Kostenmodelle

Kostenmodelle

- Wünschenswert wäre es, den jeweils optimalsten Auswertungsplan zu finden. Dazu müssten aber alle denkbaren Pläne generiert und überprüft werden → nicht möglich
- Heuristische Optimierungen sollen *in den meisten Fällen* in kurzer Zeit gute Ergebnisse liefern
- Kostenmodelle helfen beim Vergleich von Auswertungsplänen, indem sie den Aufwand (Laufzeit) der Operatoren abschätzen. Sie verwenden dazu Parameter wie Indices, Ballungen, Kardinalitäten, Verteilungen, ...

Selektivität

Selektivität bestimmt den relativen Anteil der Tupel, die ein Selektionskriterium p erfüllen.

Selektion mit Bedingung p:

$$sel_p := \frac{|\sigma_p(R)|}{|R|}$$

Selektivität eines Joins von R mit S:

$$sel_{RS} := \frac{|R \bowtie S|}{|R \times S|} = \frac{|R \bowtie S|}{|R| \cdot |S|}$$

Selektivität abschätzen – einfache Fälle

- Selektivität auf einen Schlüssel ($\sigma_{R.A=c}$):

$$sel_{R.A=C} := \frac{1}{|R|}$$

- Bei einer Gleichverteilung der Attributwerte von R.A auf *i* verschiedene Werte:

$$sel_{R.A=C} := \frac{1}{i}$$

- Equijoin, wobei R.A Schlüssel ist und S.B der Fremschlüssel:

$$sel_{R \bowtie_{R.A=S.B} S} := \frac{1}{|R|}$$

Selektivität abschätzen – Verfahren

Parametrisierte Verteilungen

Es wird versucht, die Parameter einer Funktion so zu bestimmen, dass diese die Verteilung möglichst gut annähert.

Durch Aufruf dieser Funktion wird dann die erwartete Anzahl an Tupel ermittelt.

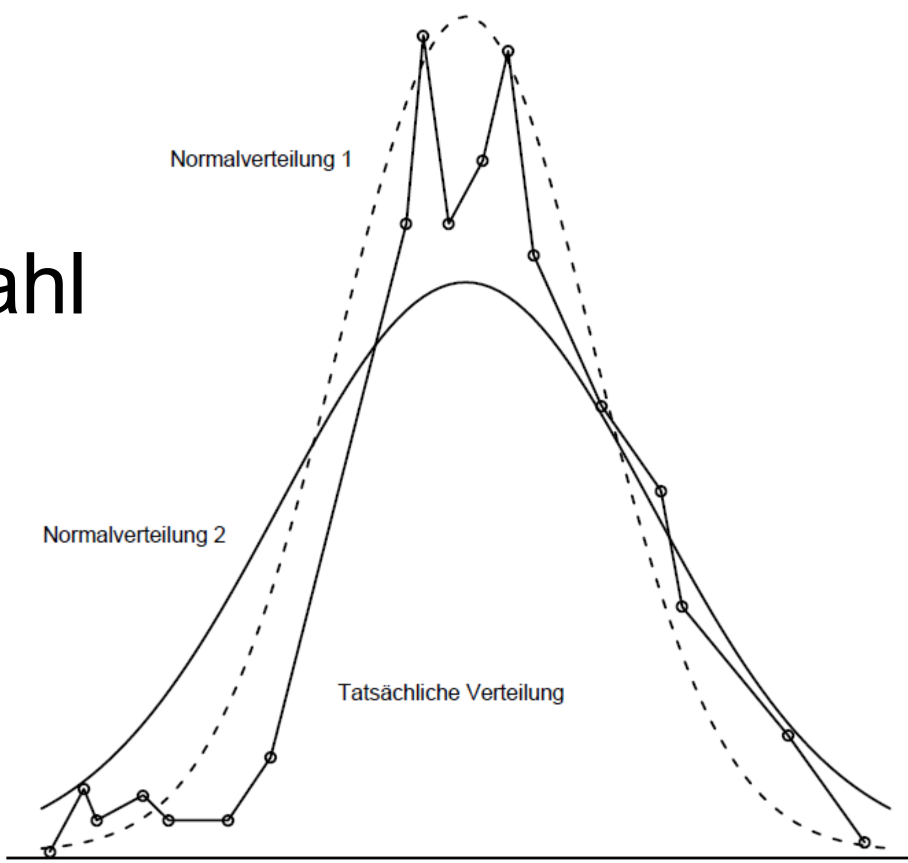

Bild: Reinhard Pichler, Vorlesungsunterlagen Datenbanksysteme, Uni Wien (DBAI), http://www.dbai.tuwien.ac.at/education/dbs/current/folien/Kapitel8c.pdf

Selektivität abschätzen – Verfahren

Histogramm

Der Wertebereich eines Attributs wird in Teilbereiche unterteilt und die relative Häufigkeit dieser Teilbereiche ermittelt.

