Kostenmodelle

Ziel: Geg. zwei Millionen Pläne; bestimme den Besten.

- Sehr lästiges Thema: Kostenmodelle sind sehr aufwendig zu entwickeln und zu implementieren/debuggen, und dann sind sie trotzdem falsch.
- Was modelliert z.B. die folgende Formel?

$$d_*^{GHJ}(m,\alpha,\beta,N,n,F) = rd\left(\frac{\alpha F(n\alpha+n\beta+Nm)+\sqrt{\alpha\beta mNF(n\alpha+n\beta+Nm)}}{m(n\alpha+n\beta+Nm)}\right)$$

- Aber jedes System braucht ein Kostenmodell!!!
- Wieso müßt Ihr das lernen?

  ... Ich weiß es nicht.

Übersicht

Es gibt 2 Arten von Kostenmodellen:

1. Klassisches Kostenmodell (K_1):
   Modelliert den Verbrauch an Ressourcen
   (CPU, Disk IO, Netzwerk)

   $\Rightarrow$ gut für hohen Durchsatz

2. Antwortzeitmodell (K_2):
   Berücksichtigt zusätzlich noch Parallelität.

   $\Rightarrow$ gut zur Optimierung von Anfragen in einem nicht ausgelastetem System

Bemerkung 1:

Die beiden Modelle sind oft widersprüchlich.
D.h. $K_1(A) > K_1(B)$ und $K_2(A) < K_2(B)$
Denkt an das Glühbirnenbeispiel!
Übung: Denkt euch ein eigenes VDBMS Beispiel aus.

Bemerkung 2:

Fast alle Systeme benutzen K_1.

Klassisches Kostenmodell

Literatur: Mackert, Lohman: VLDB'86

$$\begin{aligned}\text{Kosten} = c_1 * \text{CPU-Kosten} + c_2 * \text{IO-Kosten} \\ + c_3 * \text{Netzwerk-Kosten}\end{aligned}$$

Bewertung eines gesamten Planes:

$$\begin{aligned}\text{CPU-Kosten} = \quad & \text{CPU}(O_1) + \text{CPU}(O_2) + \text{CPU}(O_3) \\ & + \text{CPU}(O_4) \\ \text{IO-Kosten} = \quad & \text{IO}(O_1) + \text{IO}(O_2) + \text{IO}(O_3) + \text{IO}(O_4) \\ \text{Netzwerk-Kosten} = \quad & \text{Netz}(O_1) + \text{Netz}(O_2) + \text{Netz}(O_3) \\ & + \text{Netz}(O_4)\end{aligned}$$

CPU Kosten

Beispiel: Externes Sortieren

Lehrbuch zu internem Sortieren: $\mathcal{O}\ (n * \log n)$

In Datenbanken:

- $m \ll n$ Tupel passen in den Hauptspeicher
  $\Rightarrow$ sortieren in $\lceil \frac{n}{m} \rceil$ Läufen

  $$\boxed{\text{CPU-Kosten} = \lceil \tfrac{n}{m} \rceil * 70 * m * \log m}$$

  (70 ist Pfadlänge des Vergleichs – hier willkürlich gewählt, i.a. implementierungsabhängig. $\mathcal{O}$ reicht nicht.)

- Hinzu kommen CPU Kosten fürs *Mergen* der Läufe

  $$\boxed{30 * \lceil \tfrac{n}{m} \rceil * n}$$

  (30 ist Pfadlänge des Mergens – wiederum willkürlich gewählt.)

- Hinzu kommen CPU Kosten zum Lesen und Schreiben auf Festplatte.
- Hinzu kommen ggf. noch CPU Kosten zum Verschicken und Empfangen von Nachrichten.

IO Kosten

Willkürlich aus Steinbrunn et al. VLDB Journal abgeschrieben.
(der es widerum aus Shapiro-TODS'86 abgeschrieben hat.)

IO Kosten für einen Hybrid-Hash Join:

$$b_{R_1} + b_{R_2} + 2 * (b_{R_1} + b_{R_2}) * (1 - q)$$

$$q = \frac{ms - \left\lceil \frac{1.4 * b_{R_1} - ms}{ms - 1} \right\rceil}{b_{R_1}}$$

In dem Paper gibt es noch zig andere Formeln.

FAZIT: NICHT AUSWENDIG LERNEN ODER GAR SELBER MACHEN!

Sondern wissen, wo es steht.

IO Kosten, Nachschlag

Beachte: Unterschied zwischen *sequential* und *random* IO. (Macht auf einer durchschnittlichen Festplatte ungefähr 3,5ms im Vergleich zu 11,8ms aus.)

- geclusterter vs. nicht geclusterter Index
- Join Methoden sehr unterschiedlich in dieser Hinsicht

FAZIT bleibt: Wissen wo es steht.

(Join Costs Revisited, VLDB Journal, Januar 1996)

Netzwerkkosten

- Latenzzeit (nur wichtig für Antwortzeitmodell)
- verbrauchte Bandbreite:

$$\boxed{\text{Netwerk-Kosten} = \frac{\text{Volumen}}{\text{Bandbreite}}}$$

- Beachte. Bandbreite und Latenzzeiten zwischen einzelnen Knoten können stark unterschiedlich sein. (Z.B. Passau-München-New York.)
- Beachte. Im Internet ist die Bandbreite riesig. Aber sie wird auch fast komplett verbraucht, so daß in obige Formel realistischerweise "effektive Bandbreite" gehört; i.e., Bandbreite, die man im Durchschnitt bei einer Anfrage erwarten darf.

