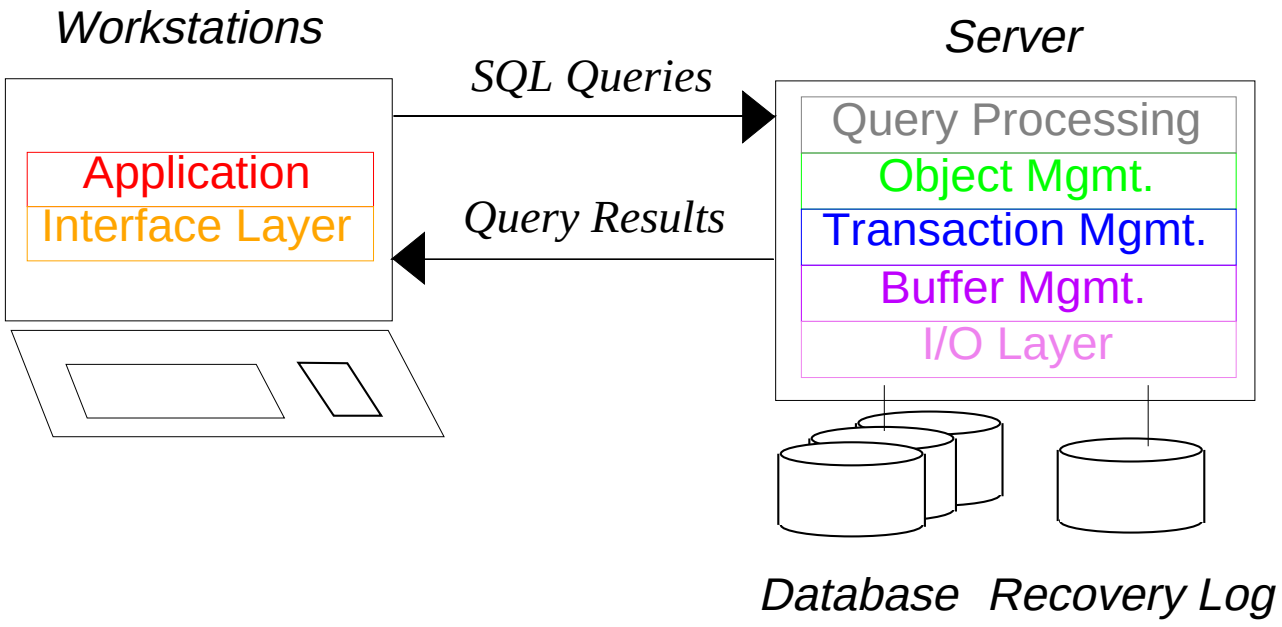
Client-Server Architekturen: Query Shipping

