Verteilte Systeme

6. Konsistenz und Replikation

Sommersemester 2011

Institut für Betriebssysteme und Rechnerverbund

TU Braunschweig

Dr. Christian Werner
– Bundesamt für Strahlenschutz –

Überblick

- Ziele der Replikation
- Unterschiedliche Replikationsanforderungen
- Replikationsmodelle
  - Datenzentriert
  - Client-zentriert
- Verteilungsprotokolle
- Konsistenzprotokolle
- Zusammenfassung

INSTITUT FÜR BETRIEBSSYSTEME UND RECHNERVERBUND
Prof. Dr.-Ing. L. Wolf | Prof. Dr. S. Fekete

Sinn der Replikation

- Replikation bedeutet das Halten einer oder mehrerer Kopien eines Datums
- Ein Prozess, der auf dieses Datum zugreifen will, kann auf jede beliebige Replika zugreifen.
- Im Idealfall erhält er immer das gleiche Ergebnis.
- Was also erreicht werden muss, ist die *Konsistenz* der Kopien – wobei unterschiedliche Anwendungen unterschiedliche Anforderungen an die Striktheit der Konsistenz haben.

INSTITUT FÜR BETRIEBSSYSTEME UND RECHNERVERBUND
Prof. Dr.-Ing. L. Wolf | Prof. Dr. S. Fekete

Ziele der Replikation

- Zwei Ziele
  - Steigerung der Verlässlichkeit eines Dienstes bzw. der Verfügbarkeit eines Datums
    - Wenn ein Replikat nicht mehr verfügbar ist, können andere verwendet werden.
    - Besserer Schutz gegen zerstörte/gefälschte Daten: gleichzeitiger Zugriff auf mehrere Replikate, das Ergebnis der Mehrheit wird verwendet
  - Steigerung der Leistungsfähigkeit des Zugriffs auf ein Datum
    - Bei großen Systemen: Verteilung der Replikate in verschiedene Netzregionen oder einfache Vervielfachung der Server an einem Ort

INSTITUT FÜR BETRIEBSSYSTEME UND RECHNERVERBUND
Prof. Dr.-Ing. L. Wolf | Prof. Dr. S. Fekete

Das große Problem

- Die verschiedenen Kopien müssen konsistent gehalten werden.
- Das ist insbesondere ein Problem
  - Wenn es viele Kopien gibt
  - Wenn die Kopien weit verstreut sind.
- Es gibt eine Reihe von Lösungen zur absoluten Konsistenzhaltung in nicht-verteilten Systemen, die jedoch die Leistung des Gesamtsystems negativ beeinflussen.
- Dilemma: wir wollen bessere Skalierbarkeit und damit bessere Leistung erreichen, aber die dazu notwendigen Mechanismen verschlechtern die Performance.
- Einzige Lösung: keine strikte Konsistenz

Anwendungsbeispiel: News

- Ansicht 1:

Bulletin board: os.interesting

Item	From	Subject
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
end		

- Ansicht 2:

Bulletin board: os.interesting

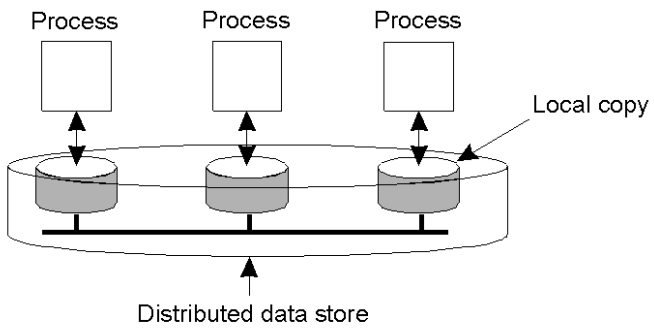
Item	From	Subject
20	G.Joseph	Microkernels
21	A.Hanlon	Mach
22	A.Sahiner	Re: RPC performance
23	M.Walker	Re: Mach
24	T.L'Heureux	RPC performance
25	A.Hanlon	Re: Microkernels
end		

- Probleme:
  - Nachrichten tauchen in unterschiedlicher Reihenfolge auf.
  - Sie kommen überhaupt nicht an.
- Für News ist das OK, aber andere Anwendungen?

