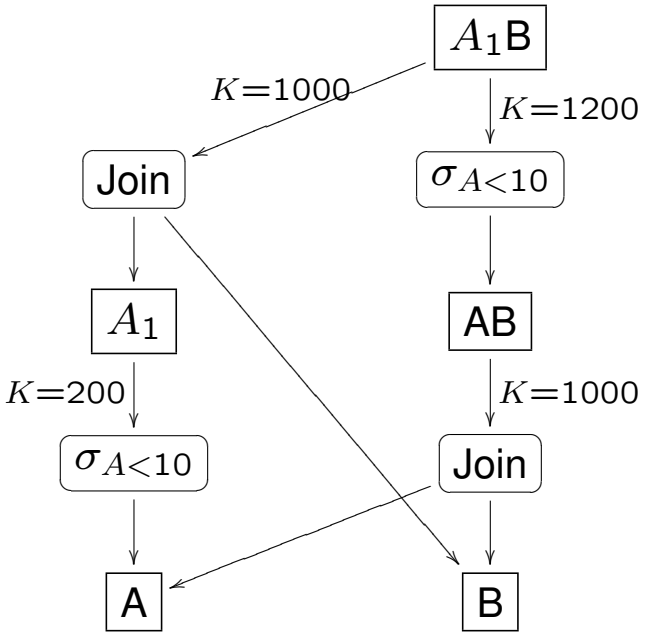
- Kosten eines Blattknotens: $cost = 0$
- Kosten eines Knotens o innerhalb des GAG:

  $cost(o) = Ausführungskosten + \sum_{e_i \in children(o)} C(e_i)$

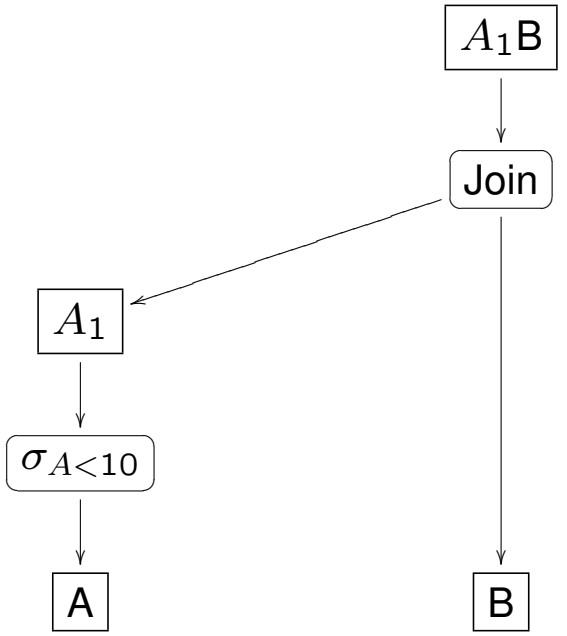
  mit $C(e_i) = cost(e_i)$ wenn $e_i \notin M$

  oder $C(e_i) = min\{cost(e_i), reusecost(e_i)\}$ wenn $e_i \in M$

Expandierter GAG



Bester Ausführungsplan



Volcano-SH

- Basiert auf bestem Ausführungsplan des Volcano-Algorithmus
- Ziel:

  Materialisierungsentscheidung für Teilausdrücke zur Kostenreduktion
- Vorgehensweise:
  - Zur Vereinigung der Anfragegraphen ist eine virtuelle Wurzel notwendig
  - GAG bottom-up betrachten
  - Materialisierungsentscheidung aufgrund der Anzahl der Vorfahrenknoten im GAG
- Nachteil:

  frühere Materialisierungsentscheidungen werden in die Ausführungskosten mit einbezogen
- Folgende Beispiele basieren auf diesen Anfragen:

  A Join B
  $\sigma_{A<10}(A)$ *Join B*
  $\sigma_{A<5}(A)$ *Join B*

Ausführung des Volcano-SH-Algorithmus

Procedure VOLCANO-SH(P)

$M = \{\}$

Perform prepass on P to introduce subsumption derivations

Let $C_{root} = COMPUTEMATSET(root)$

Set $C_{root} = C_{root} + \sum_{d \in M}(cost(d) + matcost(d))$

Undo all subsumption derivations on P where the subsumption node is not chosen to be materialized.

