Karlsruher Institut für Technologie

Datenbanksysteme

Kapitel 11: Datenbank-Funktionalität in der Cloud – ausgewählte Aspekte (Teil 1)

Lehrstuhl für Systeme der Informationsverwaltung, Fakultät für Informatik

Photo by Pattys-photos

Was ändert sich in der Cloud?

Gemeint ist: Wenn Cloud-Anbieter SQL-Schnittstelle ‚nach oben' anbietet.

- Physischer Entwurf muss automatisch erfolgen.
  Verteilung der Daten ist obligatorisch.
- Anfrageauswertung in Gegenwart anderer Anfragen.
  Muss entsprechend geplant werden.
- Unterschiedliche QoS-Vereinbarungen
  mit unterschiedlichen Dienstnehmern.
- Plötzliche krasse Zunahme von Zugriffen eines Dienstnehmers
  i. Allg. nicht vorhersagbar – Infrastruktur sollte damit umgehen können.
- „Secure Storage", d. h. Verschlüsselung der Daten,
  und gleichzeitig übernimmt aber Dienstanbieter möglichst großen Teil
  der Anfrageauswertung, ist bedeutsam.

Gliederung dieses Kapitels

- Grundlagen, nicht Cloud-spezifisch.
- (key, value)-Stores
  als Alternative zur relationalen Datenverwaltung.
  - Betonung der Unterschiedlichkeit der Anforderungen,
  - Darstellung anhand eines konkreten Beispiels/ Anwendungsfalles.
- (key, value)-Stores als Building Block
  für relationale Technologie.
  - Motivation, was sind die Vorteile?
  - Realisierung.
  - Derzeitige Limitierungen.
  - Zusammenhang zu ‚Cloud‘.

Relationale Algebra – Motivation

- Anfragesprache (SQL) ist definitiv nützlich (Datenunabhängigkeit).
- Wir betrachten in diesem Kapitel aber anderen Mechanismus, relationale Algebra: Erlaubt Aussagen zur Ausführungsreihenfolge.

Laufendes Beispiel

Künstler

KID	KNAME	LAND	JAHR
1012	Neil Young	Kanada	1945
1014	Rammstein	Deutschland	1994
1015	The Ramones	USA	1974
1016	Eric Fish	Deutschland	1969

Titel

TITLE ID	NAME	ART	GRÖSSE	KID
102	Neil Young – Heart of Gold	mp3	2.920kb	1012
103	Rammstein – Ich liebe Neil Young	wma	4.234kb	1014
104	Neil Young – Old Man	mp3	3.161kb	1012
105	Neil Young – Four Strong Winds	wma	5.125kb	1012

Projektion (1)

- Beispiel 1: Projektion auf ein Attribut

  π[Land](Künstler)

  ergibt als Ergebnisrelation

LAND
Kanada
Deutschland
USA

- Doppelte Ergebnistupel eliminiert.
- Ist mengenorientiert
  – insbesondere keine Bezugnahme auf einzelne Tupel in einer Schleife.

Projektion (2)

- Man kann Ausdrücke der relationalen Algebra aus mehreren Operatoren zusammensetzen.
  Beispiel: π[KName](π[KName, Land](Künstler))
- Gilt auch in Kombination mit andren Operatoren: Abgeschlossenheitseigenschaft

Projektion (3)

- Einfache Optimierungsregel:
  Bei vielen Projektionen hintereinander
  reicht die zuletzt ausgeführte auch allein

  `π[KName](π[KName, Land](Künstler))`

  ergibt optimiert

  `π[KName](Künstler)`

Selektion (1)

- Beispiel

  σ[Jahr $\leq$ 1980](Künstler)

  ergibt

KID	KNAME	LAND	JAHR
1012	Neil Young	Kanada	1945
1015	The Ramones	USA	1974
1016	Eric Fish	Deutschland	1969

Selektion (2)

- Einfache Optimierungsregeln:
  - Selektionen lassen sich beliebig vertauschen,
    Beispiel: $\sigma_{KID=1012}(\sigma_{Jahr\leq1980}(Künstler))$
    $= \sigma_{Jahr\leq1980}(\sigma_{KID=1012}(Künstler))$
  - Manchmal lassen sich
    Projektion und Selektion vertauschen.
    - Ist $\pi_{KID}(\sigma_{Jahr\leq1980}(Künstler))$
      $= \sigma_{Jahr\leq1980}(\pi_{KID}(Künstler))$?

Verbund: Beispiel

- *Unäre* vs. *binäre Operationen*.

- $\texttt{Künstler} \bowtie \texttt{Titel}$

  ergibt

TITLE ID	NAME	ART	GRÖSSE	KID	KNAME	LAND	JAHR
102	Neil Young – Heart of Gold	mp3	2.920kb	1012	Neil Young	Kanada	1945
103	Rammstein – Ich liebe Neil Young	wma	4.234kb	1014	Rammstein	Deutschland	1994
104	Neil Young – Old Man	mp3	3.161kb	1012	Neil Young	Kanada	1945
105	Neil Young – Four Strong Winds	wma	5.125kb	1012	Neil Young	Kanada	1945

- Künstler ohne Titel verschwinden: Tupel, die keinen Partner finden (dangling tuples), werden eliminiert.
- In SQL: *outer join*, der „dangling tuples“ übernimmt.

Eigenschaften Verbund

- Verbund kommutativ: $r_1 \bowtie r_2 = r_2 \bowtie r_1$
- Verbund assoziativ: $(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$
- Daher erlaubt: $\bowtie_{i=1}^{p} r_i$
- Beispiel dafür, daß Join-Reihenfolge wichtig:

r_1

B
b

r_2

B	C
a	c1
…	…
a	c100000

r_3

B	D
a	d1
…	…
a	d100000

Implementierung von Algebra-Operatoren

- Bis hierhin keine Aussagen zu Implementierung.
- I. Allg. gibt es mehrere Möglichkeiten, diese Operatoren zu realisieren.
- Join ist diesbezüglich am vielfältigsten. (Ist auch am aufwendigsten.)

Binäre Operationen

- Binäre Operationen – typischerweise tupelweiser Vergleich der beiden Input-Mengen.
  - *Nested-Loop* or *Schleifen-basierte Iteration*:
    Für jeden Tupel der externen Relation S
    Iteration über die innere Relation R.
  - *Merge-basierte Ansätze*:
    Iteriere über R und S, beide sortiert,
    Tupel für Tupel, in Sortierreihenfolge.