System-Modell

- Daten im System = Sammlung von Objekten (Datei, Java-Objekt, etc.)
- Jedes logische Objekt wird durch eine Reihe physischer Objekte realisiert, die Replikate.
- Die Replikate müssen nicht zu jeder Zeit absolut identisch sein – sie können es tatsächlich auch nicht sein.
- Die Replikations-Intelligenz kann in den Objekten platziert sein oder außerhalb (in einer Middleware).
  - Vorteil der letzteren Methode: Anwendungsprogrammierer ist frei von Überlegungen zur Replikation

Modell eines verteilten Datenspeichers



Konsistenzmodelle

- Konsistenzmodell:
  - Im Prinzip ein Vertrag zwischen einem Datenspeicher und den darauf zugreifenden Prozessen
  - „Wenn sich die Prozesse an gewisse Regeln halten, arbeitet der Datenspeicher korrekt.“
  - Erwartung: der Prozess, der ein Read ausführt, erwartet als Ergebnis den Wert des letzten Write
  - Frage: was ist das letzte Write in Abwesenheit einer globalen Uhr?
  - Lösung: verschiedene Konsistenzmodelle (nicht nur strikt)
- Daten-zentrierte Konsistenzmodelle: Sicht des Datenspeichers
- Client-zentrierte Konsistenzmodelle: Sicht des Client, weniger starke Annahmen, insbesondere keine gleichzeitigen Updates

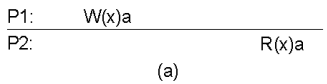
Notation

- Für die Diskussion der Konsistenzmodelle verwenden wir folgende Notation:

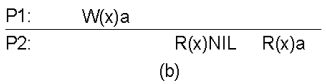
P_n: R(x)a	Prozess P_n liest die Variable x und erhält den Wert a
P_n: W(x)b	Prozess P_n überschreibt die Variable x mit dem Wert b
L_n: R(x)a	Von der lokalen Kopie L_n wird aus der Variable x der Wert a ausgelesen
L_n: W(x)b	Auf der lokalen Kopie L_n wird die Variable x mit dem Wert b überschrieben
L_n: WS(x)	Auf der lokalen Kopie L_n wird ein Satz von Schreiboperationen auf x ausgeführt

Strikte Konsistenz

- Daten-zentriert
- Konsequentestes Konsistenzmodell
- Modell: „Jedes Read liefert als Ergebnis den Wert der letzten Write-Operation.“
- Notwendig dazu: absolute globale Zeit
- Unmöglich in einem verteilten System, daher nicht implementierbar



(a)

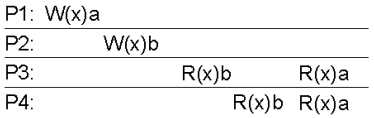


(b)

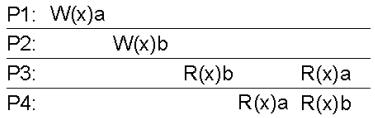
- (a) korrekt
- (b) inkorrekt

Sequentielle Konsistenz

- Etwas schwächeres Modell, aber implementierbar.
- Aussage: Wenn mehrere nebenläufige Prozesse auf Daten zugreifen, dann ist jede gültige Kombination von Read- und Write-Operationen akzeptabel, solange alle Prozesse dieselbe Folge sehen.
- Zeit spielt keine Rolle
- Beispiel: (a) ist korrekt, (b) nicht



(a)



(b)

6-13 Linearisierbarkeit

- Liegt in der Striktheit zwischen strikter und sequentieller Konsistenz
- Idee: verwende eine Menge synchronisierter Uhren, auf deren Basis Zeitstempel für die Operationen vergeben werden
- Verglichen mit sequentieller Konsistenz ist die Ordnung dann nicht beliebig, sondern auf der Basis dieser Zeitstempel
- Komplexe Implementierung, wird hauptsächlich eingesetzt zur formalen Verifikation nebenläufiger Algorithmen

6-14 Kausale Konsistenz

- Schwächeres Modell als die sequentielle Konsistenz
- Vergleichbar mit Lamports „happened-before“-Relation
- Regel: Write-Operationen, die potentiell in einem kausalen Verhältnis stehen, müssen bei allen Prozessen in derselben Reihenfolge gesehen werden. Für nicht in dieser Beziehung stehende Operationen ist die Reihenfolge gleichgültig.

