



## Übung zur Vorlesung *Einsatz und Realisierung von Datenbanksystemen im SoSe17*

Maximilian E. Schüle (schuele@in.tum.de)  
<http://db.in.tum.de/teaching/ss17/impldb/>

### Blatt Nr. 06

#### Hausaufgabe 1

Gegeben sei eine verteilte Datenbank mit 8 Knoten, welche das *Quorum-Consensus*-Verfahren verwendet. 6 Knoten haben jeweils ein Gewicht von 1, 1 Knoten hat ein Gewicht von 2, 1 Knoten ein Gewicht von 3. Geben Sie alle gültigen Schreibquoren und zugehörige Lesequoren an.

#### Lösung:

Die Summe der Gewichte ist 11, ein Schreibquorum muss mehr als die Hälfte aller Gewichte vereinen.

- $Q_w = 6$  und  $Q_r = 6$
- $Q_w = 7$  und  $Q_r = 5$
- $Q_w = 8$  und  $Q_r = 4$
- $Q_w = 9$  und  $Q_r = 3$
- $Q_w = 10$  und  $Q_r = 2$
- $Q_w = 11$  und  $Q_r = 1$

#### Hausaufgabe 2

Zeigen Sie, dass die *write-all/read-any* Methode zur Synchronisation replizierter Daten einen Spezialfall der *Quorum-Consensus*-Methode darstellt.

- Für welche Art von Workloads eignet sich dieses Verfahren besonders gut?
- Wie werden Stimmen zugeordnet um *write-all/read-any* zu simulieren?
- Wie müssen die Quoren  $Q_w$  und  $Q_r$  vergeben werden?

Dieses Verfahren fordert einen sehr großen Aufwand beim Schreiben, aber nur minimalen Aufwand beim Lesen. Daher eignet es sich besonders gut für Workloads in denen wesentlich mehr Daten gelesen als geschrieben werden. Dies entspricht z.B. analytischen Anfragen.

Siehe Übungsbuch.

#### Hausaufgabe 3

Einen weiteren Spezialfall des *Quorum-Consensus*-Verfahrens stellt das *Majority-Consensus*-Protokoll dar. Wie der Name andeutet, müssen Transaktionen sowohl für Lese- als auch für Schreiboperationen die Mehrzahl der Stimmen einsammeln. Zeigen Sie die Konfigurierung des *Quorum-Consensus*-Verfahrens für die Simulation dieses *Majority-Consensus*-Protokolls.

Siehe Übungsbuch.

## Hausaufgabe 4

Überlegen Sie sich, welche Tupel bei der Anwendung des bloomfilterbasierten Joins in Abbildung 1 übertragen werden. Markieren Sie insbesondere, welche Tupel übertragen werden, obwohl sie keinen Joinpartner finden (sog. *false drops*). Wie kann die Anzahl dieser *false drops* verringert werden? Welche Eigenschaften sollte die Hashfunktion  $h(c)$  die bei dieser Joinbearbeitung verwendet wird erfüllen?

