# Allokation (Dadam: Kapitel 4.5)

**Ziel:** Plaziere Partitionen auf Knoten so, daß die Gesamtkosten (Kommunikation plus Speicherung plus lokale Ausführung von Operationen) minimal wird.

- Das sieht nach Rechnen aus!
  (Man kann die Kostenfunktionen beliebig detailliert und kompliziert machen.)

- Man braucht Wissen über Art und Häufigkeit der Anfragen und Updates, Größe der Partitionen, Kommunikationskostenmatrix, Speicherkosten der Knoten.

- Freiheitsgrad: Nicht-redundante vs. redundante Speicherung
  (*Pro Replikation:* höhere Verfügbarkeit, bessere Performance
  *Contra Replikation:* komplizierte Implementierung und Administration)

# Parameter des Modells

## Eingaben:

| | |
|---|---|
| $K$ | Anzahl der Knoten |
| $M_i$ | Speicherkapazität von Knoten $i$ |
| $S_i$ | Speicherkosten pro Einheit von Knoten $i$ |
| $P$ | Anzahl der Partitionen |
| $G_p$ | Größe der Partition $p$ |
| $U_{ij}$ | Übertragungskosten pro Einheit von Knoten $i$ nach $j$ |
| $R_{tp}$ | Größe des Resultats einer Teiloperation vom Typ $t$ auf Partition $p$ |
| $H_{it}$ | Häufigkeit mit der eine Teiloperation vom Typ $t$ am Knoten $i$ initiiert wird. |

## Gesucht:

$V_{pi}$ Verteilung der Partitionen

$$V_{pi} = \begin{cases} 1 & : \quad p \text{ am Knoten } i \text{ allokiert} \\ 0 & : \quad \text{sonst} \end{cases}$$

# Kostenfunktionen und Nebenbedingungen
## Ohne Replikation

### Kostenfunktionen:

1. Gesamte Speicherkosten:

$$S = \sum_{p,i} G_p * V_{pi} * S_i$$

2. Übertragungskosten für alle Anfragen:

$$U = \sum_{i,t,p,j} H_{it} * R_{tp} * V_{pj} * U_{ji}$$

3. Weitere Kostenfunktionen Eurer Wahl

### Nebenbedingungen:

1. nicht-redundante Speicherung:

$$\forall p : \sum_{i} V_{pi} = 1$$

2. max. Speicherkapazität

$$\forall i : \sum_{p} G_p * V_{pi} \leq M_i$$

### Das Optimierungsproblem:

Minimiere $S + U$ unter den Nebenbedingungen.

# Allokation mit Replikation

Annahme ROWA Protokoll (wir lernen noch andere kennen):

**Read:** Man muß nur billigste Kopie einer Partition lesen.

**Update:** Man muß alle Kopien einer Partition schreiben.

**Was ändert sich am Modell:**

1. Nebenbedingung lautet jetzt:

$$\forall p : \sum_i V_{pi} \geq 1$$

2. Übertragungskosten:

$$U = \sum_{i,t,p} H_{it} * \Phi_{j:p_j=1}(R_{tp} * U_{ji})$$

$\Phi$ entspricht **min**, falls $H_{it}$ Read
$\Phi$ entspricht $\Sigma$, falls $H_{it}$ Update

**Das Optimierungsproblem ist aber im Grunde dasgleiche.**

# Beispiel

$K = 3$; $S_i = (120, 100, 110)$; $M_i = (\infty, \infty, \infty)$

$P = 4$; $G_p = (1000, 1500, 500, 2000)$

$$U_{ij} = \begin{pmatrix} 0 & 25 & 30 \\ 25 & 0 & 35 \\ 30 & 35 & 0 \end{pmatrix}$$

$$R_{tp} = \begin{pmatrix} 100 & 200 & 10 & 20 \\ 200 & 300 & 5 & 8 \\ 10 & 10 & 200 & 100 \end{pmatrix}$$

$$H_{it} = \begin{pmatrix} 50 & 7 & 10 \\ 5 & 75 & 2 \\ 3 & 8 & 50 \end{pmatrix}$$

**Fragen:**

1. Wie gut ist: $P_1 \to$ I; $P_2 \to$ II; $P_3, P_4 \to$ III?
2. Was passiert, wenn man $P_1$ nach III migriert?

# Abschließende Bemerkungen

- Das Optimierungsproblem (mit oder ohne Replikation) kann wie folgt gelöst werden:

  1. **erschöpfend:** probiere alle Allokationsmöglichkeiten
     liefert "optimale" Allokation; u.U. sehr aufwendig
  2. **heuristisch:** maximiere $H_{it}$ oder minimiere $S_i$ oder Simulated Annealing oder oder oder
     liefert u.U. schlechte Allokation; schnell

  (Im allgemeinen ist das Allokationsproblem $\mathcal{NP}$-hart.)

- Das hier vorgestellte Modell ist sehr einfach:

  1. Eigenheiten von Join-Queries sind nicht berücksichtigt.
  2. Lokale Kosten für Berechnungen sind nicht berücksichtigt.

- Parameter wie $H_{it}$ sind in der Praxis oft unbekannt; außerdem ändern sie sich ständig.

  <u>dynamische</u> Techniken → Caching, Forschung

**Fazit zu Partitionierung und Allokation:**
Das ist alles sehr akademisch – aber besser als gar nichts.