P1:	W(x)a			W(x)c	
P2:		R(x)a	W(x)b		
P3:		R(x)a		R(x)c	R(x)b
P4:		R(x)a		R(x)b	R(x)c

Kausal konsistent, aber nicht sequentiell oder strikt

6-15 Vergleich der Modelle

Consistency	***Description***
Strict	Absolute time ordering of all shared accesses matters.
Linearizability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time
Causal	All processes see causally-related shared accesses in the same order.

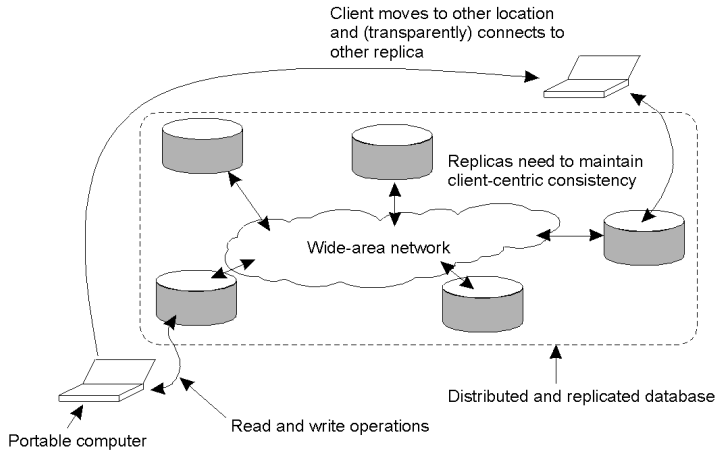
6-16 Eventual Consistency

- Idee: über lange Zeitspannen keine Updates, dann werden im Laufe der Zeit alle Replikate konsistent sein
- Beispiele für typische Anwendungen, bei denen ein solches Modell ausreicht
  - DNS
  - Web Caching
- Vorteil: meist sehr einfach zu implementieren, Write-Write-Konflikte treten meist nicht auf

Problem bei Eventual Consistency

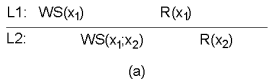
- Ein Problem tritt auf, wenn der Client die Replika wechselt, auf die er zugreift.
- Beispiel: mobiler Benutzer macht Updates, wechselt dann an anderen Platz. Updates sind eventuell noch nicht dort angekommen
  -> Benutzer stellt inkonsistentes Verhalten fest
- Lösung: client-zentrierte Modelle, bei denen für einen Client die Konsistenz garantiert wird, jedoch nicht für nebenläufigen Zugriff durch mehrere Clients

Illustration des Problems

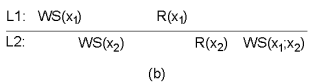


Monotonic Read Consistency

- Beispiel für ein client-zentriertes Konsistenzmodell
- Regel: Wenn ein Prozess den Wert einer Variable x liest, dann wird jede weitere Read-Operation denselben oder einen neueren Wert von x liefern.



(a)



(b)

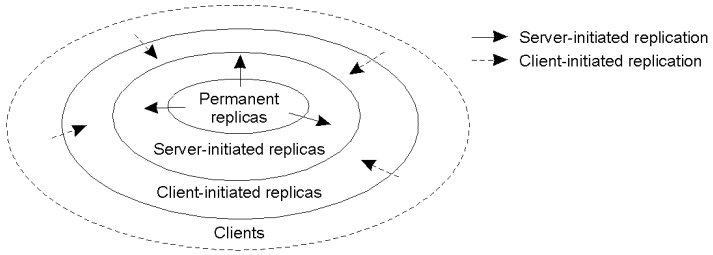
- (a) korrekt (b) nicht korrekt
- Beispiel: Zugriff auf Email-Box von versch. Orten

Verteilungsprotokolle

- Welche Möglichkeiten gibt es nun, Replikate zu verteilen und anschließend die Updates zu propagieren?
- Wir betrachten Verteilungsprotokolle und anschließend spezielle Konsistenzerhaltungsprotokolle.
- Beim Design solcher Protokolle müssen verschiedene Fragen beantwortet werden
  - Wo, wann und von wem werden die Replikate platziert?
  - Wie werden Updates propagiert?