Flexible Annäher der Verteilung möglich.

Equi-width (Intervalle gleich groß)

Equi-depth (in jedem Intervall gleich viele Werte, aber unterschiedlich breit)

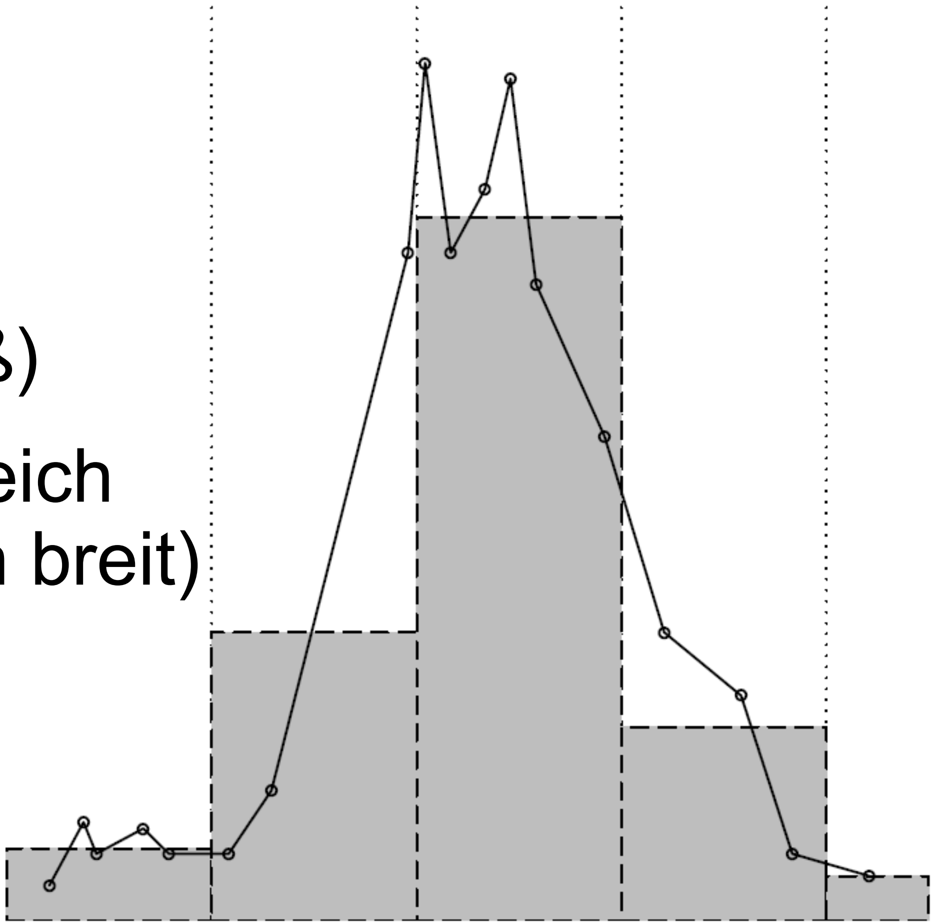

Bild: Reinhard Pichler, Vorlesungsunterlagen Datenbanksysteme, Uni Wien (DBAI), http://www.dbai.tuwien.ac.at/education/dbs/current/folien/Kapitel8c.pdf

Selektivität abschätzen – Verfahren

Stichproben

Sehr einfach zu implementieren.
Es wird eine zufällige Menge von Tupeln einer Relation analysiert und deren Verteilung als repräsentativ für die ganze Relation angesehen.

→ Teure Zugriffe auf Hintergrundspeicher notwendig

Kostenabschätzung

- Größter Kostenfaktor ist Hintergrundspeicherzugriff
- CPU-Aufwand im Verhältnis meist gering

Notation:

- m: Anzahl der Seitenrahmen im DB-Puffer
- b_R, b_S: Anzahl der Seiten (im Hintergrundspeicher) für die Relation R bzw. S
- M_R, M_S: Anzahl der Tupel von Relation R bzw. S
- p_R, p_S: Anzahl der Tupel pro Seite

Kostenabschätzung

Selektion

- Selektion mit Eingabe von einer Relation die im Hintergrundspeicher abgelegt ist
  → Alle Seiten Lesen! → Kosten b_R
- Eingabe kommt von einem anderen Operator
  → nur filtern, keine neuen Zugriffe → 0
- Selektion mit Index:
  - B^+-Baum mit Höhe von max. 4, Wurzel und Teil der 1. Ebene meist im Hauptspeicher: → <=4 (~2)
  - Hash-Index:
    > statisches Hashing (ohne Überlauf) → 1
    > erweiterbares Hashing (f. Indirektion) → +1
  - Wenn Index nur TID enthält → +1