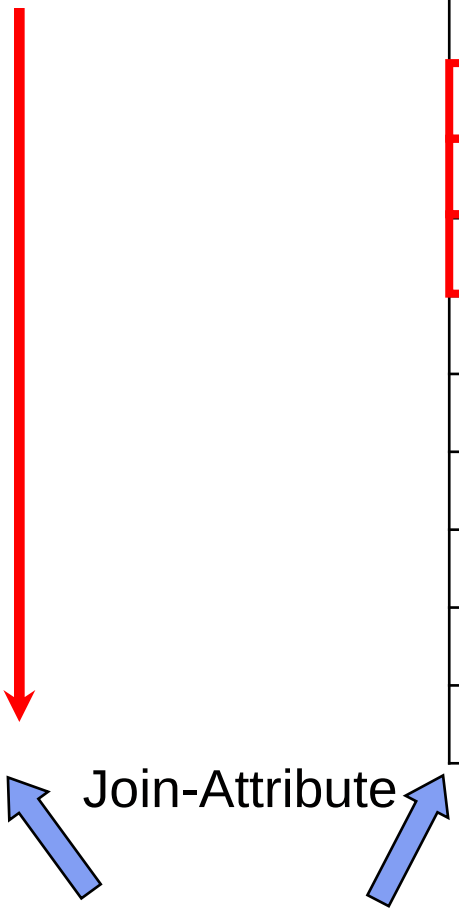
Illustration
Nested-Loop Join, Equijoin

Relation A

ATT1	ATT2
	28

Relation B

ATT2	ATT3
28	
28	



Grundlagen

- Rel. Alg.
- Operator-Implement.
- Histo-gramme
- Replikation

Dynamo

Begrenzung Erg.größe

Schluss

Illustration
Nested-Loop Join, Equijoin

Relation A

ATT1	ATT2
	28

Relation B

ATT2	ATT3
28	
28	

Relation A muss wiederholt
aus dem Sekundärspeicher geholt werden; teuer.

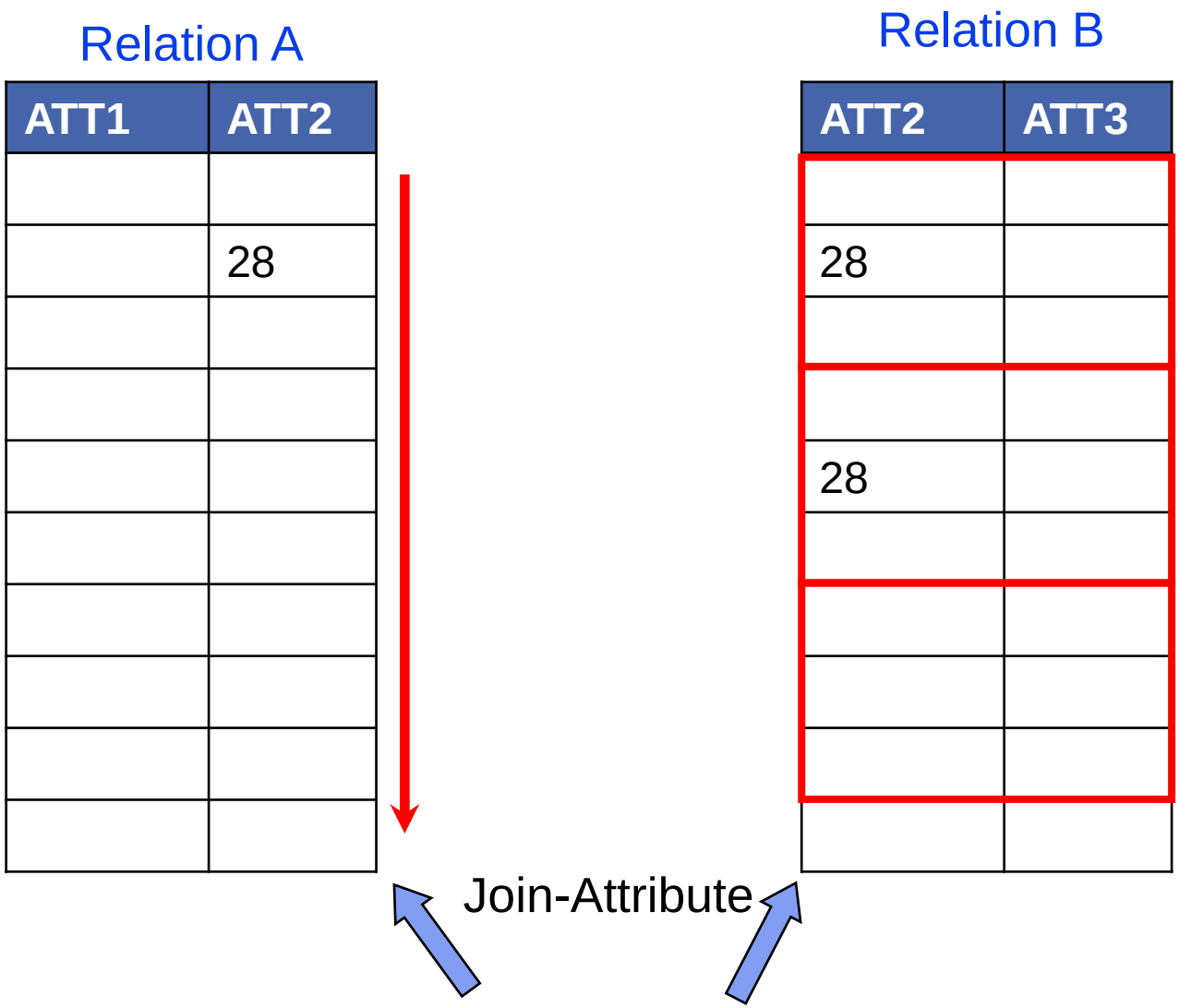
Nested-Loop Join

- Um $r \bowtie s$ zu berechnen,
  iteriert Nested Loop über alle $t_1 \in r$ und alle $t_2 \in s$.
- $r \bowtie_\phi s$:

```
for each t_r ∈ r do
   begin
      for each t_s ∈ s do
      begin
         if φ(t_r, t_s) then put (t_r · t_s) endif
      end
   end
endloop
```

- Kosten: $O(n_s \cdot n_r)$

Illustration
Block-Nested-Loop Join, Equijoin



Grundlagen

- Rel. Alg.
- Operator-Implement.
- Histo-gramme
- Replikation

Dynamo

Begrenzung Erg.größe

Schluss

Block-Nested-Loop Join

- Iteriere über Blocks anstelle von Tupeln.

```
for each Block B_r of r do
begin
   for each Block B_s of s do
   begin
      for each Tuple t_r∈B_r do
      begin
         for each Tuple t_s∈B_s do
         begin
            if ϕ(t_r, t_s) then put(t_r · t_s) endif
         end
      end
   end
end
```

- Kosten: $b_r \cdot b_s$
- Bzw. nicht nur Blocks, sondern fast ganzen Arbeitsspeicher mit Tupeln der einen Relation füllen.