Grundprinzip

1. Client schickt Anfrage zum Server
2. Server schickt Ergebnisse der Anfrage zurück

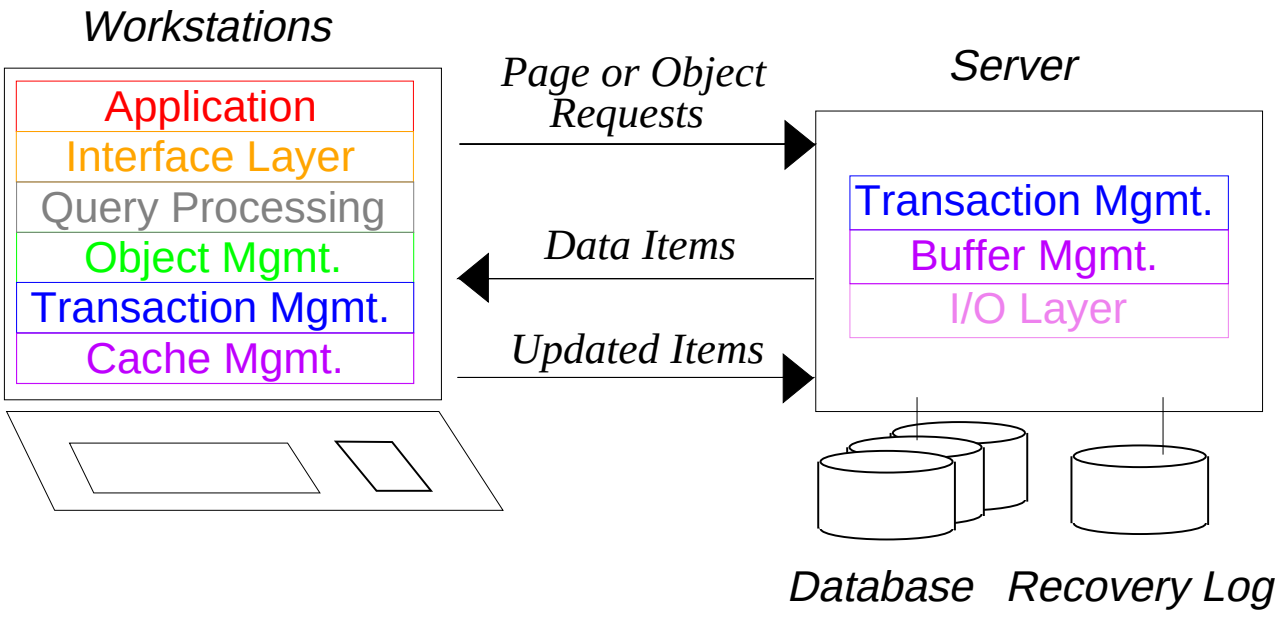


Beispiele: RDBMSe wie Oracle, IBM DB2, etc.

Client-Server Architekturen: Data Shipping

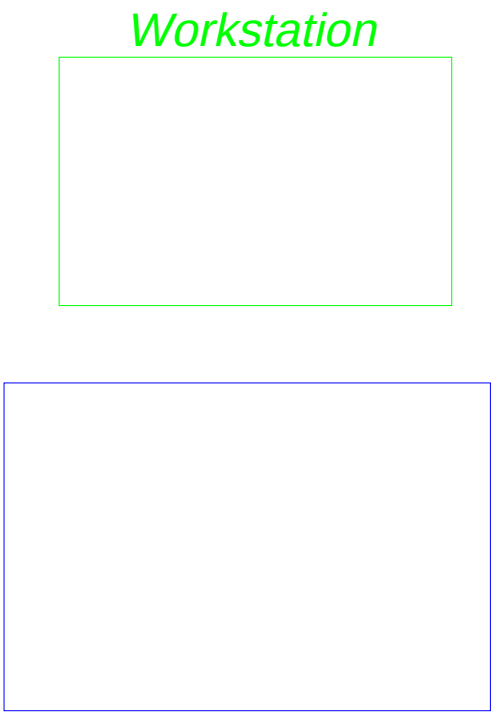
Grundprinzip

1. Anfragen werden am Client ausgeführt
2. Client holt sich Daten von den Servern bei Bedarf
3. Client "cached" Daten



Beispiele: OODBMSe wie O_2, ObjectStore, etc.

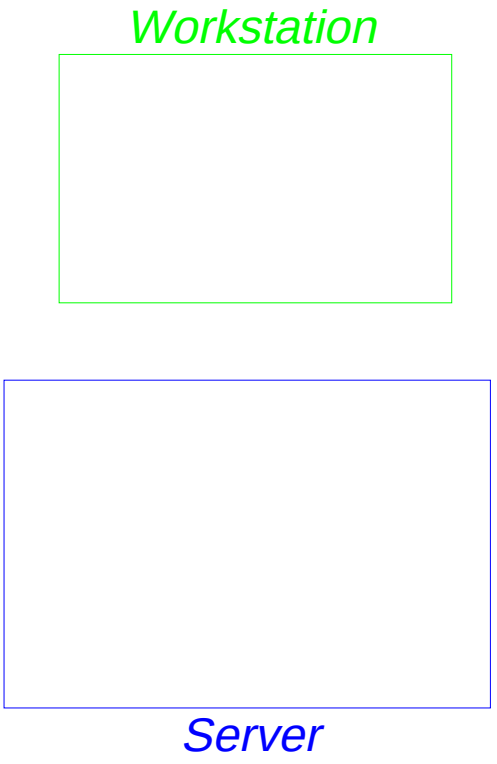
Data Shipping



Client-Server Architekturen: Hybrid Shipping

Grundprinzip

- Anfragen werden an Clients und Servern ausgeführt



Beispiele

- einige Forschungsprototypen (ORION)
- heterogene Datenbanksysteme (Garlic, Data Joiner)
- Anwendungssysteme (SAP R/3)

