Epidemische Algorithmen

Seminar Informationsmanagement im Web

Vortrag von Konrad Rieck

Sommer 2003

Institut für Informatik – Fachbereich Mathematik und Informatik

Freie Universität Berlin

Überblick

- Motivation und Hintergrund
  - Replizierte Datenbanken
  - Synchronisationsverfahren
- Epidemische Algorithmen
  - Epidemische Ausbreitung
  - Ereignisorientierte Algorithmen
  - Transaktionsorientierte Algorithmen
- Schluss und Ausblick

Motivation und Hintergrund

Motivation

- Probleme von großen Diensten
  - Enorme Last für ein Rechnersystem
  - Ausfall von Rechnersystemen
  - Unzuverlässige Netzanbindungen
- Lösungsmöglichkeit
  - Lastverteilung auf autonome Systeme
  - Datenverteilung mit Redundanz
  - Fehlertolerante Algorithmen

Verteilte Datenbank

- Autonome Rechnersysteme
  - Unabhängige Ressourcen
    - Netzanbindung
    - Stromversorgung
    - Hardware
- Verbindung über Kommunikationsnetz
- Verteilung des Datenbestandes
  - Fragmentierungsschema
  - Verteilungsschema

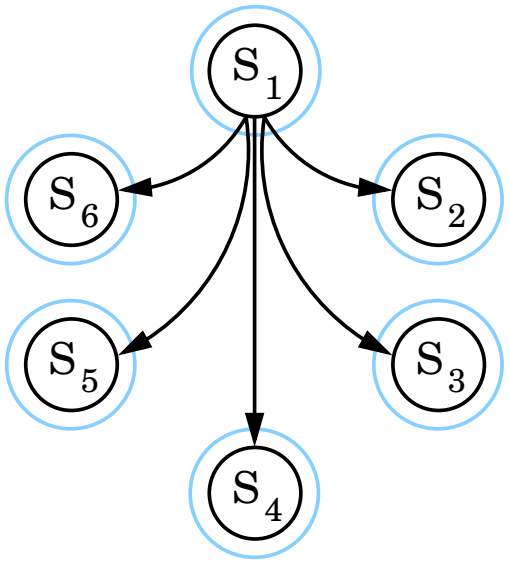
Replizierte Datenbank

- Spezialfall einer verteilten Datenbank
  - Keine Fragmentierung der Relationen
  - Verteilung mit maximaler Redundanz
- Aufbau und Funktionsweise
  - n Stationen $S_1, S_2, \ldots, S_n$
  - Jede Station verwaltet *Replikat*
  - Updates erfolgen lokal und dezentral
  - Synchronisation der Replikate

Synchronisationsverfahren

- Entscheidend für die Qualität der Datenbank
- Kompromiss zwischen ...
  - ... Konsistenz der Replikate
  - ... Leistungsfähigkeit der replizierten Datenbank
- Beispiele
  - Unicast oder Write-All Locking
  - Anti-Entropie Verfahren
  - Epidemische Algorithmen

Unicast oder Write-All Locking



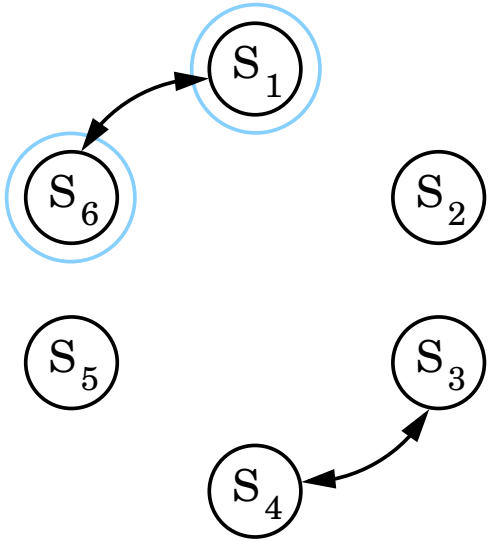
1. Lokales Update an S_1
2. Kontakt und Sperrung
3. Transfer des Updates