Merge Techniken

- $X := R \cap S$;
  Sortiere R and S zunächst nach X,
  wenn das nicht bereits geschehen ist.
  1. $t_r(X) < t_s(X)$, lies nächstes $t_r \in r$,
  2. $t_r(X) > t_s(X)$, lies nächstes $t_s \in s$,
  3. $t_r(X) = t_s(X)$, merge t_r mit t_s
     und allen Nachfolgern von t_s
     mit den gleichen Werten für X,

     Beim ersten $t_s' \in s$ mit $t_s'(X) \neq t_s(X)$,
     beginne erneut beim ursprünglichen t_s,
     wiederhole dies mit den Nachfolgern t_r' von t_r,
     so lange wie $t_r(X)=t_r'(X)$.

Merge Techniken – Illustration

Relation A

ATT1	ATT2
	28

Relation B

ATT2	ATT3
28	
28	

Merge Techniken: Kosten

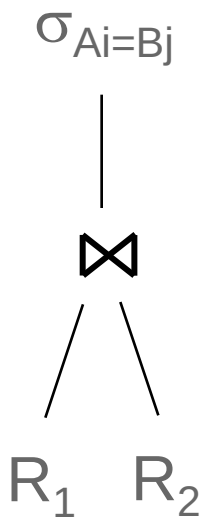
- Alle Tupel haben den gleichen X-Wert: $O(n_r \cdot n_s)$,
- X ist Schlüssel R oder S: $O(n_r \log n_r + n_s \log n_s)$
  Warum?
- Wenn die Relationen sortiert sind
  (und erstes Bullet nicht zutrifft): $O(n_r + n_s)$
  Warum?

Logische vs. physische Operatoren (1)

- Es existieren noch weitere Join-Algorithmen, z. B. Hash-basierte.
- Datenbank-System beinhaltet i. Allg. unterschiedliche Implementierungen. Gekapselt als Join-Operator. Z. B. Nested Loop Join Operator oder Merge-Join Operator.
- *Logische vs. physische Operatoren*.
- Datenbanksystem sucht selbst passenden physischen Operator heraus. I. Allg. den, der die niedrigsten Kosten verursacht. (Physische Datenunabhängigkeit)

Logische vs. physische Operatoren (2)

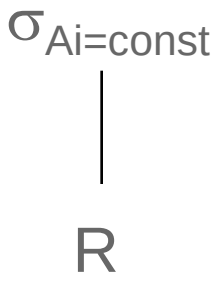
- Physische Operatoren können i. Allg. mehr als einen logischen Operator enthalten.
- Beispiel:



- Wichtig zur Vorstellung eines Ansatzes weiter hinten.

Logische vs. physische Operatoren (3)

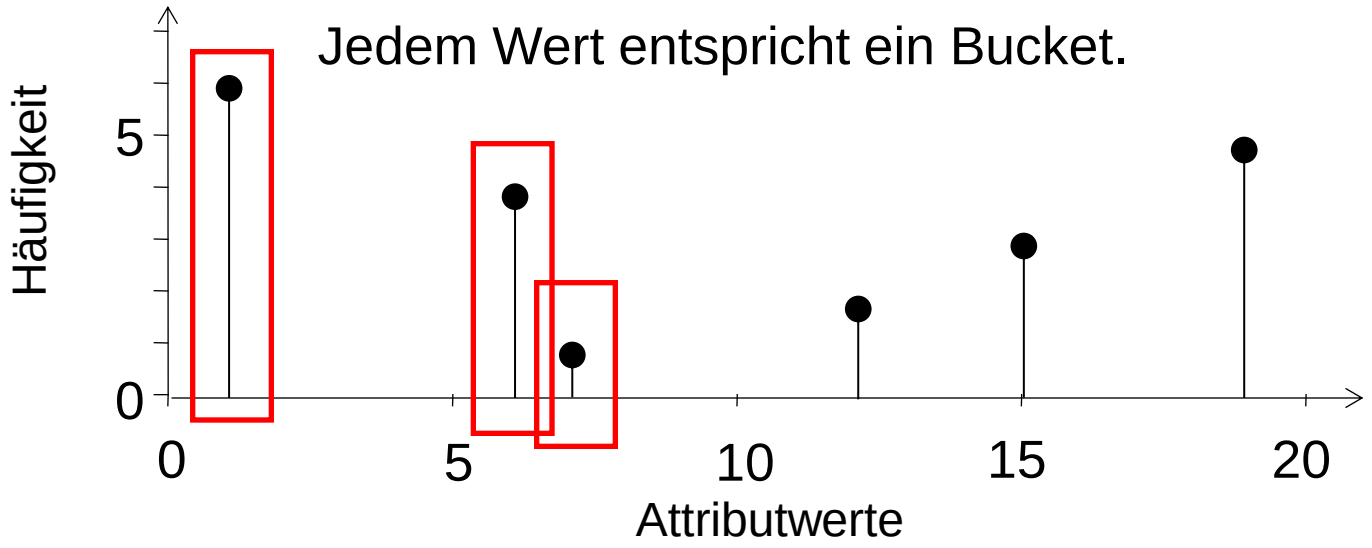
- Physische Operatoren können i. Allg. mehr als einen logischen Operator enthalten. Weiteres Beispiel:



- I. Allg. mehrere Möglichkeiten der Implementierung:
  1. Scan über die Relation; Tupel, die Selektionsbedingung überstehen, werden behalten.
  2. „Index-Scan“:
     - Angenommen, es existiert Index über Attribut A_i.
     - Mit Index genau die Tupel holen, die Attributwert const haben.

