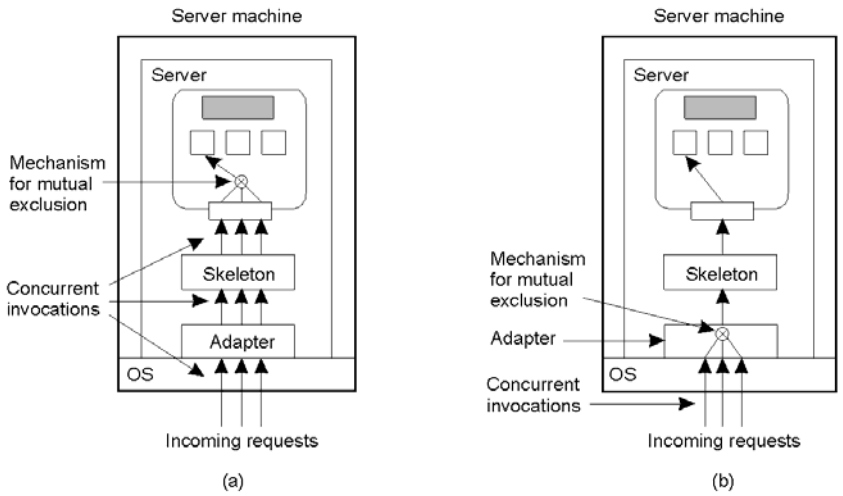
Consistența şi replicarea datelor în Internet

Bazat pe “Distributed Systems” de AS Tanenbaum

Motive pentru replicare

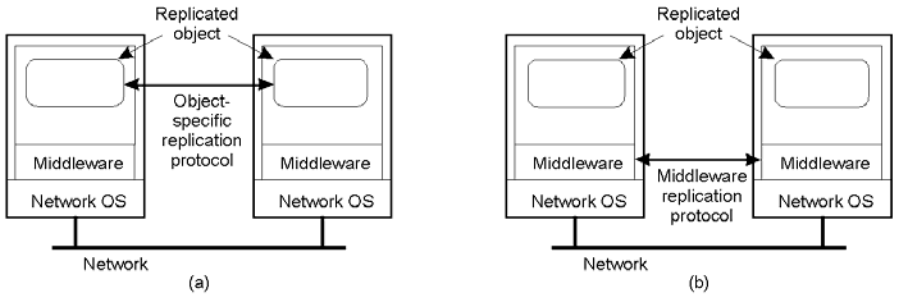
- Siguranta
  - Sistemul continua sa lucreze dupa caderea unei replici
  - Protectie mai buna impotriva datelor corupte
- Performanta
  - Scalare in numar resurse
  - Scalare in arie geografica
- Replicare => Probleme de consistenta

Controlul accesului concurent la un obiect



a) Manipularea accesului concurent de catre obiect
b) Folosirea unui adaptor pentru manevrarea accesului concurent

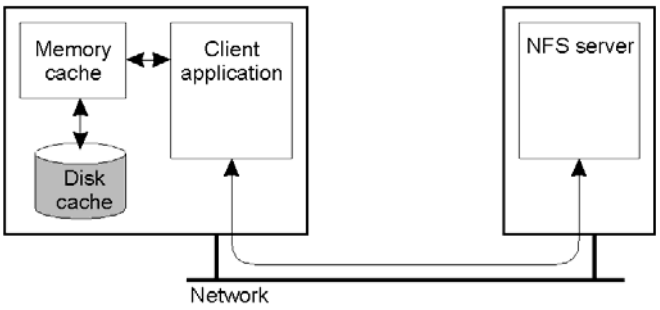
Invocari concurente ale obiectelor replicate



Problema: aceleasi modificari de stare sa apara la obiectele replicate
Solutia: invocarile sa se faca in aceeasi ordine la toate replicile
Doua **implementari** posibile:

- Obiectele sunt "constiente" de replicare (Globe)
- Sistemul distribuit (middleware) face gestiunea replicilor (Piranha)
  - de ex. ordonarea totala a invocarilor la toate replicile