Kostenabschätzung

Sortierung

- Erzeugung der Level-0-Runs: jede Seite lesen, sortierung und wieder schreiben → $2 * b_R$
- Länge Level-0-Runs: m Seiten
  Anzahl der Level 0 Runs: → $i = [b_R / m]$
- Bei jedem Pass: m-1 Runs zu einem gemerged
  Anzahl der benötigten Passes: → $l = [\log_{m-1}(i)]$
- Bei jedem Pass: alle Seiten gelesen und wieder geschrieben. Pro Pass also → $2 * b_R$
- Gesamtkosten:

$$2b_R + l \cdot 2b_R = 2b_R \cdot (1 + l) = 2b_R \cdot (1 + [\log_{m-1}([b_R / m])])$$

Kostenabschätzung

Joins: (Simple) Nested Loop Join

- Jede Seite von R wird einmal gelesen → b_R
- Für jedes Tupel von R muss jede Seite von S einmal gelesen werden: → $M_R \,.\, b_S$
- Gesamtkosten: $b_R + M_R \cdot b_S$

Beispiel:

Anzahl Seiten	b_R=1.000	b_S=500
Anzahl Tupel	M_R=100.000	M_S=50.000
Tupel / Seite	p_R=100	p_S=100
DB Puffer	M=100	

Gesamtkosten =
1.000 + 100.000 * 500 =
50.001.001 I/Os

(bei 10ms pro I/O): ~ 140 Std

Kostenabschätzung

Joins: Pagewise Nested Loop Join

- Jede Seite von R wird einmal gelesen $\rightarrow b_R$
- Für jede Seite von R muss jede Seite von S einmal gelesen werden: $\rightarrow b_R \,.\, b_S$
- Gesamtkosten: $b_R + b_R \cdot b_S$

Beispiel:

Anzahl Seiten	b_R=1.000	b_S=500
Anzahl Tupel	M_R=100.000	M_S=50.000
Tupel / Seite	p_R=100	p_S=100
DB Puffer	M=100	

Gesamtkosten =
1.000 + 1.000 * 500 =
501.000 I/Os

(bei 10ms pro I/O): ~ 1,4 Std

Kostenabschätzung

Joins: Block Nested Loop Join

- Jede Seite von R wird einmal gelesen $\rightarrow b_R$
- Für jeden Block aus (m-k-1) Seiten von R muss jede Seite von S einmal gelesen werden. Ab dem 2. Durchlauf von S stehen die ersten k Seiten bereits im Puffer.
  > 1. Durchlauf von S: $\rightarrow b_S$
  > Weitere DL von S: $\rightarrow (b_S-k)$
  > Gesamtanzahl der Durchläufe: $\rightarrow [b_R/(m-k-1)]$
- Gesamtkosten: $b_R + k + \frac{b_R}{(m-k-1)} \cdot (b_S - k)$

Unter welchen Bedingungen sind die I/O Kosten optimal?
- soll R oder S kleiner sein?
- soll k groß oder klein gewählt werden?

Kostenabschätzung

Joins: Index Nested Loop Join

- Jede Seite von R wird einmal gelesen $\rightarrow b_R$
- Für jedes Tupel in R Zugriff auf Tupel in S je nach Indexart $\rightarrow$ c=1-5 I/Os
- Gesamtkosten

  wenn max 1 Tupel in S (B Schlüssel in S): $b_R + c \cdot M_R$

  wenn mehrere Treffer in S möglich:
  > geballter Index: $b_R + M_R \cdot (c + b_S \cdot sel_{RS})$

  > ungeballter Index: $b_R + M_R \cdot (c + M_S \cdot sel_{RS})$

Kostenabschätzung

Joins: Sort Merge Join

- Sortieren von R: $\rightarrow 2b_R \,.\, (1+l_R)$
- Sortieren von S: $\rightarrow 2b_S \,.\, (1+l_S)$

- Kosten für Merge Join, wenn
  > A in R oder B in S Schlüssel ist: $\rightarrow b_R + b_S$
  (je 1 Durchlauf von R und S)

  > für jedes R kann es mehrere Tupel in S geben:
  Worst Case als Nested Loop, wenn fast alle Werte von R.A und S.B gleich sind.

Kostenabschätzung

Joins: Hash Join

- Build-Phase:
  je einmal lesen und schreiben → $2(b_R + b_S)$
- Probe-Phase:
  jede Seite von R und S je einmal → $(b_R + b_S)$
- Gesamtkosten: $3 \cdot (b_R + b_S)$

Beispiel:

Anzahl Seiten	b_R=1.000	b_S=500
Anzahl Tupel	M_R=100.000	M_S=50.000
Tupel / Seite	p_R=100	p_S=100
DB Puffer	M=100	

Gesamtkosten =
3 * (1.000 + 500) =
4.500 I/Os

(bei 10ms pro I/O): 45 Sek.