Platzierung der Replikate

- Es können drei verschiedene Arten von Kopien unterschieden werden
  - Permanente Replikate
  - Server-initiierte Replikate
  - Client-initiierte Replikate



Permanente Replikate

- Grundlegende Menge von Replikaten, die meist beim Design eines Datenspeichers schon angelegt werden
- Beispiele:
  - replizierte Web-Site (Client merkt nichts von der Replikation),
  - Mirroring (Client sucht bewusst ein Replikat aus)
- Meist nur sehr wenige Replikate

Server-initiierte Replikate

- Kurzfristig initiiert bei hohem Bedarf, meist in der (Netz-)Gegend, in der der Bedarf auftritt
- Wichtige Grundlage für das Geschäftsmodell von *Web Hosting Services*
- Schwierige Entscheidung: wann und wo sollen die Replikate erzeugt werden?
- Existierende Algorithmen verwenden die Namen der gesuchten Dateien, die Anzahl und Herkunft der Requests zur Verteilung von Dateien (Web-Seiten)
- Dieser Ansatz kann permanente Replikas ersetzen, wenn garantiert ist, dass immer mindestens ein Server ein Datum vorrätig hält.

Client-initiierte Replikate

- Meist als (Client) Cache bezeichnet
- Management des Caches bleibt völlig dem Client überlassen, d.h., der Server kümmert sich nicht um Konsistenzerhaltung
- Einziger Zweck: Verbesserung der Datenzugriffszeiten
- Daten werden meist für begrenzte Zeit gespeichert (verhindert permanenten Zugriff auf alte Kopie)
- Der Cache wird meist auf der Client-Maschine platziert, oder zumindest in der Nähe von vielen Clients.

6-25 Propagierung von Updates

- Updates werden generell von einem Client auf einer Replika durchgeführt.
- Diese müssen dann an die anderen Replikas weiter gegeben werden.
- Verschiedene Design-Gesichtspunkte für die entsprechenden Protokolle
  - Was wird zu den anderen Replikaten propagiert?
  - Wird pull oder push eingesetzt?
  - Unicast oder Multicast?

6-26 Was wird propagiert?

- Spontan würde man sagen, dass derjenige Server, dessen Replikat geändert wurde, diesen neuen Wert an alle anderen schickt.
- Das muss aber nicht unbedingt so gemacht werden.
- Alternativen:
  - Sende nur eine Benachrichtigung, dass ein Update vorliegt (wird von Invalidation Protocols verwendet und benötigt sehr wenig Bandbreite)
  - Transferiere die das Update auslösende Operation zu den anderen Servern (benötigt ebenfalls minimale Bandbreite, aber auf den Servern wird mehr Rechenleistung erforderlich)

6-27 Pull oder Push?

- Push:
  - die Updates werden auf Initiative des Servers, bei dem das Update vorgenommen wurde, verteilt.
  - Die anderen Server schicken keine Requests nach Daten
  - Typisch, wenn ein hoher Grad an Konsistenz erforderlich ist
- Pull: umgekehrtes Vorgehen
  - Server fragen nach neuen Updates für Daten
  - Oft von Client Caches verwendet

6-28 Unicast oder Multicast

- Unicast: sende eine Nachricht mit demselben Inhalt an jeden Replika-Server
- Multicast: sende nur eine einzige Nachricht und überlasse dem Netz die Verteilung
- Meist wesentlich effizienter, insbesondere in LANs
- Multicast wird meist mit Push-Protokollen verbunden, die Server sind dann als Multicast-Gruppe organisiert
- Unicast passt besser zu Pull, wo immer nur ein Server nach einer neuen Version eines Datums fragt.