\+ Replikate immer konsistent

\- Alle Stationen global bekannt und verfügbar

\- Sperrung beeinträchtigt Leistungsfähigkeit

Anti-Entropie Verfahren



1. Lokales Update an S_1
2. Datenbankabgleich zufälliger Stationen
3. Update wird indirekt propagiert

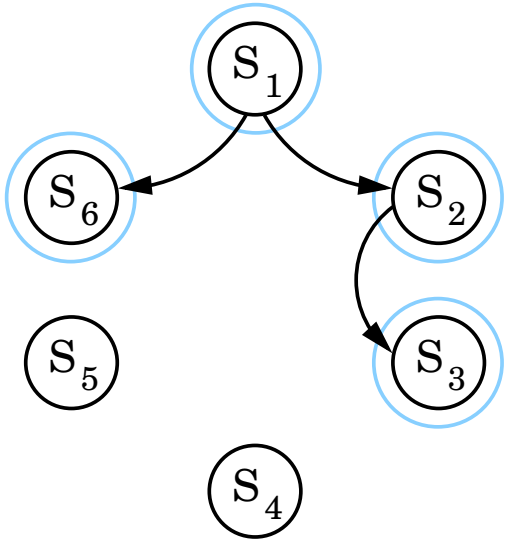
\+ Sperrungen nur an einzelnen Stationen

\+ Nicht alle Stationen global bekannt oder verfügbar

\- Replikate zeitweise inkonsistent

\- Datenbankabgleich beeinträchtigt Leistungsfähigkeit

Epidemische Algorithmen



1. Lokales Update an S_1
2. Infektion einer Station
3. Epidemische Ausbreitung

\+ Sperrungen nur an einzelnen Stationen

\+ Nicht alle Stationen global bekannt oder verfügbar

\+ Kein Datenbankabgleich, nur Transfer von Updates

\- Replikate zeitweise inkonsistent

Epidemische Algorithmen

Epidemische Ausbreitung (1)

- Informationsausbreitung ähnlich ...
  - ... einer Epidemie
    - Virus springt von Wirt zu Wirt
    - Virus $\hat{=}$ Nachricht über Update
    - Wirt $\hat{=}$ Station
- Lokales Update oder Nachricht über Update an S_i
- Nachricht über Update wird
  - ... gespeichert im lokalen Log L_i
  - ... versendet an zufällige Station S_j

Epidemische Ausbreitung (2)

- Stationen in einem von drei Zuständen
  1. Empfänglicher Zustand
     - Zustand *vor* einer Epidemie.
  2. Infektiöser Zustand
     - Zustand nach lokalem Update oder nach Kontakt mit anderer infektiöser Station.
  3. Deaktivierter Zustand
     - Zustand ausgelöst mit Wahrscheinlichkeit $\frac{1}{k}$ durch Kontakt mit bereits infizierter Station

Ausbreitungsstärke

- Ausbreitung hängt von Deaktivierungswahrscheinlichkeit $\frac{1}{k}$ ab.
- Anzahl s der nicht-infizierten Stationen

$$s = e^{-(k+1)(1-s)}$$

- s exponentiell abhängig von k
- Beispiel: $k = 1 \Rightarrow s = 0.20$ und $k = 2 \Rightarrow s = 0.06$
  - Zu kleines k: Stationen werden nicht erreicht
  - Zu großes k: Unnötige Kommunkationslast
- Keine Garantie für vollständige Ausbreitung