Leistungsbewertung: QS vs. DS vs. HY

Query Shipping

- effiziente Bearbeitung an Servern
- niedrige Kommunikationskosten, da nur gefilterte Zwischen- oder Endergebnisse verschickt werden
- nutzt Client Ressourcen nicht aus

Data Shipping

- nutzt Client Ressourcen aus
- falls Caching nicht greift, hohe Kommunikationskosten
- falls Caching greift, sehr gut

Hybrid Shipping

- verbindet Vorteile von QS und DS
- nutzt Parallelität zwischen Client und Server aus

Cache Investment für Hybrid Shipping

Beobachtung

Es gibt eine wechselseitige Abhängigkeit zwischen Anfrageoptimierung und Caching.

Caching beeinflußt Anfrageoptimierung

- falls Daten gecached, führe Operation am Client aus
- falls Daten nicht gecached, führe Operation am Server aus

Anfrageoptimierung beeinflußt Caching

- falls Operation am Client ausgeführt wird, sind die Daten anschließend gecached
- falls Operation am Server ausgeführt wird, ist der Zustand des Caches unverändert

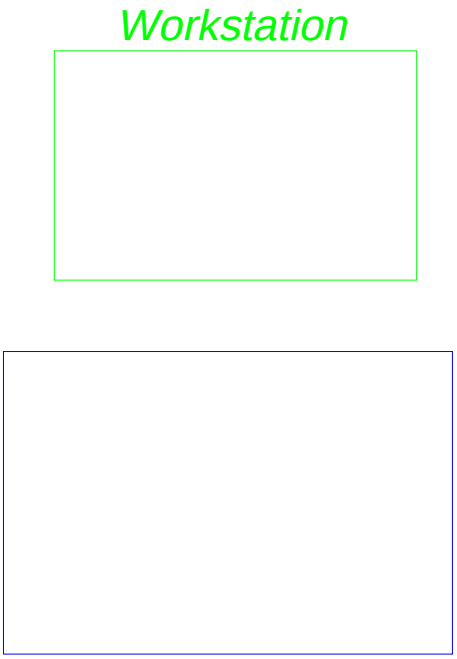
Zusammenfassung der Beispiele

1. Wenn man nicht aufpaßt, dann degeneriert Hybrid Shipping zu Query Shipping.
2. Data Shipping ist nicht die Lösung.
3. Man braucht *Cache Investment*; entscheidet
   - wann suboptimale Pläne erzeugt werden
   - für welche Daten suboptimale Pläne erzeugt werden

Profitable Policy für Tabellen

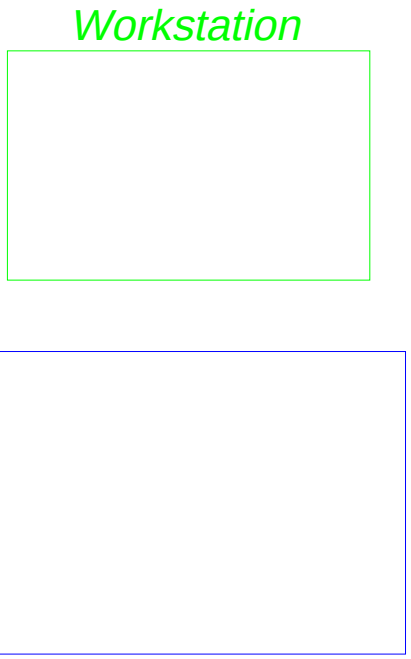
Führe "what-if" Analysen aus, um zu entscheiden, wie wertvoll es ist, eine Tabelle zu cachen.