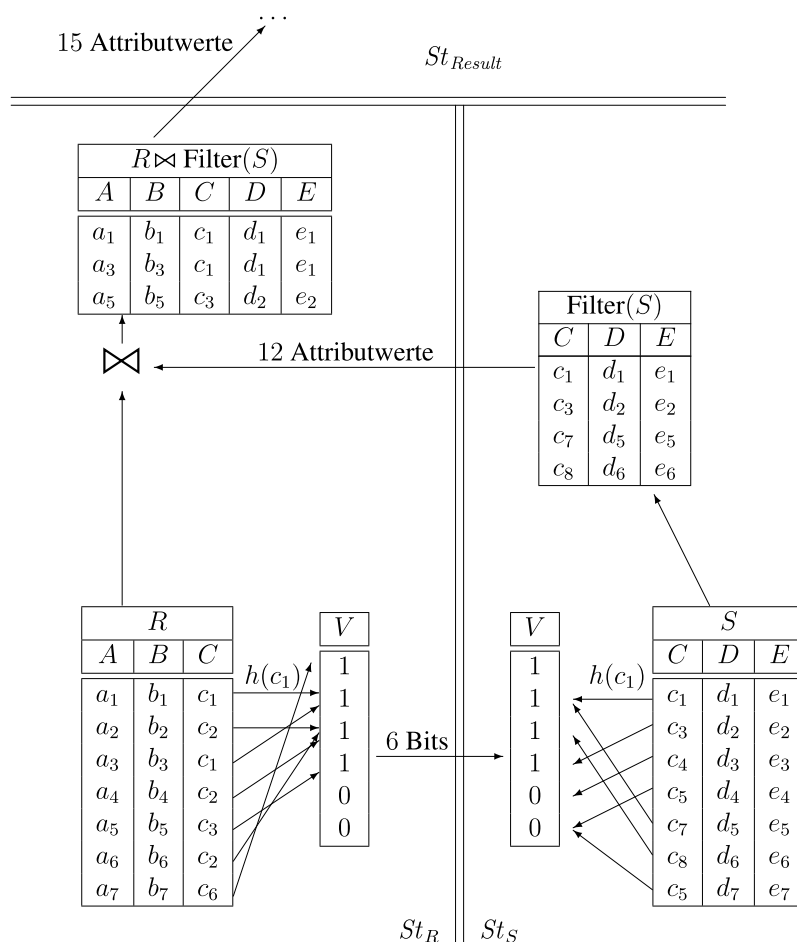


Abbildung 1: Beispiel einer verteilten Joinbearbeitung mit Bloomfilter.

- False Drops sind  $c_7$  und  $c_8$  die übertragen werden, obwohl sie keinen Joinpartner finden.
- Bei sinnvoller Hashfunktion weniger False Drops je größer der Vektor ist. Gute Hashfunktion! Mehrere Hashfunktionen zu benutzen verbessert auch die Effizienz des Bloomfilters
- Hashfunktion sollte möglichst gleichmäßig verteilen.

## Hausaufgabe 5

Zeigen Sie, dass die Suche in einem Chord-Overlaynetzwerk durch die Nutzung der FingerTabellen in maximal logarithmisch vielen Schritten zur Größe des Zahlenrings (bzw.

der Anzahl der Stationen) durchgeführt werden kann. Verwenden Sie die Suche nach K57 beginnend an Station P11 (siehe Abbildung 2) zur Illustration.