Replicare in sistemele de fisiere



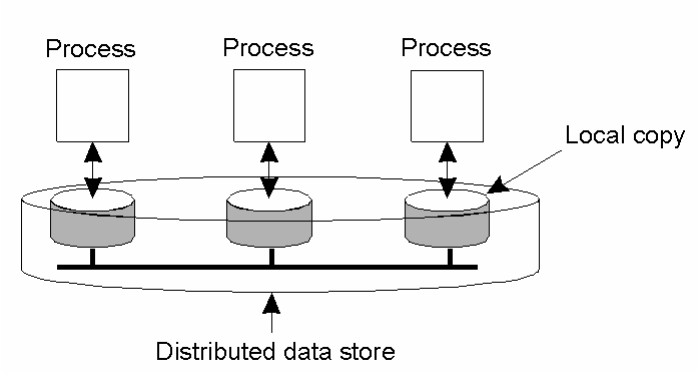
Problema
- eficientizarea accesului partajat la fisiere

Solutii
- replicare la client - caching pentru date, atribute, handlere, directoare
- replicarea fisierelor la server sau in sisteme p2p

Modele de consistență

- Consistenta stricta - greu de pastrat
- Solutie = constrangeri mai slabe de consistenta
- Depind de
  - tiparele de acces si actualizare
  - aplicatii
- Modele de consistenta
  - centrate pe date
  - centrate pe client

Modele de consistenta centrate pe date



Organizare generala depozit de date distribuit si replicat.
Operatii: **read** (copie locala), **write** (propagata la celelalte copii)

Model de consistenta = contract intre proces si data store

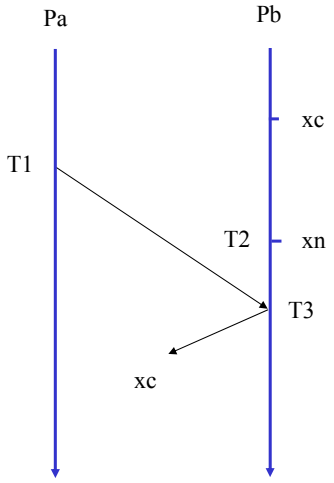
- **read ar trebui sa intoarca rezultatul ultimei operatii**

Consistența Strictă

Orice citire a unei date x intoarce valoarea rezultata din cea mai recenta scriere a lui x.

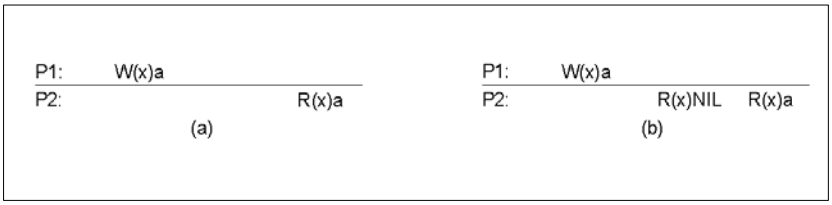
Ex:

- Doua masini A si B
- Doua procese: Pa pe A si Pb pe B
- **x** memorat pe B (doar); valoarea curenta este **xc**
- La T1, Pa citeste **x** (trimite un mesaj lui B sa citeasca x)
- La T2>T1, Pb scrie **x**; noua valoare este **xn**
- La T3>T2, mesajul de citire de la Pa soseste la B
- Consistenta stricta => B ar trebui sa intoarca **xc**



Consistența Strictă (2)

- **Se pastreaza ordinea absoluta globala in timp**
- **Toate scrierile sunt instantaneu vizibile tuturor proceselor**



Comportarea a doua procese operand pe acelasi item.
a) Memorie strict consistenta.
b) Memorie care nu este strict consistenta.

Greu de implementat

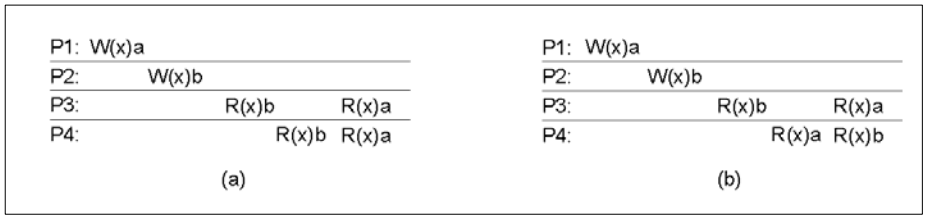
Programele concurente nu sunt bazate pe timpul global sau pe viteza proceselor (ex. Producer / Consumer)