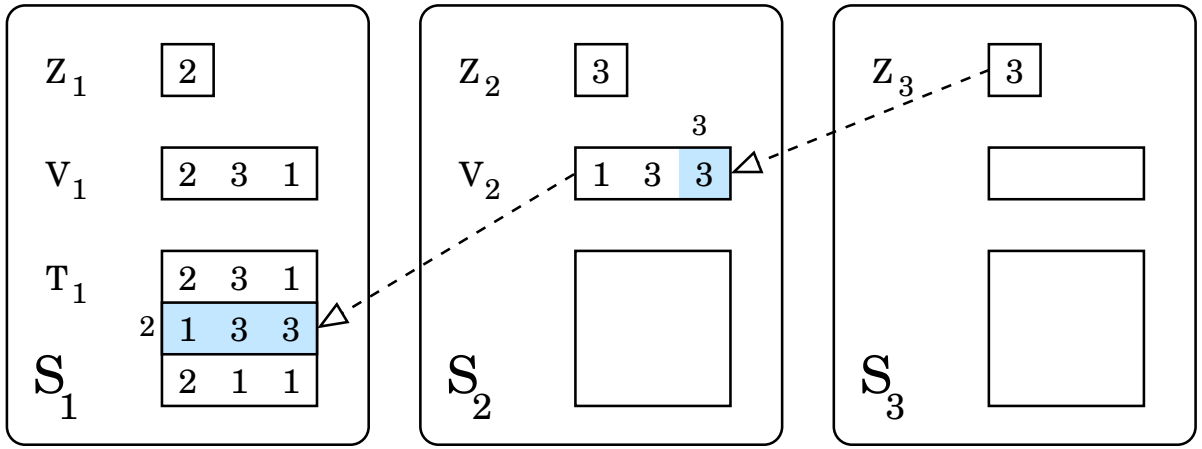
Ereignisorientierte Algorithmen

- Problem der epidemischen Ausbreitung
  - Verlust des zeitlichen Zusammenhangs
    - S_i sendet Updates U_1, U_2, U_3
    - S_j empfängt Updates U_3, U_2, U_1
- Lösungsmöglichkeit
  - Einführung eines Ereignismodels $\langle E, \rightarrow \rangle$
  - Ereignismenge E, z.B. $E = \{\text{READ}, \text{WRITE}\}$
  - „$\rightarrow$“ Relation, z.B. $e \rightarrow f \quad \equiv \quad e$ vor f eingetreten.

Realisierung der „$\rightarrow$“ Relation

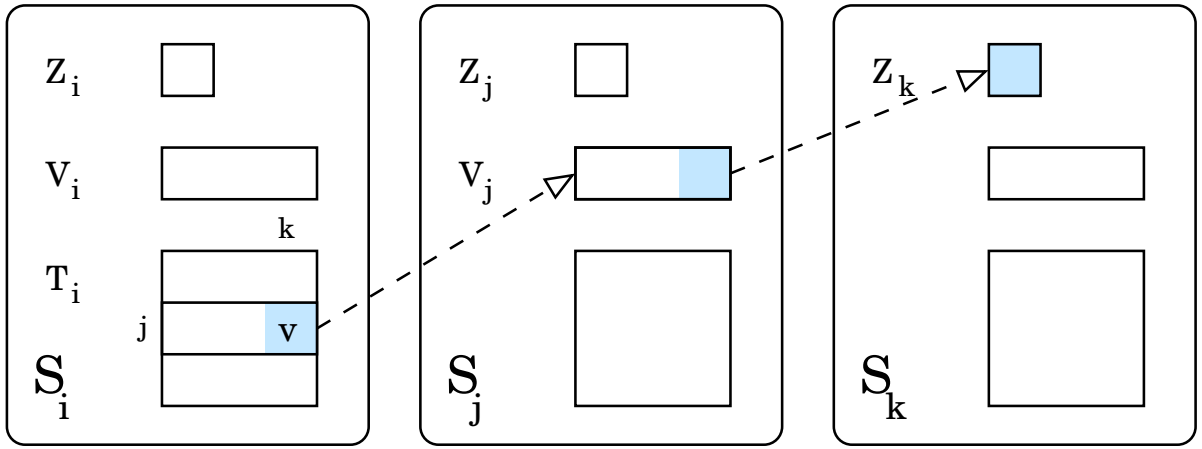
- „$\rightarrow$“ Relation verlangt Zeitmessung
- Jede Station verwaltet eigene lokale Zeit
  - Lokale Zeit unabhängig von globaler Zeit
  - Änderung der lokalen Zeit ereignisorientiert
- Lokale Zeit ermöglicht ...
  - ... Zeitstempel von Ereignissen $\mathsf{TS}(e)$
  - ... $\forall e, f \in E: \ e \rightarrow f \Leftrightarrow \mathsf{TS}(e) < \mathsf{TS}(f)$
  - ... Speicherung von Zeiten (Vektoruhr V_i)
  - ... Wissen über Speicherung (Matrixuhr T_i)

Vektor- und Matrixuhren



- Station S_i verwaltet unabhängige lokale Zeit Z_i
- Station S_i verwaltet Vektoruhr $V_i = (Z_1, Z_2, Z_3)$
- Station S_i verwaltet Matrixuhr $T_i = (V_1, V_2, V_3)^T$
- Es gilt $\forall i, j, k : \ Z_k \geq V_j[k] \geq T_i[j, k]$

Interpretation der Matrixuhr



- Station S_i kennt S_j's Vektoruhr
- Station S_j verwaltet letzte Zeit von S_k

$T_i[j,k] = v \quad \Rightarrow$ S_i weiß, dass S_j alle Ereignisse von S_k bis zur Zeit v erhalten hat.

