Update Propagation Protokolle In Replizierten Datenbanken

Michael Kropfberger

michael.kropfberger@gmx.net

January 31, 2000

Einführung und Aufbau

- verschiedene Ansätze von Update Propagation Protokollen
- “Eager” Protokoll
- “Lazy” Protokolle
  - Lazy-Master Replikation
  - Replikationsgraphen-basierte Protokolle mit RGTest
  - Copygraph-basierte Protokolle

Ideale Ziele eines Replikationsschemas

Verfügbarkeit und Skalierbarkeit durch Replikation bei Erhaltung von Stabilität

Mobilität erlaubt Mobilgeräten (Handys, Laptops) lesenden und schreibenden Zugriff auf den Datenbestand, während sie offline sind

Serialisierbarkeit bei beliebiger Ausführungsreihenfolge von globalen Transaktionen

Konvergenz an einen überall gültigen Datenbankzustand ohne kurzzeitige Diskrepanzen

Transaktionsvarianten

Einzelknoten Transaktion

write A
write B
write C
Commit

"Eager" Transaktion auf drei Knoten

write A		
	Write A	
		Write A
write B		
	Write B	
		Write B
write C		
	Write C	
		Write C
Commit		
	Commit	
		Commit

"Lazy" Transaktion auf drei Knoten (3 Einzeltransaktionen)

write A
write B
write C
Commit

write A
write B
write C
Commit

write A
write B
write C
Commit

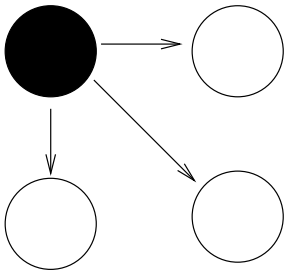
"Eager" Protokoll

+ Serialisierbarkeit mit Hilfe von einfachen globalen Locking-Algorithmen
- viele globale Locks zu jedem Replikationsknoten
- Deadlockgefahr steigt mit Anzahl Knoten kubisch an! $\rightsquigarrow$ schlechte Skalierbarkeit
- keine Möglichkeit für Mobilgeräte, offline zu gehen

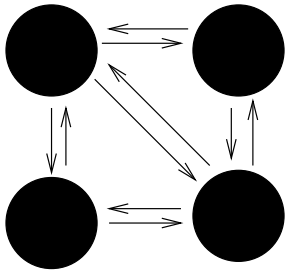
“Lazy” Protokolle

Möglichkeiten für Update-Propagation:

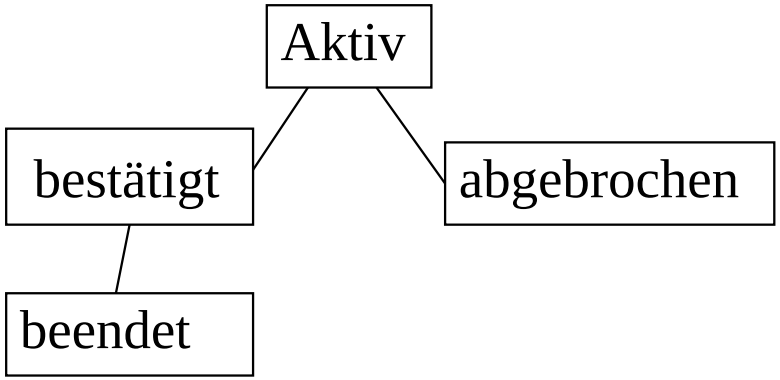
Masterknoten



gleichberechtigte Gruppe



Zustandsübergangsdiagramm:



Aktuelle Kommerzielle Produkte:

- Option für primitivste "lazy" Replikation
- lokale Abarbeitung und ungeprüfte Update Propagation
- nicht serialisierbar! $\rightsquigarrow$ Abgleichungsregeln (zB. Zeitstempel)

Lazy-Master Replikation:

- jedes Datenobjekt hat Masterknoten
- bei Zugriff Read und Write Locks zu Masterknoten
- Update wird vom Masterknoten aus weitergeleitet