Sunt necesare modele mai slabe

Consistenta Secvențială (1)

Consistenta secventiala: **Rezultatul oricarei executii este acelasi cu cel obtinut daca operatiile (read, write) tuturor proceselor asupra depozitului de date sunt executate intr-o secventa oarecare si operatiile fiecarui proces individual apar in aceasta secventa in ordinea specificata de programul sau.**

- **Orice intretesere valida a operatiilor read si write este acceptabila**
- **Toate procesele vad aceeasi intretesere de operatii**
- **Operatiile nu au amprente de timp**



Un depozit de date **(a)** secvential consistent si **(b)** care nu este secvential consistent.

Consistența Secvențială (2)

x, y, si z sunt initializte la 0

Proces P1	Proces P2	Proces P3
x = 1; print (y, z);	y = 1; print (x, z);	z = 1; print (x, y);

Trei procese concurente.

assignment = write
print = doua read simultane

- Sunt posibile 90 ordonari valide diferite ale instructiunilor
  90 = 720 (=6!) / 8
- mai putin de 64 tipare de signatura
  - signatura = concatenarea iesirilor lui P1, P2 si P3 in aceasta ordine
- procesele trebuie sa accepte toate rezultatele valide

Consistența Secvențială (3)

x = 1;
print (y, z);
y = 1;
print (x, z);
z = 1;
print (x, y);
 	
Prints: 001011
 	
Signatura:
001011
(a)
	x = 1;
	y = 1;
	print (x,z);
	print(y, z);
	z = 1;
	print (x, y);
	
	Prints: 101011
	
	Signatura:
	101011
	(b)
		y = 1;
		z = 1;
		print (x, y);
		print (x, z);
		x = 1;
		print (y, z);
		
		Prints: 010111
		
		Signatura:
		110101
		(c)
			y = 1;
			x = 1;
			z = 1;
			print (x, z);
			print (y, z);
			print (x, y);
			
			Prints: 111111
			
			Signatura:
			111111
			(d)

Patru secvente de executie valide pentru procesele anterioare.
Axa verticala este timpul.
Nu toate semnaturile sunt permise. Ex. 000000 si 001001

Consistența Cauzală (1)

Operatii:

- Legate Cauzal
  - ex:
    - procesul p executa (1) write x;
    - ulterior, procesul q executa (2) read x; (3) write y
    - (1) si (3) sunt legate cauzal
- Concurente

Conditie necesara:

Operatiile "write" care sunt potential legate cauzal trebuie sa fie vazute de toate procesele in aceeasi ordine. Scrierile concurente pot fi vazute in diferite ordini pe masini diferite.

Consistența Cauzală (2)

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

$W_1(x)a$ si $W_2(x)b$ sunt dependente cauzal;

$W_2(x)b$ si $W_1(x)c$ sunt concurente.

Aceasta secventa este permisa cu o memorie cauzal-consistenta, dar nu cu una secvential sau strict consistenta.

Consistenta Cauzala (3)

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

(a)

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

(b)

a) Violare a consistentei cauzale. (Cele doua scrieri sunt legate cauzal.)

b) O secventa corecta de evenimente intr-o memorie cauzal-consistenta. (Cele doua scrieri nu mai sunt legate cauzal.)

Consistenta FIFO (1)

Conditie Necesara:

Scrierile facute de un singur proces sunt vazute de toate celelalte procese in ordinea in care au fost executate, dar scrierile din procese diferite pot fi vazute in ordini diferite de procese diferite.

Usor de implementat

- etichetand fiecare operatie write cu perechea (proces, numar secventa)

Consistenta FIFO (2)

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

O secventa valida de evenimente pentru consistenta FIFO

Consistenta FIFO (3)

Process P1	**Process P2**
x = 1;	y = 1;
if (y == 0) kill (P2);	if (x == 0) kill (P1);

Doua procese concurente.

- Consistenta FIFO: ambele procese pot fi omorate
  - scrierile sunt vazute in ordini diferite
- Consistenta Secventiala: sase posibile intreteseri, nici una cu ambele procese omorate

Gruparea operatiilor

Datele partajate pot fi asociate cu o variabila de sincronizare.