Histogramme – Prinzip

Spalte einer Relation: 1, 1, 1, 1, 1, 1, 6, 6, 6, 6, 7, 12, 12, 15, 15, 15, 19, 19, 19, 19, 19



Attributwert	1	6	7	12	15	19
Häufigkeit	6	4	1	2	3	5

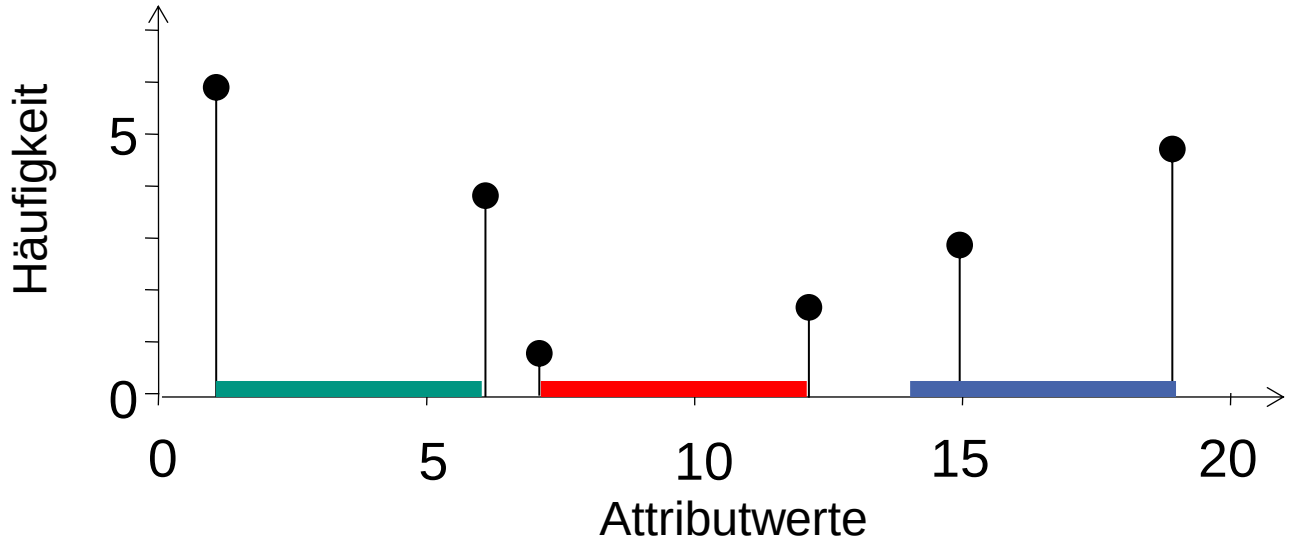
Kompaktere Darstellung als ursprüngliche Werteliste.

Histogramme – Erläuterungen

- Zeigt Häufigkeit, mit der einzelne Werte auftreten.
- **Kompression:** Einteilen der Dimension in eine vorgegebene Anzahl Intervalle = “Buckets”,
- sehr verbreitet in der Praxis,
- i. Allg. relativ gute Approximation,
- Alternativen bezüglich Partitionierung; Auswahl schwierig.

Equi-Width Histogramme

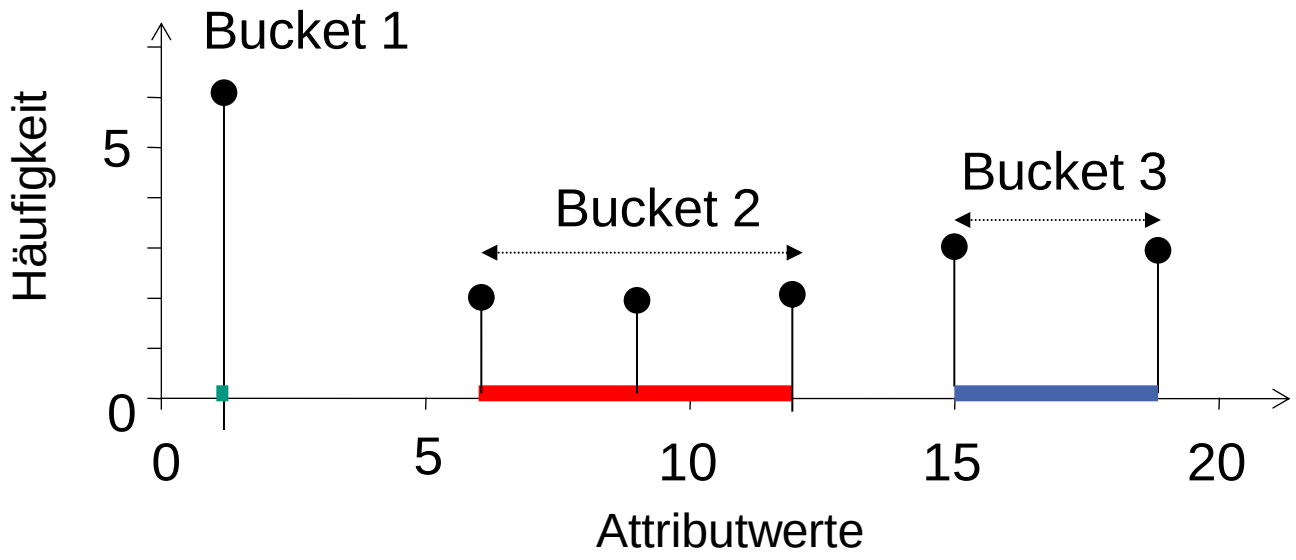
Die Breite aller Buckets ist gleich.



Attributwerte	[1, 6]	[7, 12]	[14, 19]
Häufigkeit	6 + 4 = 10	1 + 2 = 3	3 + 5 = 8

Equi-Depth Histogramme

Die “Tiefe” aller Buckets (Summe der Häufigkeiten) ist überall gleich.



Attributwerte	1	[6, 12]	[15, 19]
Häufigkeit	6	3 x 2 = 6	2 x 3 = 6

Histogramme – kurze Diskussion

- Nützlich bei Anfrage, die sich auf ein Attribut beschränkt.
- ("Selektiere alle Mitarbeiter mit Gehalt > 5000.")
- Deutlich weniger nützlich bei mehreren Attributen – Attributwerte i. Allg. nicht unabhängig voneinander.
  Beispiel:

```
SELECT * FROM car
WHERE brand == 'Ferrari' and color == 'red'
```

- Es gibt mehrdimensionale Histogramme, jedoch:
  - Schwierig zu konstruieren.
  - Sehr schwierig zu warten.
  - Anzahl der Attributkombinationen wächst exponentiell mit der Anzahl der Attribute.

Wockierende und nichtblockierende Operatoren

- *Operator blockiert* := Ergebnis des Operators muss vor Ausführung des nachfolgenden Operators vollständig berechnet sein.
- Beispiel: Sort-Operator
  Solange Ergebnis nicht vorliegt, ist unklar, welcher Tupel der erste im Ergebnis ist usw.
- Gegenbeispiel: Select
  Select-Operator kann einzelnen Tupel, sobald Selektionsbedingung für ihn erfolgreich überprüft wurde, an nachfolgenden Operator weitergeben.