return(M,P)

Procedure COMPUTEMATSET(e)

if cost(e) is already memorized, return cost(e)

Let operator o_e be the child of $e \in P$

For each input equivalence node e_i of o_e

Let $C_i = COMPUTEMATSET(e_i)$

if e_i is materialized, let $C_i = reusecost(e_i)$

Compute $cost(e)$ = cost of operation $o_e + \sum_i C_i$

if $(matcost(e)/(numuses^-(e) - 1) + reusecost(e) < cost(e))$

if (e is not introduced by a subsumption derivation)

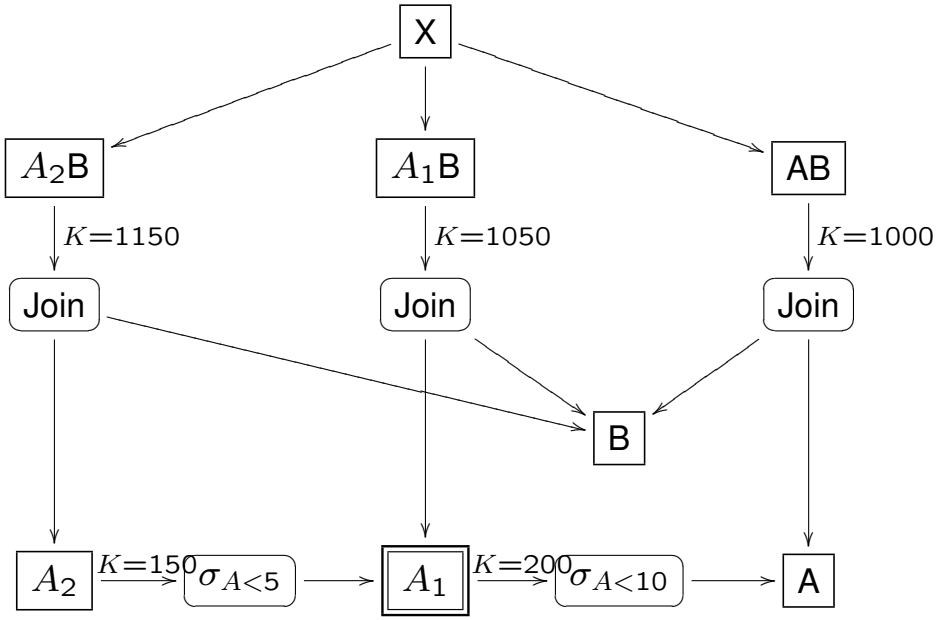
add e to M //Decide to materialize e

else if $cost(e) + matcost(e) + reusecost(e) * (numuses^-(e) - 1)$ is less than savings to parents of e due to introducing materialized e

add e to M //Decide to materialize e

Memorize and return cost(e)

Der Beispielgraph:



Volcano-RU

- Grundidee:

  Zusammenfassung solcher Teilausdrücke, die scheinbar nicht zusammengefasst werden können

  z. B. *(A Join B) Join C* und *A Join (B Join D)* $\Rightarrow$ *A Join B*

- Vorgehensweise:
  - 1.Phase: Festlegung einer Knotenkandidatenmenge zur Materialiserung
  - 2.Phase: Ausführung des *Volcano-SH-Algorithmus*

- Anfragenreihenfolge relevant

  $\longrightarrow$ je nach Reihenfolge resultieren verschiedene Ergebnismengen

  $\Rightarrow$ betrachtet wird die Eingabereihenfolge und deren Umkehrung

  $\Rightarrow$ die Variante mit den geringeren Kosten wird verwendet

Ausführung des Volcano-RU-Algorithmus

Procedure VOLCANO-RU

Input: Expanded DAG on queries $Q_1 \ldots Q_k$(including subsumptions derivations)
Output: Set of nodes to materialize M, and the corresponding best plan P

$N = \{\}$ //Set of potentially materialized nodes
for each equivalence node e, set count[e]=0
for $i = 1$ to k