Zwei-Schichten Replikation:

- zwei Rechnergruppen: mobile Knoten und Basisknoten
- mobile Knoten führen Transaktionen nur lokal aus
- beim Andocken wird nomal mit Lazy-Master propagiert

Strategien bei N Knoten

Weiterleitung vs. Eigentum	Lazy	Eager
Gruppe	N Transaktionen N Objekteigentümer	eine Transaktion N Objekteigentümer
Master	N Transaktionen ein Objekteigentümer	eine Transaktion ein Objekteigentümer
Zwei-Schichten	N+1 Transaktionen, ein Objekteigentümer, lokale Versuchsupdates, Eager Updates auf die Basisknoten	

Replikationsgraphen-basierte Protokolle

Virtuelle Knoten:

- angepasstes RDBMS für virtuelle Knoten auf einem physischen
- Transaktion i auf Knoten k hat VK_i^k, $\exists\ VK_i^k = VK_j^k$

Lokalitätsregel pro Transaktion $\exists$ ein VK_i^k pro Knoten

Vereinigungsregel Zugriff auf gleiches Datenobjekt
$\leadsto VK_i^k = VK_j^k$

Spaltungsregel bei Zustand **abgebrochen** oder **beendet**
Löschung der von T_i benutzen Datenobjekte aus VK_i^k und vereinigten VK_j^k, ggf. Löschung von VK_i^k

Replikationsgraphen-basierte Protokolle

Replikationsgraph:

- für jede Transaktion T_i existiert eine Kante (T_i, VK_i^k) zu allen virtuellen Knoten der involvierten physischen Knoten k.
- jede Operation kann eine der drei Regeln auslösen
- ein zu jedem Zeitpunkt azyklischer Replikationsgraph ist immer global serialisierbar
- Text auf Azyklität (RGTest)

Pessimistischer Ansatz RGTest bei jedem Read oder Write

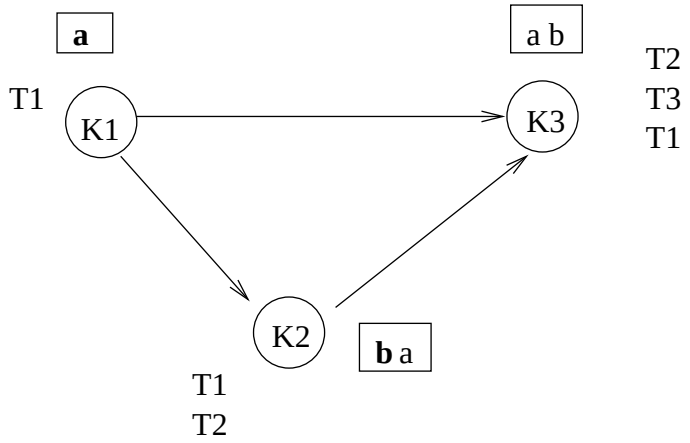
Optimistischer Ansatz RGTest erst beim lokalen Commit

Copygraph-basierte Protokolle

- ein Masterknoten und $0 \ldots n$ Replikationsknoten pro Datenobjekt
- Backedges führen zu einem zyklischen Graphen

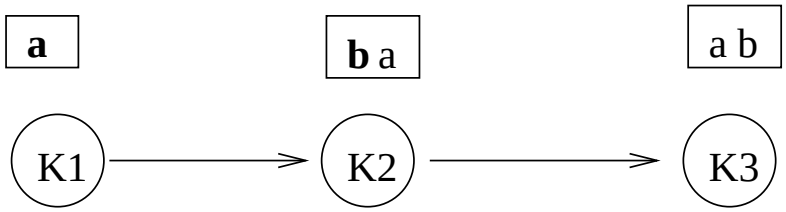
nicht serialisierbares Beispiel:

- primäre Transaktionen: $T_1^1[w(a)],\ T_2^2[r(a), w(b)],\ T_3^3[r(a), r(b)]$
- falsche Ausführung: $T_1^1[w(a_2)],\ T_1^2[w(a_2)],\ T_2^2[r(a_2), w(b_2)],$ $T_2^3[w(b_2)],\ T_3^3[r(a_1), r(b_2)],\ T_1^3[w(a_2)]$



DAG(WT) (DAG ohne Zeitstempel)

- Baum mit K_l als Unterknoten von K_k, wenn K_l eine Replikation von K_k besitzt
- Updates werden entlang des Baums weitergegeben und dort FCFS exekutiert
- Bsp: Updates von $T_1[w(a_2)]$ noch vor T_2 zum Knoten K_3: $T_1^1[w(a_2)]$, $T_1^2[w(a_2)]$, $T_1^3[w(a_2)]$, $T_2^2[r(a_2), w(b_2)]$, $T_2^3[w(b_2)]$, $T_3^3[r(a_2), r(b_2)]$
- Nachteil: Nachrichten/Updates gehen auch an Kinder ohne Replikas!



DAG(T) (DAG mit Zeitstempel)

Updates nur noch entlang des Copygraphen direkt zu den Kindern
Generierung der Zeitstempelvektoren:

- durch Azyklität des Copygraphen $\exists$ totale Ordnung über Knoten
- gegen "Verhungern" werden Epochen eingeführt

Somit gilt bei $K_1 < K_2 < K_3$

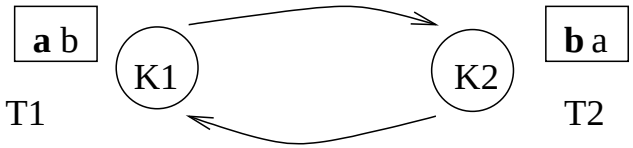
- $E_1(K_1,1) < E_1(K_1,1)(K_2,1)$
- $E_1(K_1,1)(K_3,1) < E_1(K_1,1)(K_2,1)$
- aber auch $E_1(K_1,1)(K_2,1) < E_2(K_1,1)(K_3,1)$
- $E_1(K_1,1)(K_2,1) < E_1(K_1,1)(K_2,2)$

Details von DAG(T)

- jeder Knoten speichert Zeitstempelvektor über alle vorhergehenden Knoten des Copygraphen und sich selbst
- bei Commit einer lokalen Transaktion $\rightsquigarrow INC(lokalerZS_k)$
- verschicke Updates + Zeitstempelvektor an Replikationsknoten
- $\rightsquigarrow$ jeder Knoten weiß nun über Serialisierungsreihenfolge der eingehenden Transaktionen bescheid
- Bsp: T_1 hat Zeitstempel $E_1(K_1, 1)$, T_2 hat $E_1(K_1, 1)(K_2, 1)$ K_3 führt somit T_3 aus, da T_1 ein Präfix von T_2 ist

BackEdge Protokoll

- Bsp. zyklischer Copygraph: $T_1^1[w(a), r(b)]$ und $T_2^2[w(b), r(a)]$



- hybrides Replikationsprotokoll
- kann mit zyklischen Copygraphen umgehen
- BackEdges mit "Eager" Replikation (mit globalen Locks)
- Rest als azyklischer Graph mit zB DAG(WT)
- erhöhte Deadlockgefahr, degeneriert aber ohne BackEdges zu normalem DAG(WT)

Zusammenfassung

verschiedene Ansätze von Update Propagation Protokollen
"Eager" vs. "Lazy" Protokolle

Lazy-Master globale Locks auf Masterknoten, "lazy" an Replika

Replikationsgraph Verwaltung virtueller Knoten, RGTest

Copygraph keine globalen Locks auf Masterknoten, Copy-Baum oder Zeitstempelvektoren

BackEdge hybrider Ansatz, BackEdges "eager", restlicher DAG "lazy"

Zukunftsaussichten:

- mobile Computing, Teleworking, WWW-Mirrors, ...