Das HasRecvd Prädikat

- Sei t ein Ereignis,
  $S_k = \text{SITE}(t)$ die Station an der t aufgetreten ist,
  $\text{TS}(t)$ die Vektorzeit als t aufgetreten ist und
  $\text{TS}_k(t)$ die letzte bekannte Zeit von Station S_k

$$\text{HASRECVD}(T_i, t, S_j) \equiv T_i[j,k] \geq \text{TS}_k(t)$$

- $\text{HASRECVD}(T_i, t, S_j)$ ist wahr, wenn S_j das Ereignis t erhalten hat.
- $\text{HASRECVD}(T_i, t, S_j)$ wird als *Timetable Eigenschaft* bezeichnet.

Ein ereignisorientierter Algorithmus

- Station S_i besitzt Log L_i
- Log L_i enthält Nachrichten über Updates
- Station S_i wählt zufällig Station S_j
- Station S_i sendet alle Nachrichten t über Updates an S_j für die gilt HASRECVD(T_i, t, S_j) ist falsch.
- Station S_j erneuert Replikat
- Station S_j trägt erhaltene Nachrichten in Log L_j

Transaktionsorientierte Algorithmen

- Problem der ereignisorientierten Algorithmen
  - Konflikte werde nicht berücksichtigt
    - S_i erhält Transaktion $t_1 \equiv \textsc{Read}(x) \rightarrow \textsc{Write}(y)$
    - S_j erhält Transaktion $t_2 \equiv \textsc{Read}(y) \rightarrow \textsc{Write}(x)$
- Lösungsmöglichkeiten
  - Pessimistischer epidemischer Algorithmus
    - Optimistischer epidemischer Algorithmus
    - Epidemischer Quorum-Algorithmus

Pessimistischer Algorithmus

Pessimistische Annahme: Konflikte häufig

- Erweiterung des ereignisorientierten Algorithmus
- Updates enthalten Transaktionen, nicht Ereignisse
- Zu jeder Transaktion t werden übertragen ...
  - ... Zeitstempel TS(t) (Vektoruhr von Station)
  - ... Menge der gelesenen Objekte RS(t)
  - ... Menge der geschriebenen Objekte WS(t)

Identifikation von Konflikten

- Transaktionen t und t' im Konflikt, wenn ...
  1. Transaktionen greifen auf gleiche Objekte zu
     - $\mathsf{RS}(t) \cap \mathsf{WS}(t') \neq \emptyset$ *oder*
     - $\mathsf{WS}(t) \cap \mathsf{WS}(t') \neq \emptyset$ *oder*
     - $\mathsf{WS}(t) \cap \mathsf{RS}(t') \neq \emptyset$
  2. Transaktionen wurden gleichzeitig ausgeführt
     - S_i erhielt zuerst t: $\mathsf{TS}_i(t) > \mathsf{TS}_i(t')$ *und*
     - S_j erhielt zuerst t': $\mathsf{TS}_j(t') > \mathsf{TS}_j(t)$
     - Inkompatibilität: $\mathsf{TS}(t) <> \mathsf{TS}(t')$

Das Aborted Prädikat

- ABORTED Prädikat

$$\text{ABORTED}(t, S_j) \equiv \exists t' \in L_j : \text{TS}(t) <> \text{TS}(t') \wedge \left(\begin{array}{ll} \text{RS}(t) \cap \text{WS}(t') \neq \emptyset & \vee \\ \text{WS}(t) \cap \text{WS}(t') \neq \emptyset & \vee \\ \text{WS}(t) \cap \text{RS}(t') \neq \emptyset & \end{array} \right)$$

- Station S_j bricht Transaktion t und t' ab und sendet Abort-Nachricht, wenn $\text{ABORTED}(t, S_j)$ wahr ist.

Das Commit Prädikat

- Station S_i führt Commit für Transaktion aus, wenn ...
  1. Alle Stationen das Update erhalten haben
  2. Keine Station eine Abort-Nachricht gesendet hat
- COMMIT Prädikat

$$\text{COMMIT}(t, S_i) \equiv \forall j : \text{HASRECVD}(T_i, t, S_j) \wedge \neg\text{ABORTED}(t, S_i)$$

- Station S_i führt Commit für Transaktion t aus, wenn $\text{COMMIT}(t, S_i)$ wahr ist.

Verarbeitung einer Transaktion

- Station S_i erhält Update der Transaktion t
- Station S_i setzt Schreibsperre auf WS(t)
- Station S_i wartet
- Entweder COMMIT(t, S_i) wird wahr ...
  - Transaktion t wird übernommen und in L_i vermerkt
- ... oder ABORTED(t, S_j) wird wahr
  - Transaktionen t und t' werden abgebrochen
  - Abort-Nachricht wird versendet
- Sperre auf WS(t) wird gelöst

Optimistischer Algorithmus

- Beobachtung:
  - Pessimistischer Algorithmus langsam auf Grund längerer Sperrzeit

Optimistische Annahme: Konflikte selten

- Variante des pessimistischen Algorithmus
- Zu jeder Transaktion t wird gespeichert ...
  - Menge aller Transaktionen, die von t lesen READFROM(t).