Synchroner vs. asynchroner Zugriff

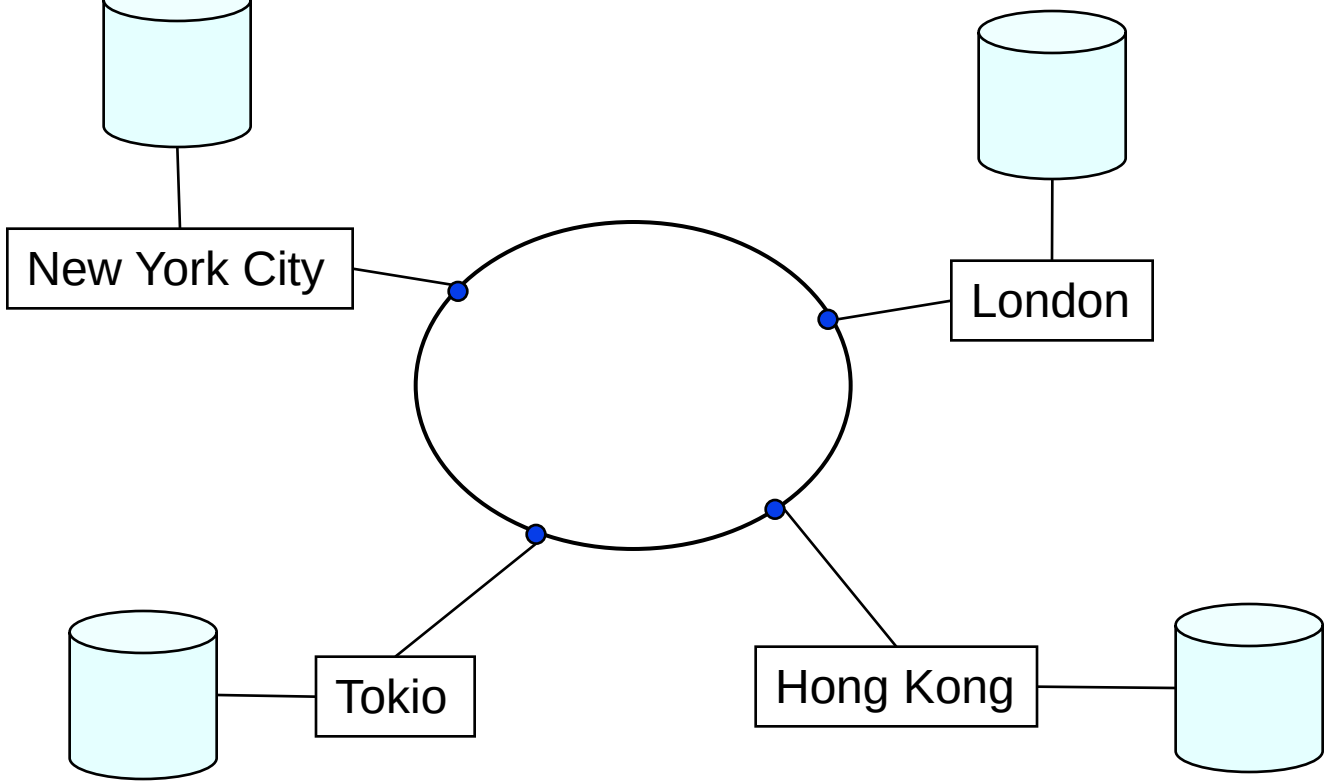
- *Synchroner* vs. *asynchroner Zugriff* – bedeutet im Folgenden:
  - synchron – Zugriff innerhalb einer Transaktion,
  - asynchron – mehrere Transaktionen.

Partitionierung vs. Replikation

- Folie aus früherem Kapitel -

NYC Employees, London Employees,
NYC Projects

NYC Employees, London Employees,
NYC Projects, London Projects



Tokio Employees,
Tokio Projects, London Projects

HK Employees,
HK Projects

Replikation – Vorteile

- Replikation – mehrere Kopien der Datenobjekte.
  Vorteile u. a.:
  - bessere Lese-Leistung,
  - höhere Verfügbarkeit beim Lesen,
  - aber: Umgekehrtes Bild beim Schreiben.

Service-Level Agreements

- *Service-Level Agreements*:
  Vereinbarung zwischen Client und Server,
  die Ausführung des Dienstes betreffend.
  - Client-seitig z. B.:
    Erwartete Rate von Dienstaufrufen,
  - Server-seitig z. B.: Erwartete Latenzzeit
    unter diesen Bedingungen.

  Beispiel: “Antwort innerhalb von 300 ms für 99,9% der Aufrufe bei einer Last von 500 Zugriffen pro Sekunde.”

Zustände

- *Zustandsbehaftet* vs. *zustandslos*.
  - zustandslos
    – z. B. Dienst, der irgendeine Umrechnung vornimmt,
  - zustandsbehaftet
    – z. B. Ausführung eines Geschäftsprozesses.

Quorum (1)

- *Quorum*.

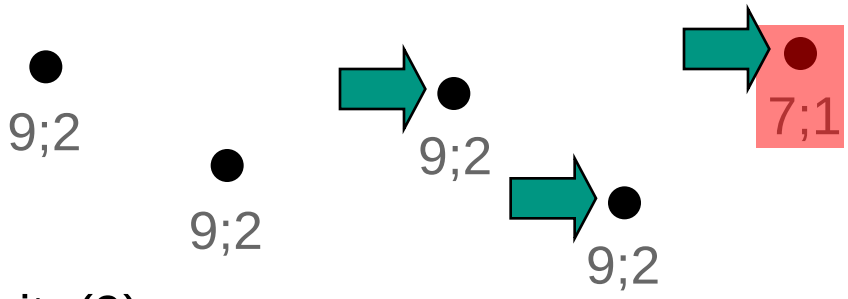
- Szenario: Replikation, N Knoten/Kopien.
- Wie kann man Konsistenz sicherstellen?
- Naives Vorgehen:
  - Transaktion will schreiben:
    Schreibsperre auf allen Knoten ist Voraussetzung.
  - Lesen: Lesesperre auf einer Kopie.
  - Um zu schreiben, müssen alle Knoten verfügbar sein.
- Was geht, ohne dass alle Knoten verfügbar sind?
  (Wir haben Schreiboperationen
  und wollen strenge Konsistenz.)

Quorum Consensus (1)

- Konsistenz mit Hilfe von Quorum-basiertem Protokoll:
  - Lesen:
    Man liest Mindestanzahl von Versionen (R)
    und nimmt die aktuellste.
  - Schreiben:
    Man aktualisiert Mindestanzahl von Kopien (W).
  - Jede Kopie hat Versionsnummer.

Quorum Consensus – Illustration

- Fünf Knoten; W=3, R=3



Wert; Versionsnummer

- write(9)
- Read

(Animierte Pfeile illustrieren Lesen.)

Es ist die read-Operation, die in der Animation dargestellt ist. Zustand ist der nach der write-Operation.