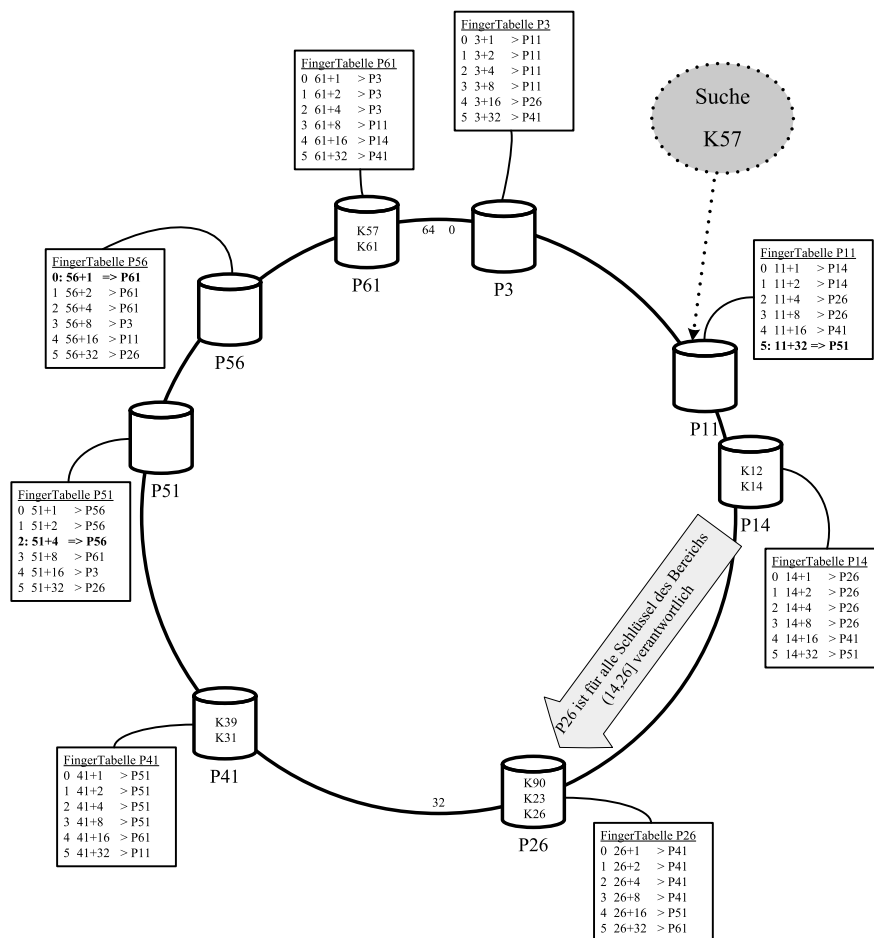


Abbildung 2: Beispiel eines Chord-Overlaynetzwerks.

Vgl. Übungsbuch.

**z.z.:** Mit jedem Sprung halbiert sich die Anzahl der Stationen.

**Beweis durch vollständige Induktion** über die Anzahl der Sprünge  $i$ .

**Induktionsanfang:** Sind wir direkt vor dem gesuchten Peer, also  $i = 0$ , müssen wir nicht mehr springen, es verbleiben  $\ln(1) = \ln(2^0) = 0$  Sprünge.

**Induktionsschritt:** Vom  $i$ -ten zum  $i + 1$ -Sprung halbiert sich die Anzahl verbleibender Stationen von  $2^{n-i}$  nach  $2^{n-i-1}$

**Induktionsvoraussetzung:** Nach  $i$  Sprüngen reduziert sich die Anzahl verbleibender Stationen auf  $2^{n-i}$ .

Wir wissen, dass sich innerhalb eines Intervalls  $[p + 2^i, p + 2^{i+1})$  genau  $2^i$  Stationen befinden, da wir sonst bereits zum nächsten Peer gesprungen wären. Damit haben wir bereits den Abstand auf  $2^{n-i}$  Stationen halbiert (Induktionsvoraussetzung). Jetzt springen wir in das Intervall  $[(p + 2^i) + 2^{i-1}, (p + 2^i) + 2^i)$  mit  $2^{i-1}$  möglichen Stationen. Für den maximalen Abstand zum Ziel-Peer verbleiben nach dem  $i + 1$ -ten Sprung maximal  $2^{n-(i+1)}$  Stationen, wenn wir  $n - (i + 1)$  mal springen. Damit halbiert sich die Zahl verbleibender Sprünge auf  $2^{n-(i+1)} = 2^{n-i-1}$

## Hausaufgabe 6

Skizzieren Sie die Vorgehensweise beim Hinzufügen eines neuen Peers im Chord Netzwerk. Als Beispiel nehmen Sie die Hinzunahme eines Peers P33 in dem Beispiel-Netzwerk aus Abbildung 2.

Vgl Übungsbuch: Zuerst kontaktiert der neue Peer einen bekannten Peer im Chord Netzwerk. Dieser leitet ihn mittels der Hash-Funktion an den direkten Vorgänger des neuen Peers weiter. Von diesem übernimmt er die Fingertabelle und sein direkter Nachfolger wird ihm bekannt. Von diesem wiederum übernimmt er alle relevanten Daten, für die er nun zuständig ist.