Protokolle zur Konsistenzerhaltung

- Wie lassen sich nun die verschiedenen Konsistenzmodelle implementieren?
- Dazu benötigt man Protokolle, mit deren Hilfe sich die verschiedenen Replika-Server abstimmen.
- Man unterscheidet zwei grundlegende Ansätze für diese Protokolle:
  - Primary-based Protocols (Write-Operationen gehen immer an dieselbe Kopie)
  - Replicated-Write Protocols (Write-Operationen gehen an beliebige Kopien)

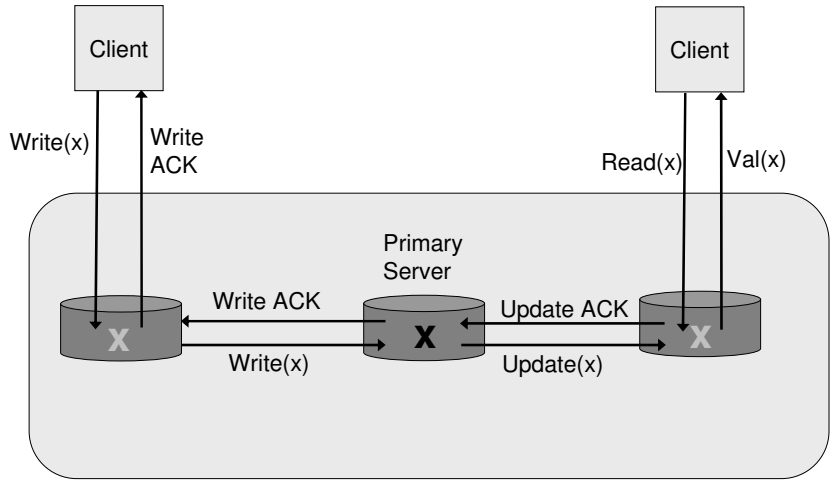
Primary-Based Protocols

- Wenn alle Write-Operationen immer nur an eine Kopie gehen, kann man noch einmal unterscheiden,
  - Ob diese Kopie immer am selben entfernten Platz bleibt
  - Ob die Primärkopie zu dem schreibenden Client verlagert wird.
- Dementsprechend werden unterschiedliche Algorithmen und Protokolle verwendet.

Remote-Write-Protokolle

- alle Updates auf einem einzigen entfernten Server
- Lese-Operationen auf lokalen Kopien
- Nach Update Aktualisierung der Kopien, ACK zurück an Primary, der dann den Client informiert → damit bleiben alle Kopien konsistent
- Problem: Performance, deshalb wird auch non-blocking Update eingesetzt (aber hier wieder Problem mit Fehlertoleranz)
- Beste Umsetzung für sequentielle Konsistenz

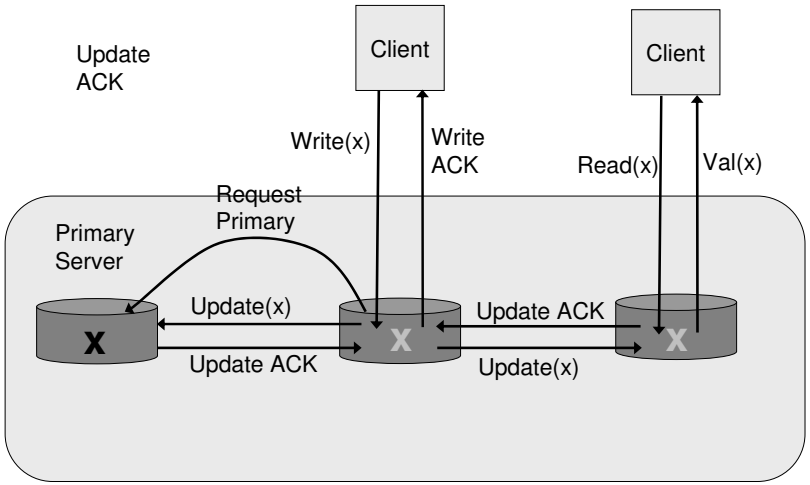
Ablauf von Remote Write



Local-Write-Protokolle

- Jeder Prozess, der ein Update ausführen will, lokalisiert die Primary Copy und bewegt diese dann an seinen eigenen Platz.
- Gutes Modell auch für mobile Benutzer:
  - hole primary copy
  - breche Verbindung ab
  - Arbeite
  - baue später Verbindung wieder auf
  - keine Updates durch andere Prozesse!