Un proces foloseste acquire si release pe variabilele de sincronizare

- acquire – cand intra in sectiunea critica - datele protejate de variabila de sincronizare sunt facute consistente
- release - cand iese din sectiunea critica

Model specific de folosire variabile de sincronizare

- Fiecare variabila de sincronizare are un proprietar curent
- Proprietarul poate modifica datele protejate de variabila de sincronizare, in mai multe sectiuni critice (ramane proprietar)
- Un proces care vrea sa acapareze (acquire) trebuie sa faca o cerere proprietarului, devenind noul proprietar
- Mod ne-exclusiv – Mai multe procese detin variabila doar pentru citirea datelor protejate

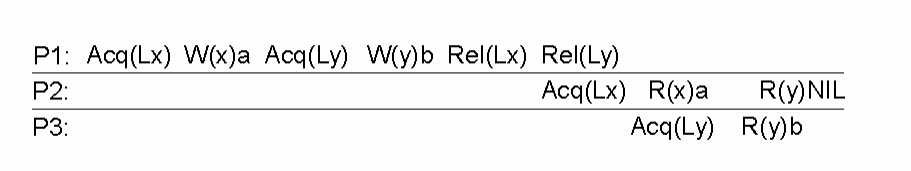
Criterii de consistenta

Un proces nu poate termina un acquire pe o variabila de sincronizare pana cand nu s-au actualizat toate datele partajate protejate de acea variabila.

Pentru ca un proces sa modifice date partajate, el trebuie sa capate accesul exclusiv la variabila de sincronizare care protejeaza acele date (nici un alt proces sa nu detina variabila de sincronizare, nici macar in mod ne-exclusiv)

Dupa un acces exclusiv la o variabila de sincronizare, orice acces ne-exclusiv ulterior al unui alt proces la acea variabila de sincronizare este admis doar dupa ce procesul a preluat de la proprietarul variabilei de sincronizare cele mai recente copii ale variabilelor partajate protejate.

Consistenta la Intrare



O secventa valida pentru consistenta la intrare.

Fiecare variabila partajata **x** este asociata cu o variabila de sincronizare (lock) **Lx**.

P2 citeste corect pe **x** (face acquire) dar nu si pe **y**

Modele de Consistenta Centrata pe Client

Este un caz special: lipsesc actualizarile simultane

Exemple:

- Sisteme de baze de date
- DNS
- WWW

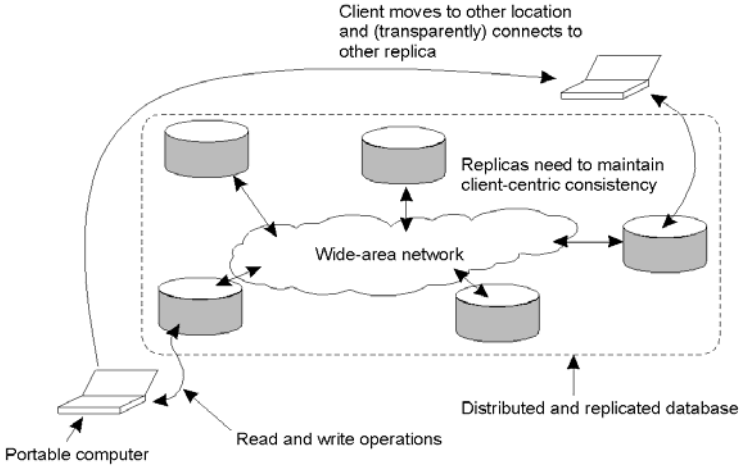
Consistenta eventuala:

- in absenta indelungata a unor actualizari, toate replicile vor deveni treptat consistente

Consistenta "Client-centric"

- un utilizator poate opera pe replici diferite
- da garantii unui singur client privind consistenta acceselor la replici diferite ale acelui client

Consistenta eventuala vs. Client-centric



Un utilizator mobil accesand diferite replici ale unei baze de date distribuite

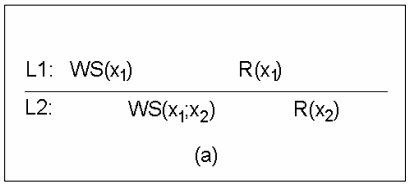
- **valorile citite si modificarile facute pe o replica sa fie regasite la cealalata replica**