Quorum Consensus (2)

- Üblich ist ist R+W>N. Warum?

- Flexibler, als entweder alle Kopien lesen oder alle Kopien schreiben zu müssen.

Motivation (1)

- Gegenstand der Betrachtung zunächst: Datenhaltung bei Amazon. *Dynamo*.
- Maximum 36,8 Mio. Einkäufe pro Tag 2013. D. h. 426 Verkäufe pro Sekunde.
- Anforderungen, kurz zusammengefasst: Kurze Antwortzeiten, Zuverlässigkeit, Effizienz, Skalierbarkeit.
  Das Datenvolumen ist hingegen nicht riesig.

Motivation (2)

Motivation (3)

- Beispiele für ‘Anwendungen’/Services:
  - Bestseller-Listen
  - Einkaufswagen
  - Kundenpräferenzen
  - Session-Verwaltung
  - Verkaufszahlen
  - Produkt-Verzeichnis
- Laufendes Beispiel im Folgenden: Einkaufswagen. Lese-/Schreibzugriff sollte immer möglich sein.
- Zuverlässigkeit als wichtigste Anforderung: Nichtverfügbarkeit (auch nur kurz) schmälert Umsatz/Profit und führt zu Image-Schaden.

Dynamo-Schnittstelle

- Beispiel für (Schlüssel, Wert)-Paar:
  ("KBöhm", "intellektuellesBuch1 BirnbaumBuch sehrIntellektuellesBuch28")
- get-/put-Interface.
  Ausreichend für o. g. Dienste.
- Objekte sind BLOBs. Kein Datenbank-Schema.
- Operationen greifen auf ein (Schlüssel, Wert)-Paar zu, nicht mehrere.
- Beispiel: Shopping Cart
  - Schlüssel sind die Kunden,Werte der (aktuelle) Inhalt des Carts. (BLOB muss interpretiert werden.)
  - Operationen “Add to Cart”, “Delete Item from Cart”. Implementiert mit Hilfe von get und put.

Weitere Entwurfsaspekte (1)

- Das Folgende ist Zusammenfassung von bereits Gesagtem.

- Synchroner Zugriff auf Replikate
  - System muss für Schreibzugriff Sperre für jede Kopie (auf unterschiedlichen Knoten) bekommen.
  - Sogenannter *Koordinator-Knoten*
    - kümmert sich für eine Transaktion um diese Sperren,
    - führt Buch, auf welchem Knoten Arbeit schon zuende ist.
  - Nachteile:
    - System wartet auf den langsamsten Knoten.
    - Kein Fortschritt bei Ausfall eines Knotens.

  Je mehr Replikate, desto schwerfälliger.
  Keine Skalierung.
- Allgemein bekannt (CAP-Theorem):
  Hohes Maß an Konsistenz unvereinbar mit hoher Verfügbarkeit.

Weitere Entwurfsaspekte (2)

- Asynchroner Zugriff
  - Höhere Verfügbarkeit
  - weniger Konsistenz
    (d. h. Inkonsistenzen sind i. Allg. möglich).

Transaktionseigenschaften

- Dynamo verzichtet auf Konsistenz ‘in voller Schönheit’. D. h. keine Isolation.
- Schreibzugriff jeweils nur für ein Objekt. (Anscheinend kann man mehrere Zugriffe auf ein Objekt zu Transaktion bündeln.)

Weitere Entwurfsaspekte (3)

- *Eventual Consistency*
  – hier: Jeder Update muss irgendwann bei allen Replikaten ankommen, falls weitere Updates ausbleiben.
- Inkonsistenzen
  (gemeint ist: unterschiedliche Werte von Kopien) müssen erkannt und aufgelöst werden.

  a) Wann erkennt man sie?
  Wann hält man nach ihnen Ausschau?
  Alternativen insbesondere:
  Beim Lesen oder beim Schreiben.

  b) Wie löst man sie auf?

Weitere Entwurfsaspekte (4)

- Zu a)
  (“Wann Inkonsistenzen erkennen/auflösen?”)
  - Oft beim Schreiben, damit Lesen einfach bleibt.
  - Beispielsweise Quorum-basiert und beim Lesen:
    - Vergleich der Werte der Kopien im Lesequorum.
    - Ggf. Überschreiben bestimmter Kopien.
    - I. Allg. Verzögerung des Lesens gesamthaft durch Vergleich und zusätzliche Schreiboperationen.
  - Im Amazon-Kontext muss Schreiben aber stets möglich sein. Insbesondere Befüllen des Einkaufswagens.

Weitere Entwurfsaspekte (5)

- Zu b) (“Wie Inkonsistenzen auflösen?”)
  - Entweder System oder Anwendung.
  - System: Nur generische Policies möglich, z. B. “Last Write Wins.”.
  - Anwendung:
    - Z. B. Vereinigung von zwei unterschiedlichen Versionen des Einkaufswagens.
      D. h. keine Artikel gehen verloren.
    - Muss Anwendungsentwickler sich natürlich überlegen und implementieren.

Weitere Anforderungen: Effizienz und Konfigurierbarkeit (1)

- Performance-Anforderungen
  sollen vom 99,9-ten Perzentil erfüllt sein.

- Konfigurierbarkeit.
  Je nach Anwendung unterschiedliche Gewichtung von
  - Verfügbarkeit,
  - Konsistenz,
  - Kosteneffizienz,
  - Performance.

  Z. B. ‘Einkaufswagen’ vs. ‘Lagerverwaltung’
  (“Nur noch 3 auf Lager (mehr ist unterwegs).”):
  Wie diese Eigenschaften aus Amazon-Sicht jeweils gewichten?

Weitere Anforderungen: Effizienz und Konfigurierbarkeit (2)

Antwort:

- Einkaufswagen:
  - Verfügbarkeit maximal hoch,
  - Konsistenz hoch,
  - Kosteneffizienz nicht so hoch,
  - Performance hoch.
- Lagerverwaltung – für vorhin sichtbare Benutzersicht:
  - Verfügbarkeit, Konsistenz, Performance nicht so hoch,
  - Kosteneffizienz hoch.

Weitere Anforderung: Security

- Keine Security-Features erforderlich, da ‘nicht-feindliche’ Umgebung.

Photo by Emily Carlin

Weitere Entwurfsaspekte (6)

- *Scale-Out* (Hinzunahme neuer Knoten) soll möglich sein. Möglichst ohne Auswirkungen auf System und Betrieb.
  - D. h. ohne Systemstillstand oder gar Reimplementierung.
  - Hinterher natürlich entsprechende Beschleunigung.
- Symmetrie – kein Knoten mit Sonderrolle/besonderen Verantwortlichkeiten.
- Heterogenität – Knoten mit unterschiedlichen Fähigkeiten/ unterschiedlicher Leistungsfähigkeit sind die Norm.