Plan I: Tabelle gecached



Kosten: 10 secs
Gewinn: 190 secs

Plan II: Tabelle nicht gecached

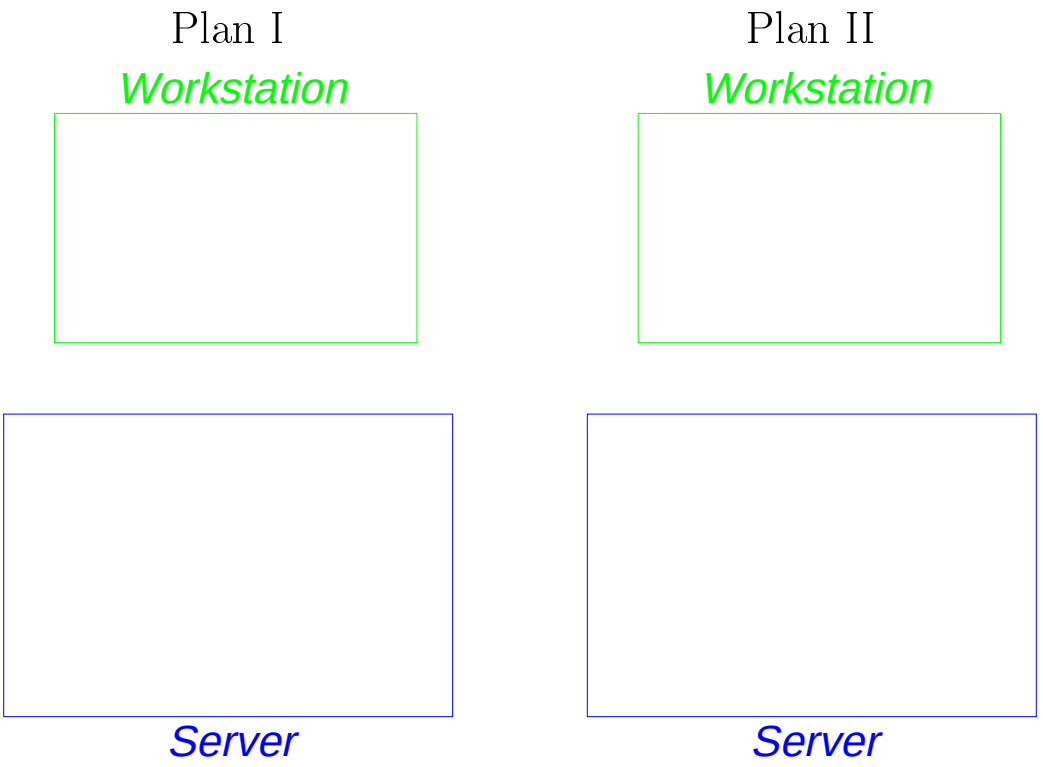


Kosten: 200 secs

- Kumuliere Gewinne zu Gesamtwert (V)
- Cache Tabellen mit höchstem V/S
  im Grunde ist das ein Rucksackproblem; S Größe der Tabelle

Profitable Policy: Investment

- manchmal ist es am besten gar nichts zu cachen
- *Wert* einer Tabelle muß höher sein als *Investment*



Tatsächliche Kosten: 430 secs
Investment: 230 secs

Kosten: 200 secs

- I. hängt u.a. von Selektivität von Prädikaten ab.
- Manchmal ist es am besten auf Anfragen mit niedriger Selektivität zu warten.

Reference-counting Policy für Tabellen

Motivation

- Profitable Policy trifft sehr gute Entscheidungen
- aber: "what-if" Analysen sind teuer, da der Optimierer viele Pläne für eine Anfrage erzeugen muß

Ansatz

- Vorbild: LRU/LFU Seitenersetzungsstrategie
- zähle, in wievielen Anfragen eine Tabelle vorkommt
- cache die Tabellen mit dem höchsten $Zähler/S$ (das ist das Rucksackproblem)
- cache nur, falls $Zähler > Schwellwert$ (Schwellwert ist Maß für das Investment)