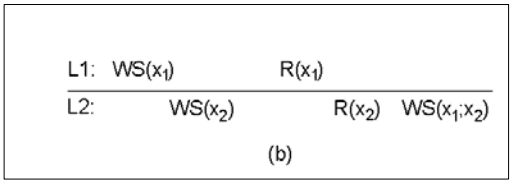
Citiri monotone (Monotonic Reads)

Daca un proces citeste valoarea unei date x, orice citire succesiva a lui x de catre acel proces va returna acea valoare sau o valoare mai noua.

Exemplu: baza de date distribuita, pentru e-mail.

$WS(x_1)$ este seria de operatii de scriere pe x executate la locatia 1 de la initializare; $WS(x_1;x_2)$ => $WS(x_1)$ este parte a lui **$WS(x_2)$**





Operatiile *read* excutate de un singur proces *P* pe doua copii locale diferite ale aceleiasi baze de date.

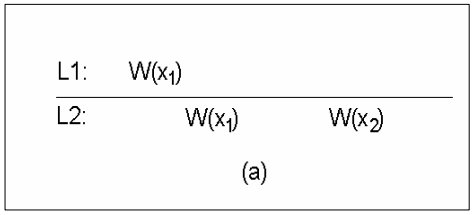
a) Memorie **consistenta** "monotonic-reads"
b) Memorie **ne-consistenta** "monotonic-reads"

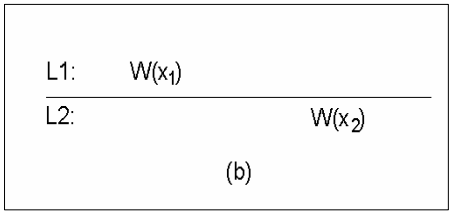
Scrieri monotone (Monotonic Writes)

O operatie *write x* a unui proces este terminata inaintea oricarei operatii *write x* ulterioare a aceluiasi proces.

Exemplu: actualizarea unei biblioteci software.

Obs. Seamana cu consistenta FIFO





Operatiile de scriere executate de un proces *P* pe doua copii locale diferite ale aceluisi depozit de date

a) Un depozit consistent "monotonic-writes".

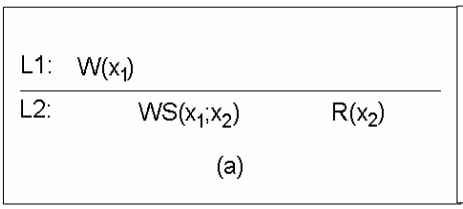
b) Un depozit ne-consistent "monotonic-writes".

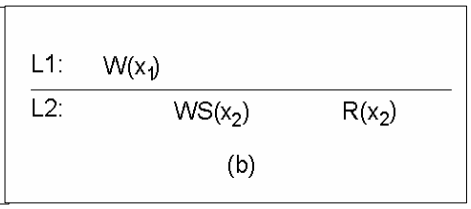
Citirea scrierilor proprii (Read Your Writes)

Efectul unei operatii *write x* a unui proces P va fi totdeauna vazut de operatiile *read x* executate ulterior de acelasi proces P.

Contra-exemplu: actualizarea paginilor Web si observarea efectelor.

Similar: actualizarea password.





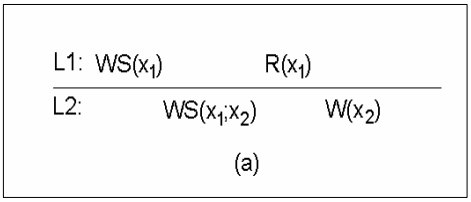
a) Un depozit care **ofera** consistenta "read-your-writes".

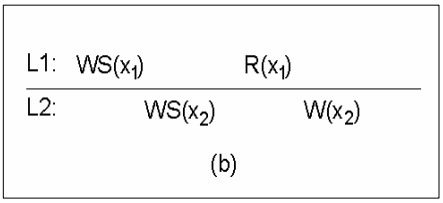
b) Un depozit care **nu ofera**.