Schätzen der Größe von Zwischenergebnissen

Essentiell, da dies Variable für alle Kostenformeln ist.

```
SELECT *
FROM   Teile t, Lieferant l
WHERE  t.Preis > 500
       AND t.LiefNr = l.LiefNr
       AND l.Name = Strunz
```

Laut Histogramm: vermutlich 30 Teile mit Preis > 500.

(Literatur: Poosala, Ioannidis, Haas: SIGMOD'96)

Wieviel Teile mit Preis > 500 werden denn dann von Strunz geliefert?

Beachte Korrelation von Prädikaten. Lösung: Multidimensionale Histogramme (Literatur: Poosala und Ioannidis: VLDB'97)

Kalibrierung eines Kostenmodells

Formel (Wiederholung):

$$\text{Kosten} = \underline{c_c} * \text{CPU} + \underline{c_s} * \text{SEQ-IO} + \underline{c_r} * \text{RAND-IO} + \underline{c_n} * \text{Netz}$$

Frage: Wie sehen c_c, c_s, c_r, c_n aus, wenn Hardware der einzelnen Knoten unterschiedlich und das Netzwerk nicht homogen ist. (Fürs Netz haben wir das ja schon angesprochen)

Antwort: c_c, c_s, c_r sind Vektoren. Die Komponenten werden für jeden Knoten separat ermittelt. c_n ist eine Matrix. Die Komponenten werden für jede Punkt-zu-Punkt-Verbindung separat ermittelt. (Beachte. Die Matrix muß nicht unbedingt symmetrisch sein – Beispiel: Radio oder Fernsehen.)

Antwortzeit-Modell

Literatur: Ganguly et al. SIGMOD'92

Motivation:

Frage: Beträgt die Antwortzeit wirklich 150 Sekunden?

Antwort: Nein!

Begründung:

- op2 und op3 laufen *unabhängig* parallel
- op1, op2 und op1, op3 laufen *pipelined* parallel

Funktionsweise des Modells

1. Bestimme Antwortzeit jedes Operators für sein erstes Ergebnistupel.
   (Das wird auch als *Materialized Front* bezeichnet.)
2. Bestimme Antwortzeit jedes Operators für alle seine Ergebnistupel.
3. Verknüpfe bottom-up Antwortzeiten der einzelnen Operatoren eines Planes unter Berücksichtigung von *unabhängiger* und *pipeline* Parallelität.

In 1. und 2. werden im wesentlichen dieselben Formeln für Joins, Sortierung, Netzwerk usw. wie beim klassischen Modell verwendet.

Antwortzeit eines gesamten Planes
Alle Ressourcen haben ∞ Bandbreite

Notationen und Definitionen: (seien t_1, t_2 integer Werte)

$t_1 \| t_2 := max(t_1, t_2)$ unabhängige Parallelität

$t_1; t_2 := t_1 + t_2$ sequentielle Ausführung

$t_1 \ominus t_2 = t_1 - t_2$ Pipeling;

Berechnung der Antwortzeit einer Pipeline:

$p : (p_f, p_l)$ Produzent mit p_f, p_l Antwortzeit für das erste bzw. alle Ergebnistupel.

$c : (c_f, c_l)$ Konsument mit entsprechenden Antwortzeiten

Die Antwortzeiten von $t = p \mid c$ werden dann wie folgt berechnet:

Die *materialized front* wird sequentiell ausgeführt:

$$t_f = (p_f; c_f)$$

Der Rest wird unabhängig parallel ausgeführt:

$$t_l = (p_f; c_f; ((p_l \ominus p_f) \| (c_l \ominus c_f))$$

Berechnung der Antwortzeit einer Pipeline mit 2 Produzenten (L, R):

Die *materialized fronts* von L und R werden unabhängig parallel ausgeführt.

$$t_1 = (L_f \| R_f, L_f \| R_f)$$

Der Rest von L und R bilden eine Pipe, um die komplette Eingabe des Konsumenten zu liefern:

$$t_2 = t_1; (0, L_l \ominus L_f) \mid (0, R_l \ominus R_f)$$

(Interpretiere das als vereinige die beiden Ströme Operator, der durch t_2 beschrieben wird.)

Dieser Strom wird dann in den Konsumenten gepiped:

$$\boxed{t = t_2 | c}$$

In unserem Beispiel kommt heraus: (40, 70)

Antwortzeit eines gesamten Planes
Ressourcen haben beschränkte Bandbreite

Idee: Protokolliere Ressourcenverbrauch mit. Anstatt einfach nur Antwortzeit wird für jeden Operator noch in einem Ressourcevektor festgehalten, wieviel er von jeder Ressource verbraucht hat:

$\vec{r} = (\text{CPU}_{NY}, \text{CPU}_{DC}, \text{CPU}_{PA}, \text{Netz})$

Nun überlade $\|$, ;, $\ominus$ so daß sie komponentenweise auf die Ressourcevektoren $\vec{r}$ und Antwortzeiten angewendet werden können:

Sequentielle Ausführung:

$$t_1; t_2 = t_1 + t_2$$
$$\vec{r_1}; \vec{r_2} = \vec{r_1} + \vec{r_2}$$

Pipeline Ausführung:

$$t_1 \ominus t_2 = t_1 - t_2$$
$$\vec{r_1} \ominus \vec{r_2} = \vec{r_2} - \vec{r_1}$$

Unabhängig parallele Ausführung:

$$t_1 \| t_2 = max(t_1, t_2, max_i(r_1^i + r_2^i))$$
$$\vec{r_1} \| \vec{r_2} = \vec{r_1} + \vec{r_2}$$

Mit diesen Formeln ist Schema von vorher anwendbar.

In unserem Beispiel kommt heraus: (40, 80)