Weitere Aspekte

Behandlung von Updates

- Updates machen Caching weniger attraktiv aufgrund des Synchronisationsaufwandes
- mit jedem Update wird Wert einer Tabelle reduziert
- bei Callback Locking: reduziere Wert proportional
- bei Propagierung: reduziere Wert um eine Konstante

Wechselnde Anfrageprofile eines Client

- gewichte "alte" Anfragen weniger als "neue" Anfragen
- CI Ansatz: exponential aging; [Effelsberg & Härder 1982]
- im Buffer Manager: LRU Ersetzungsstrategie

Cache Investment für Tabellen: Ergebnisse

Hybrid Shipping mit Cache Investment schlägt

- Query Shipping oder HY ohne Cache Investment
  Faktor: 1 bis 5
- Data Shipping
  Faktor: 1 bis 20

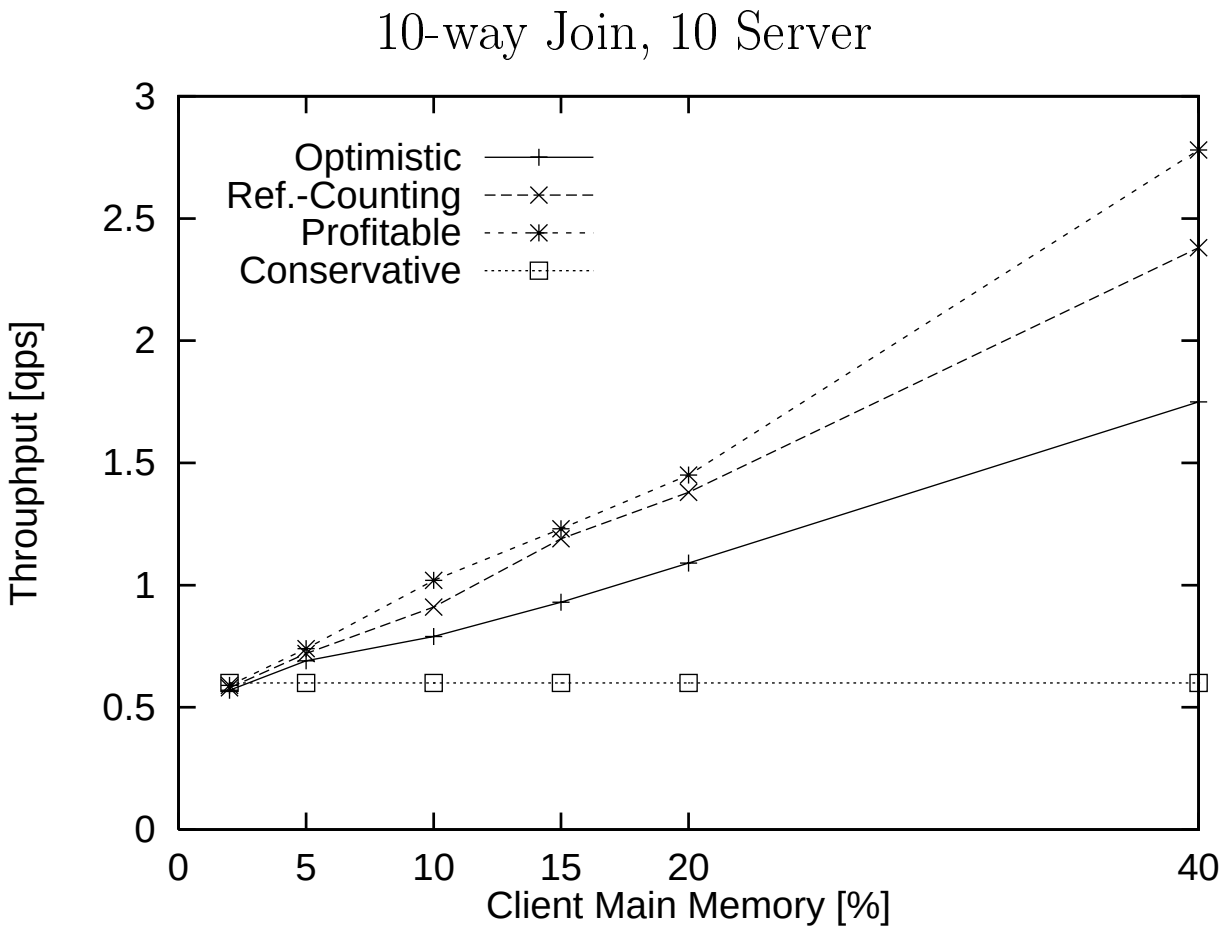
Profitable Policy

- erzeugt immer sehr gute Pläne
- Zusatzaufwand der "what-if" Analysen wird schmerzhaft für Anfragen mit mehr als 5 Joins