Nach dem Einfügen eines neuen Peers P<sub>k</sub> in ein Chord-Netzwerk müssen die Fingertabellen anderer Peers, „die bislang P<sub>j</sub> referenziert haben, jetzt möglicherweise auf P<sub>k</sub> geändert werden“<sup>a</sup>. Dies passiert allerdings nicht beim Einfügen, sondern beim Stabilisieren<sup>b</sup>. Der Stabilisierungsalgorithmus wird in 30s Abständen aufgerufen<sup>c</sup>, bzw. beim Einfügen eines neuen Peers<sup>d</sup>. Beim Einfügen wird nur der direkte Vorgänger über seinen neuen Nachfolger informiert.

---

<sup>a</sup>Kemper, Eickler: Datenbanksysteme, 10. Auflage

<sup>b</sup>[https://en.wikipedia.org/wiki/Chord\\_\(peer-to-peer\)#Pseudocode](https://en.wikipedia.org/wiki/Chord_(peer-to-peer)#Pseudocode)

<sup>c</sup><http://graal.ens-lyon.fr/~abenoit/reso06/papier/chord1.pdf>

<sup>d</sup>[http://sarwiki.informatik.hu-berlin.de/Chord#Eintritt\\_neuer\\_Knoten](http://sarwiki.informatik.hu-berlin.de/Chord#Eintritt_neuer_Knoten)

## Hausaufgabe 7

Zum CAP-Theorem hieß es in der Vorlesung, dass in verteilten Systemen nur zwei der drei „Wünsche“ (Konsistenz, Verfügbarkeit und Partitionstoleranz) gleichzeitig erfüllbar sind.

Welche der drei Kombinationen CA, CP, und AP sind jedoch sehr ähnlich?

### Lösung:

Das CAP-Theorem beschreibt das Problem, dass bei einem verteilten System nur zwei der drei wichtigen Eigenschaften Konsistenz (**C**onsistency), Verfügbarkeit (**A**vailability), und Partitionstoleranz (**P**artition tolerance) eingehalten werden können. Dementsprechend sollte es drei verschiedene Arten von Systemen geben:

**CA:** Konsistent und verfügbar, aber nicht partitionstolerant.

**CP:** Konsistent und partitionstolerant, aber nicht verfügbar.

**AP:** Verfügbar und partitionstolerant, aber nicht konsistent.

Wie Daniel Abadi in einem Blogartikel<sup>1</sup> beschreibt, gibt es jedoch keinen praktischen Unterschied zwischen CP- und CA-Systemen. Ein CP-System gibt die Verfügbarkeit auf, wenn das Netzwerk partitioniert ist (der Definition nach könnte es auch nie verfügbar sein; ein solches System macht jedoch wenig Sinn). CA-Systeme tolerieren per Definition Netzwerkpartitionen nicht. Aber was bedeutet das? In der Praxis ist es so, dass auch bei einem CA-System die Verfügbarkeit verloren geht, sobald das Netzwerk partitioniert ist. Auf diese Weise ist sichergestellt, dass bei Wiederherstellung des Netzwerk-Konnektivität wieder CA erfüllt werden kann.

Es gibt daher nur zwei Arten von Systemen: CP/CA und AP. Es stellt sich nur die folgende Frage: Wie reagiert das System, wenn das Netzwerk partitioniert wird? Entweder das System gibt die Verfügbarkeit auf (CP/CA) oder die Konsistenz (AP).

---

<sup>1</sup><http://dbmsmusings.blogspot.de/2010/04/problems-with-cap-and-yahoos-little.html>

Als Schlussbemerkung, sei noch darauf hingewiesen, dass CAP nichts über die Performanz des Systems aussagt. Es kann Sinn machen, sowohl die Verfügbarkeit als auch die Konsistenz einzuschränken, um eine höhere Leistungsfähigkeit zu erreichen (z.B. eine geringere Latenz). Mehr zu diesem Thema findet sich in dem oben erwähnten Blogeintrag von Daniel Abadi.