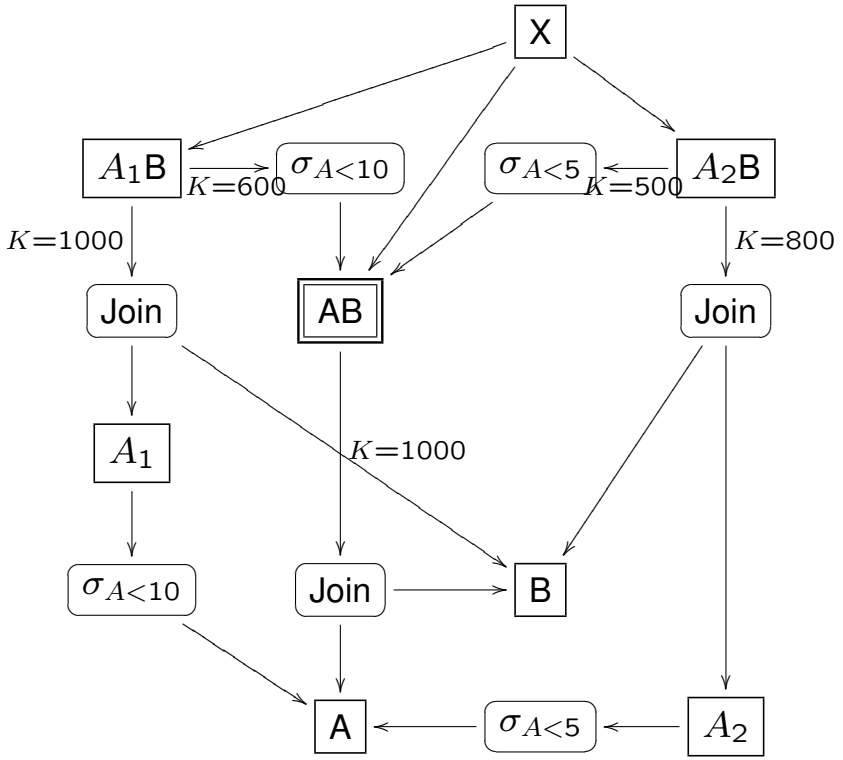
Compute P_i, the best plan for Q_i, using the Volcano, assuming nodes in N are materialized for every equivalence node in P_i

set count[e] = count[e] + 1
if (cost(e) + matcost(e) + count[e] * reusecost(e) $<$ (count[e] + 1) * cost(e))
//Worth materializing if used once more

add e to set N

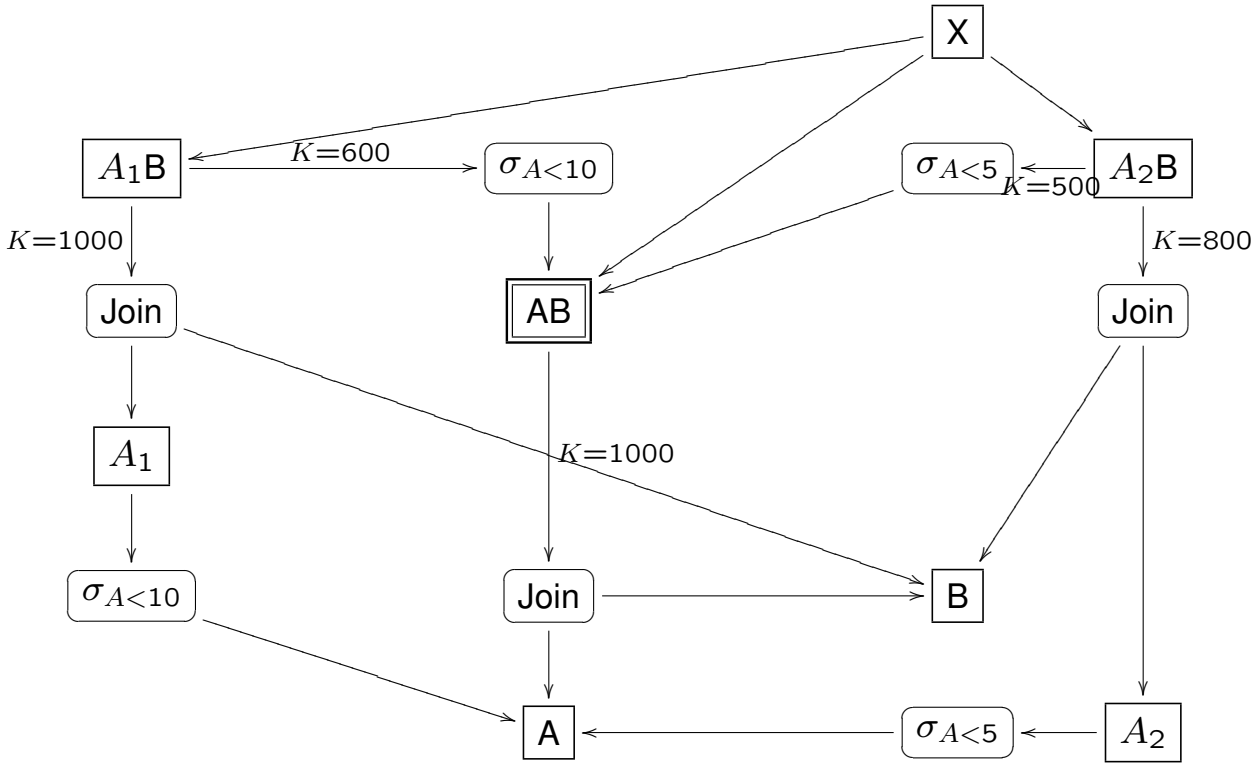
Combine $P_1 \ldots P_k$ to get a single DAG-structured plan P
//Volcano-SH makes final materialization decision
(M,P) = VOLCANO-SH