Photo by evadedave

Zugrundeliegende Konzepte und Technologie: Strukturierte P2P-Systeme

- Es existieren vielfältige Konzepte/Systeme für hochgradig verteilte Datenhaltung.
  Z. B. Systeme für (legales/illegales) Filesharing.
- *Strukturierte Peer-to-Peer Systeme* (P2P-Systeme) haben hier den unmittelbarsten Bezug.
  - Verwaltung von (Schlüssel, Wert)-Paaren.
  - (put, get)-Interface.
  - Jeder Knoten ist verantwortlich für einen Ausschnitt des Schlüsselraums.
    (Ausschnitt des Schlüsselraums – Teilmenge der Menge der möglichen Schlüssel.)
  - Suchaufwand i. Allg. logarithmisch mit der Größe des Schlüsselraums.

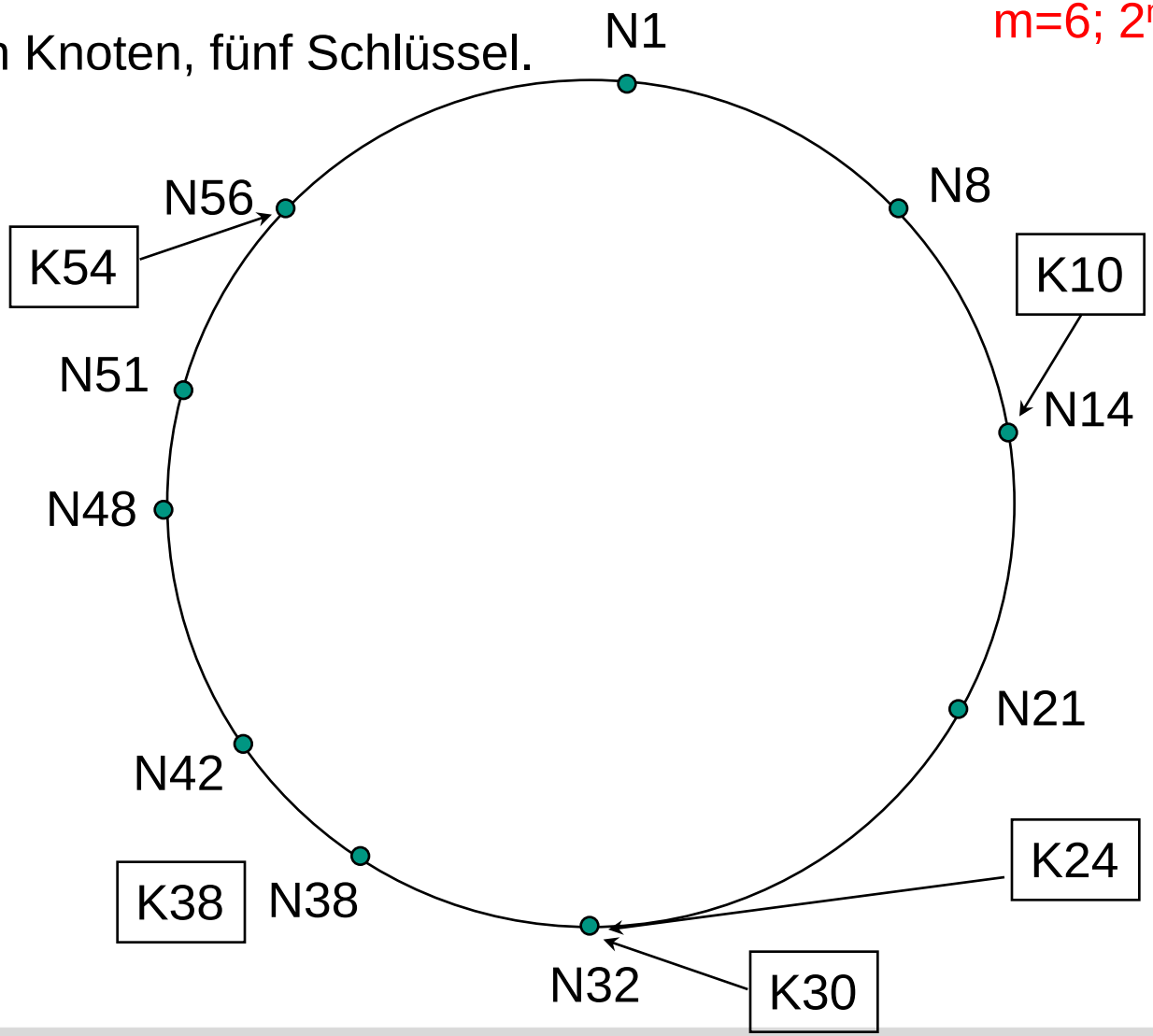
Chord – Datenorganisation (1)

- Chord:
  Prominentes Beispiel für strukturierte P2P-Systeme.

Identifier Circle

Zehn Knoten, fünf Schlüssel.

$m=6;\ 2^m=64$



Chord – Datenorganisation (2)

- Zentrale Datenstruktur:
  Identifier circle, *Chord Ring*.
- Jedes Datenobjekt hat einen Schlüssel,
  bestehend aus m Bits.
- Weise Schlüssel k
  dem (im Uhrzeigersinn) nächsten Knoten zu.

- Neuer Knoten kommt hinzu –
  er ist verantwortlich für Schlüssel,
  für die sein Vorgänger bisher verantwortlich war.
- Exit – analog.