Ablauf von Local Write



Replicated-Write Protocols

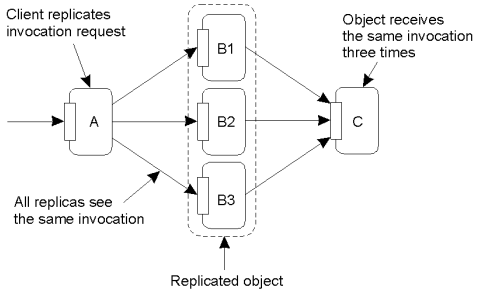
- Bei dieser Art von Protokollen können Write-Operationen auf beliebigen Replikaten ausgeführt werden.
- Es muss dann entschieden werden, welches der richtige Wert eines Datums ist.
- Zwei Ansätze:
  - Active Replication: eine Operation wird an alle Replikas weiter gegeben
  - Quorum-based: es wird abgestimmt, die Mehrheit gewinnt

Aktive Replikation

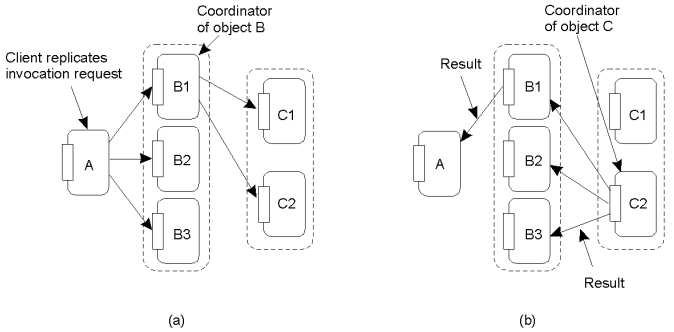
- Jede Replika besitzt einen Prozess, der die Updates durchführt.
- Updates werden meist als Operation propagiert
- Wichtigstes Problem: alle Updates müssen auf allen Replikas in derselben Reihenfolge ausgeführt werden,
- D.h., es wird Multicast mit totaler Ordnung benötigt (s. Kapitel über Zeit.), implementiert mittels Lamport-Uhren
- Skaliert aber nicht gut
- Alternative: zentraler Prozess, der die Sequentialisierung übernimmt
- Kombination aus beiden Ansätzen hat sich als brauchbar erwiesen

Replizierte Objektaufrufe

- Was passiert, wenn ein repliziertes Objekt ein anderes Objekt aufruft?
- Jede Replika ruft das Objekt auf!
- Lösung: verwende eine Middleware, die sich der Replikation bewusst ist.
- Löst auch das Problem der Verarbeitung der Antworten



Koordination der replizierten Objekte



a) Weiterleitung eines Aufrufs von einem replizierten Objekts an ein anderes
b) Rückgabe der Antwort

Quorum-Based Protocols

- Idee: Clients müssen zur Ausführung einer Read- oder Write-Operation die Erlaubnis mehrerer Server einholen
- Jedes Objekt besitzt eine Versionsnummer.
- Wenn der Client ein Read oder Write durchführen will, muss er die Erlaubnis von N/2+1 aller N Server erhalten.
- Ist das der Fall, kann kein anderer Client eine entsprechende Operation ausführen, da er auf keinen Fall mehr als die Hälfte der Server „hinter sich“ hat.

Zusammenfassung

- Replikation ist ein mächtiges Instrument, um Verfügbarkeit und Performance in einem VS zu steigern.
- Großes Problem: Konsistenz
- Für unterschiedliche Anwendungsanforderungen wurden unterschiedliche Lösungen erarbeitet.
- Konsistenzmodelle müssen implementiert werden!
- Benötigt werden
  - Verteilungsprotokolle
  - Konsistenzprotokolle
- Wir haben eine Vielzahl von Protokollen kennen gelernt, die für jeweils unterschiedliche Modelle geeignet sind.

Diskussion