Der Beispielgraph:

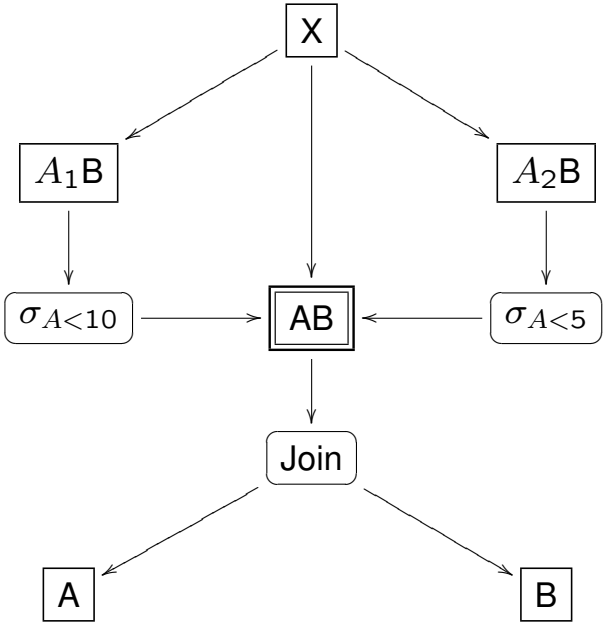


Ergebnisgraph des Volcano-RU-Algorithmus

Expandierter GAG mit Kosten



GAG − RU



Greedy-Ansatz

- *gierige* Herangehensweise an die Problematik:

  $\Rightarrow$ in jedem Teilschritt wird der Folgeschritt mit dem höchstem Gewinn ausgewählt

- basiert auch auf der GAG-Repräsentation der Anfragen
- eine mögliche Realisierung ist:

Procedure GREEDY

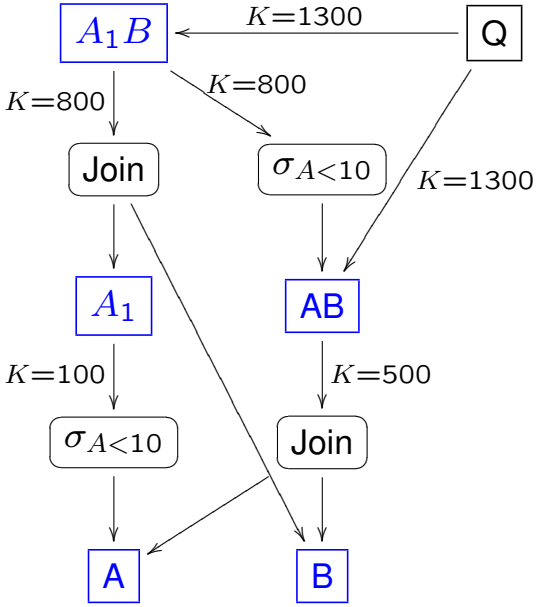
Input: Expanded DAG for the consolidated input query Q
Output: Set of nodes to materialize and the corresponding best plan
$X = \{\}$
Y = set of equivalence nodes in the DAG
while (Y! $= \{\}$)