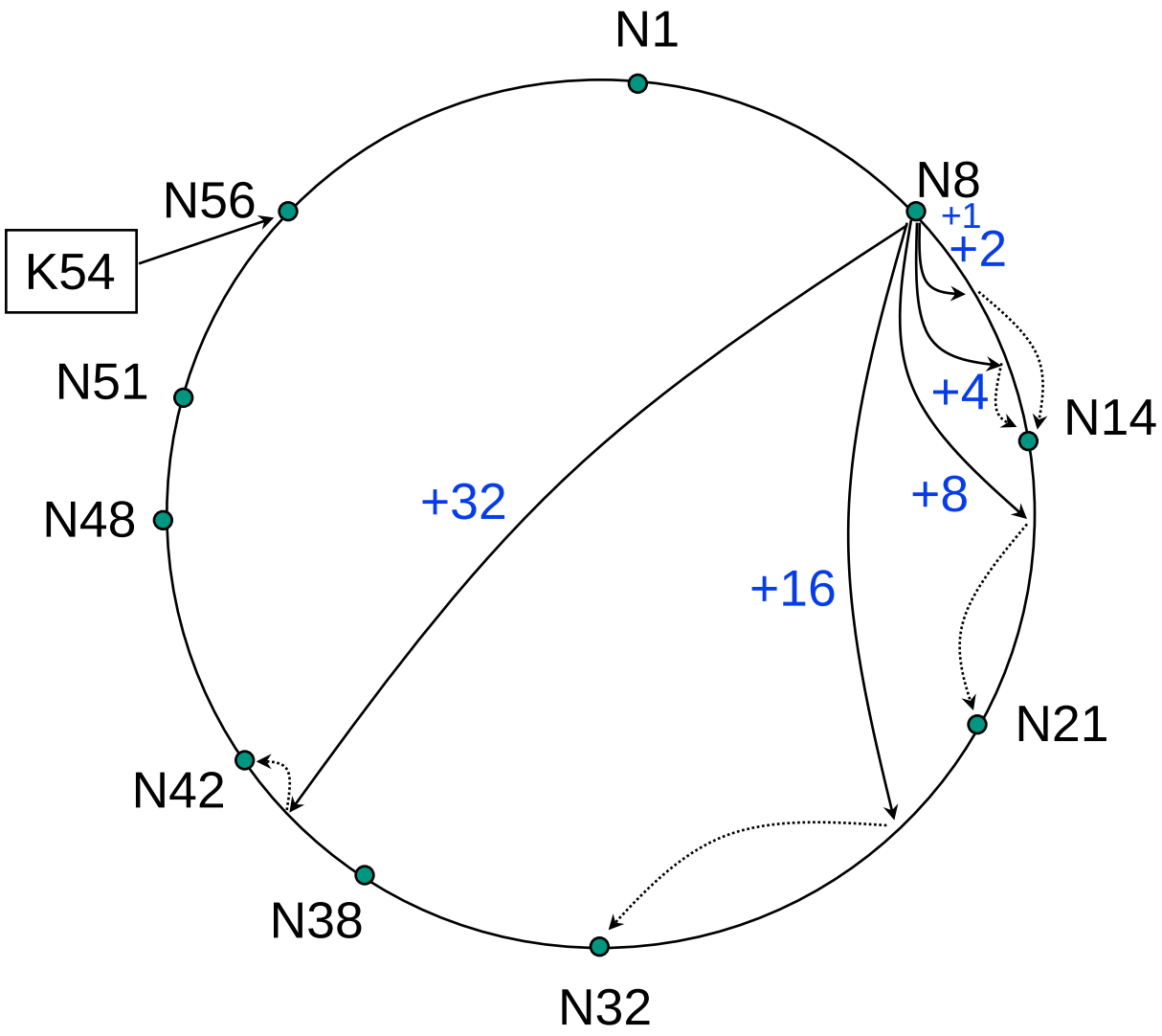
Chord – Datenorganisation (3)

- Unterscheidung zwischen *Anwendungs-* und *System-Schlüssel*. Hash-Funktion bildet Anwendungs- auf System-Schlüssel ab.
- Motivation für Hashing: Bessere Lastverteilung.

Suche

- Naive Variante – gehe zum nächsten Nachfolger, bis Objekt mit diesem Schlüssel gefunden wurde.
- Ausgefeilter:
  - Zusätzliche Routing Information, sogenannte *finger table*.
  - i-ter Eintrag in der finger table von Knoten n: successor($n+2^{i-1}$), $1 \leq i \leq m$; Berechnung erfolgt modulo 2^m.
    m ist Anzahl der Bits.
  - Nachfolger –
    ist erster Eintrag in der finger table.

Finger Table Illustration der Suche

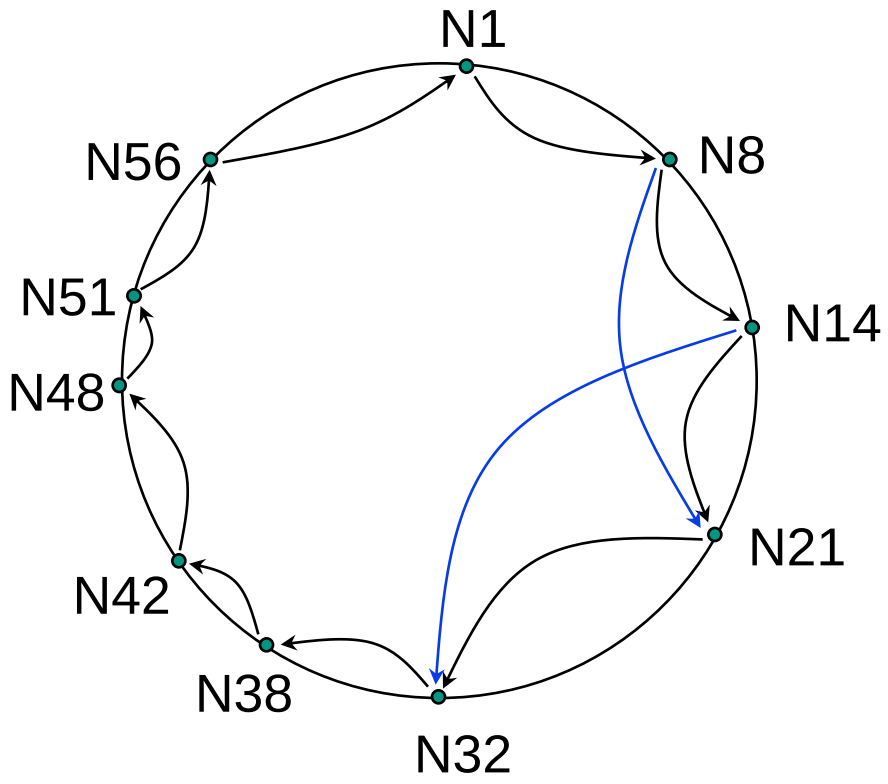


Diskussion

- Finger table ist klein;
  Größe=m (für einen Knoten).
- Suchkosten sind logarithmisch.
- Es existieren strukturierte P2P-Systeme
  mit O(1)-Suchaufwand.
  Kommt zustande durch umfangreichere Routing-Information auf jedem Knoten.

Ausfälle und Replikation

- Nachfolger-Liste der Länge r, für höhere Robustheit.
- Eine Möglichkeit für Replikation
  – Schlüssel nicht nur bei einem Knoten einfügen, sondern auch bei den k Nachfolgern. („Chained Replication“)



Heterogenität

- Knoten können unterschiedlich leistungsfähig sein.
- Unterschiedlich große Zuständigkeitsbereiche/ unterschiedliche Last.