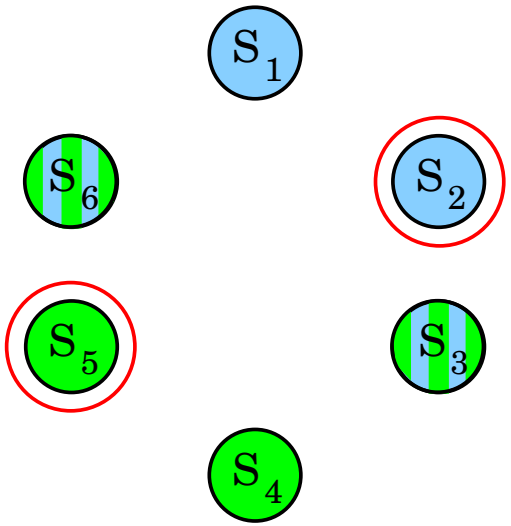
Verarbeitung einer Transaktion

- Station S_i erhält Update der Transaktion t
- Station S_i wartet
- Entweder COMMIT(t, S_i) wird wahr ...
  - Transaktion t wird übernommen und in L_i vermerkt
- ... oder ABORTED(t, S_j) wird wahr
  - Transaktionen t und t' werden abgebrochen
  - Transaktionen READFROM(t) und READFROM(t') werden abgebrochen. (Abbruch-Kaskade)
  - Abort-Nachricht wird versendet

Epidemischer Quorum-Algorithmus

- Beobachtung:
  - Pessimistischer Algorithmus bricht bei einem Konflikt mindestens zwei Transaktionen ab.
- Variante des pessimistischen Algorithmus
- Verbesserung:
  - Votum welche Transaktionen abgebrochen bzw. übernommen werden soll.
  - Verwendung von Quoren und Antiquoren der Stationen

Quoren und Antiquorum



1. Mehrheitsquorum 1
2. Mehrheitsquorum 2
3. Antiquorum

- *Quoren* sind Teilmengen, die so gewählt werden, dass jede Schnittmenge von zwei Quoren nicht leer ist.
- Ein *Antiquorum* ist jede Teilmenge, die ein Element aus jedem Quorum enthält.

Verarbeitung einer Transaktion

- Station S_i erhält Update der Transaktion t
- Station S_i setzt Schreibsperre auf WS(t)
- Stationen votieren bei Konflkit mit *Ja*/*Nein*

1. Station S_i erhält *Ja*-Quorum, so wird Commit ausgeführt
2. Station S_i erhält *Nein*-Antiquorum, so wird t abgebrochen und Abort-Nachricht versendet
3. Station S_i erhält Abort-Nachricht, so wird t abgebrochen

- Sperre auf WS(t) wird gelöst

Schluss und Ausblick

Schluss

- Epidemische Algorithmen geeignet zur leistungsfähigen Synchronisation von replizierten Datenbanken
- Epidemische Algorithmen können in unzuverlässigen Netzen eingesetzt werden
- Synchronisation von Ereignissen und sogar Transaktionen möglich. Berücksichtigung von Konflikten
- Verschiedene Algorithmen können entsprechend der Umgebung ausgewählt werden

Vergleich der Algorithmen

- Optimistischer Algorithmus (1) effizient in Umgebung mit wenigen Konflikten
- Quorum-Algorithmus (2) effizienter als pessimistischer, da weniger Transaktionsabbrüche
- Quorum-Algorithmus effizient in Umgebung mit häufigen Konflikten
- Beide Algorithmen (1 & 2) effizienter als Unicast-Verfahren
- Beide Algorithmen (1 & 2) geben keine Garantie auf vollständige Ausbreitung

Ausblick

- Epidemische Algorithmen könnten auch replizierte Fragmente synchronisieren
  - Mehr Ausfallsicherheit für verteilte Systeme mit fragmentierten Daten, z.B. P2P
- Kombination aus Anti-Entropie und epidemischen Algorithmen garantiert vollständige Ausbreitung
  - Anti-Entropie Verfahren synchronisiert in großen Intervallen das System

Fragen?