L1: Pick the node $x \in Y$ with the smallest value for $bestcost(Q, x \cup X)$
if ($bestcost(Q, x \cup X) < bestcost(Q, X)$)

$Y = Y - x$; $X = X \cup x$

else $Y = \{\}$ //benefit<0, so break out of loop

return X



Optimierungsmöglichkeiten des Greedy-Ansatzes

- Nachteil:

  Pick the node $x \in Y$ with the smallest value for $bestcost(Q, x \cup X)$

  $\Rightarrow$ wenn Menge Y sehr groß

  $\Rightarrow$ Funktion $bestcost$ wird sehr oft ausgeführt

- Optimierungsmöglichkeiten:
  - *Teilbare Knoten* zur Materialisierung definieren.

    $\Rightarrow$ Menge Y mit diesen teilbaren Knoten initialisieren.
  - Einführung eines Algorithmus, der die Kosten inkrementell aktualisiert

    $\Rightarrow$ Funktion $bestcost$ muss nicht so oft aufgerufen werden
  - Mittels *Monotonie-Heuristik* die Ausführungsanzahl der Funktion $bestcost$ verringern. Hierbei wird verhindert, dass die Funktion für jedes $x \in Y$ ausgeführt wird.

Vergleich Volcano - Greedy

- Durchführung der Algorithmen anhand von TPCD-Anfragen vom Typ BQ5 auf einer TPCD-Datenbank

  $\Rightarrow$ Der Greedy-Ansatz bearbeitet den Ausführungsplan bei einer Benchmarkausführung

  etwa 9 Stunden schneller als der Volcano-Algorithmus

Projekt Niagara

- Ziel:
  - Verarbeitung verteilter Anfragen anhand von XML-Datensätzen in verteilten DBMS
  - Möglichkeit mittels einer *High-Level-Anfragesprache* Anfragen erstellen zu können

- Annahme:

  große Ähnlichkeit der Anfragen $\Rightarrow$ Gruppierungsstrategie möglich

- Vorteile:
  - Auswertungsergebnisse können gemeinsam verwendet werden
  - I/O-Kostenreduktion aufgrund Speicherung gemeinsamer Ausführungspläne und gemeinsame Ausführung
  - geringerer Aufwand bei der Überprüfung der Anfragebedingungen

- *NiagaraCQ* ist die Realisierung innerhalb dieses Projekts

NiagaraCQ-Ansatz

Dieser Ansatz basiert auf:

- einer Ausdruckssignatur

  $\longrightarrow$ dient zur Gruppierung von Anfragen gleicher syntaktischer Struktur

- Gruppeneigenschaften

- und einer Gruppenoptimierungsstrategie.

Diese Grundlagen werden auf den folgenden Folien näher erläutert.

NiagaraCQ - Beispiel zur Ausdruckssignatur

XML-QL-Anfragbeispiel:

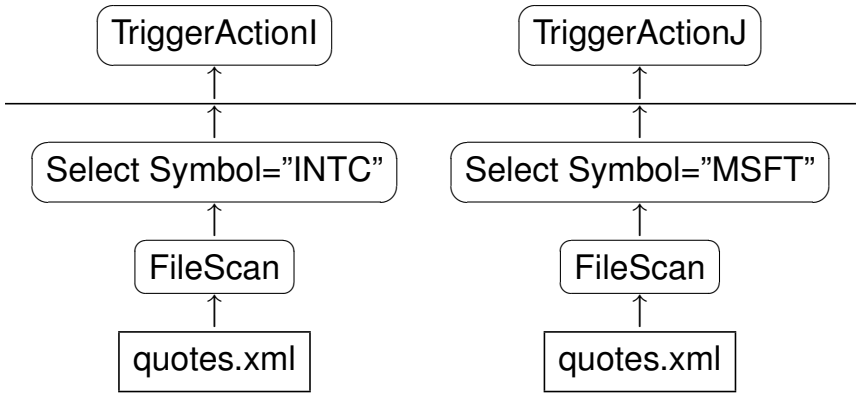
```
Where <Quote><Quote><Symbol>INTC</>
</></>element_as $g
in "http://www.cs.wisc.edu/db/quotes.xml"
construct $g
```