Reference-counting Policy

- kein spürbarer Zusatzaufwand
- klappt meistens; Probleme bei Update-intensiven Anwendungen und Umgebungen mit vielen Servern

Throughput


10-way Join, 10 Server
Throughput [qps]
Client Main Memory [%]
Optimistic
Ref.-Counting
Profitable
Conservative
0
0.5
1
1.5
2
2.5
3
0
5
10
15
20
40

Cache Investment für Indexe

- bisherige Betrachtungen nur sinnvoll,
  wenn es keine Indexe gibt
- Indexe sind jedoch nützlich in jeder Art von System
- Indexe erhöhen die Anzahl der Alternativen
  1. nur den `Emp.salary` Index cachen
  2. nur die `Emp` Tabelle cachen
  3. `Emp.salary` Indexe und `Emp` Tabelle cachen
  4. gar nichts cachen, was mit `Emp` zu tun hat

- ändert das Modell, Anpassung der Policies notwendig
- Grundmechanismen sind allerdings gleich

Beispiel 1

```
SELECT    e.name, e.salary
FROM      Employee e
WHERE     e.age > 50
```

	`Emp` Tabelle	`Emp.age` Index
Wert	0 secs	20 secs
Investment	0 secs	5 secs

Richtige Entscheidung

`Emp.age` Index sollte gecached werden.

Beispiel 2

```
SELECT    e.name, e.salary
FROM      Employee e
WHERE     e.age > 3
```

	`Emp` Tabelle	`Emp.age` Index
Wert	370 secs	372 secs
Investment	200 secs	1760 secs

Richtige Entscheidung

`Emp` Tabelle sollte gecached werden.

Reference-counting Policy für Indexe

Vorgehensweise

1. erzeuge besten Plan für die Anfrage
2. führe diesen Plan aus
3. analysieren den Plan
   - falls `idxscan` im Plan, erhöhe Zähler des Index
   - falls `tbscan` im Plan, erhöhe Zähler der Tabelle

Vorläufige Bewertung

- klappt in den beiden Beispielen
- allerdings ein bißchen kriseliger in anderen Situationen, wie wir gleich sehen werden

Berechnung von S für Indexe

Anzahl benötigter Indexseiten für eine Anfrage

$$i_{new} = \frac{k}{N} * m$$

Anzahl benötigter Tabellenseiten für eine Anfrage:

geclusterter Index	nicht geclusterter Index
$t_{new} = \frac{k}{N} * n$	$t_{new} = \text{Yao}(k, N, n)$

Anzahl benötigter Indexseiten für alle Anfragen:

$$I := I + i_{new} - \frac{I * i_{new}}{m}$$

Anzahl benötigter Tabellenseiten für alle Anfragen:

$$T := T + t_{new} - \frac{T * t_{new}}{n}$$

Insgesamt

$$S = I + T$$

Cache Investment für Indexe: Ergebnisse

Hybrid Shipping mit Cache Investment schlägt

- Query Shipping oder HY ohne Cache Investment
  Faktor: 1 bis 20
- Data Shipping
  Faktor: 1 bis 10

Profitable Policy

- erzeugt, wie gehabt, die besten Pläne
- Zusatzaufwand der "what-if" Analysen bereits schmerzhaft bei einfachen Anfragen mit vielen Indexen

Reference Counting

- kein spürbarer Zusatzaufwand
- es gibt mehr Fälle, in denen es scheitert
- manchmal sogar schlechter als DS oder QS

Response Time

POINTRANGE Workload