Entsprechende Ausdruckssignatur der beiden Anfragen:

```
Quotes.Quote.Symbol in quotes.xml = constant
```

Weitere Vorgehensweise:

- unterer Teil des generierten Anfrageplans ergibt die Ausdruckssignatur
- neuer Operaton *Trigger-Action* am Kopf der Signatur hinzufügen



NiagaraCQ - Gruppeneigenschaften

- Gruppenbestandteile:
  - Gruppensignatur
    $\longrightarrow$ gemeinsamer Teil der Ausdruckssignatur aller Anfragen einer Gruppe
  - konstante Gruppentabelle
    $\longrightarrow$ Speicherung der Signaturen aller Anfragen in einer XML-Datei
  - Gruppenplan
    $\longrightarrow$ leitet sich aus den gemeinsamen Teilanfragegraphen ab

- Das Ergebnis enthält alle möglichen Werte der Anfragekonstanten
  $\Rightarrow$ Filterung des Ergebnisses

NiagaraCQ - Gruppenoptimierungsstrategie

- Gruppierung aufgrund der Ausdruckssignaturen
- Vorgehensweise:
  - Gruppenoptimierer durchläuft den Anfrageplan bottom-up
  - Gruppenzuweisung aufgrund der Ausdruckssignatur
    - wurde eine Gruppe gefunden
      $\Rightarrow$ Anfrageplan in 2 Teile teilen: der obere Teil wird dem Gruppenplan hinzugefügt
    - wurde keine Gruppe gefunden
      $\Rightarrow$ neue Gruppe mit dieser Gruppensignatur erzeugen und in die Gruppentabelle aufnehmen
- Nachteil:

  Ergebnisverschlechterung durch dynamisches Hinzufügen und Löschen von Anfragen
  $\Rightarrow$ dynamisches Umgruppieren wäre notwendig

Gruppenstrategie vs. Nichtgruppierung

- Parameter zur Experimentausführung:
  - N Anzahl der installierten Anfragen
  - F Anzahl der ausgeführten Anfragen innerhalb einer Gruppe
  - C Anzahl der geänderten Tupel
  - T Ausführungszeit $\Rightarrow T = T_g + \sum_F T_{ng}$

- 1.Versuch: $C = 1000$ und $F = N$ und nur ein Anfragetyp
  $\Rightarrow$ keine Gruppierung: wenn N wächst, steigt T drastisch
  $\Rightarrow$ Gruppierung: benötigt bedeutend weniger Ausführungszeit

- 2. Versuch: $F = N = 2000\ Anfragen$ und nur ein Anfragetyp
  $\Rightarrow$ keine Gruppierung: T steigt an, sobald C ansteigt
  $\Rightarrow$ Gruppierung: T verhält sich proportional zu C

- Anfragetypkombination nur möglich mit gleicher Ausdruckssignatur

Zusammenfassung

- Im Bereich Multi-Query-Optimierung gibt es schon einige gute Ansätze
  - Materialisierung von gemeinsamen Teilausdrücken
  - Gruppenzuordnungsstrategie
- Einige wurden hier vorgestellt:
  - Volcano mit den Optimierungen Volcano-SH und Volcano-RU
  - Greedy mit einigen Optimierungsvarianten
  - Projekt Niagara mit Realisierung NiagaraCQ

Ausblick

- Da die Bedeutung der Datenströme mit der Zeit zunimmt, ist auch deren Verarbeitung in der Zukunft relevant
- Es sind bestimmt noch längst nicht alle Möglichkeiten ausgeschöpft, so dass hier weiter geforscht werden kann
- Beispielsweise besteht die Möglichkeit, eine eigene Datenstrom-Algebra zu entwickeln - aber ob dies notwendig wird, ist noch fraglich

Vielen Dank für ihre Aufmerksamkeit!

Fragen?