Scrierile urmeaza citirile (Writes Follow Reads)

O operatie *write x* executata de un proces P dupa o operatie prealabila *read x* a aceluiasi proces se executa garantat pe aceeasi valoare a lui *x* sau pe una mai recenta decat cea citita.

Ex.: un utilizator citeste un articol A la care raspunde cu B; B va fi scris in orice copie a newsgroup-ului doar dupa ca A a fost scris de asemenea.





a) Un depozit care **ofera** consistenta "writes-follow-reads".
b) Un depozit care **nu ofera**.

Implementarea

Fiecarei operatii write i se asociaza un identificator global unic, de catre serverul care accepta operatia pentru prima data

Pentru fiecare client se pastreaza doua seturi de identificatori de write:

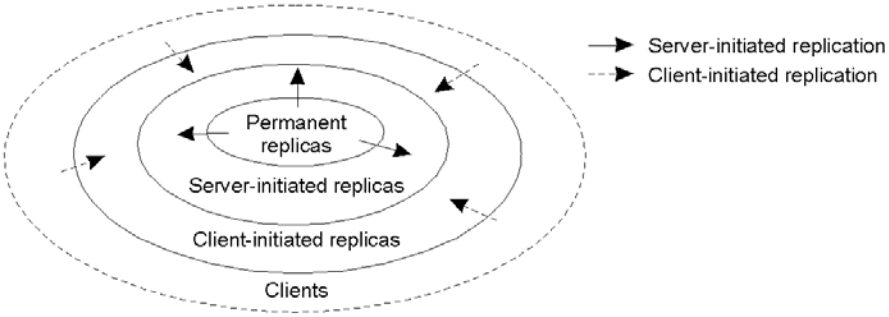
- **Read – identificatori ai operatiilor write relevante pentru operatiile read executate de client**
- **Write – identificatori ai operatiilor write executate de client**

Monotonic read – pentru fiecare read:

- **Serverul primeste setul read al clientului**
- **Verifica daca toate operatiile write au fost facute local**
- **Nu -> contacteaza celelalte servere si face actualizarea**
- **Serverul face citirea**
- **Serverul actualizeaza setul read cu operatiile write realizate si care sunt relevante pentru client**

Actiuni similare sunt executate pentru celelalte modele

Protocoale de distributie - Plasarea Replicilor



Organizarea logica a diferitelor categorii de copii ale depozitelor de date in trei inele concentrice.

Plasarea Replicilor (2)

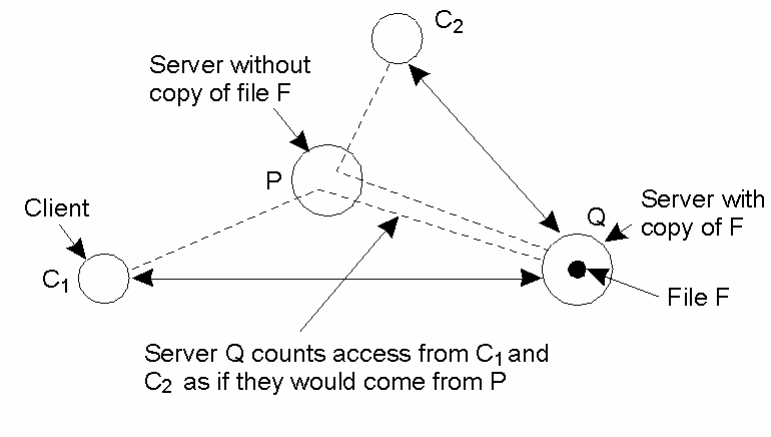
Replici permanente

1. servere "replica" situate in acelasi sistem (cluster)
2. Mirroring – replici distribuite geografic

Replici initiate de server

- Create dinamic de servere pentru a imbunatati performanta
  - replici apropiate de clienti
  - replici pentru reducerea incarcarii unui server
- Cand si unde? Algoritm (Rabinovich):
  - Fiecare server tine evidenta numarului de accese per fisier si a originii cererilor (vezi slide urmator)
  - Prag de stergere si prag de replicare
  - Intre => posibila migrare

Numararea cererilor de acces de la diversi clienti.



Plasarea Replicilor (3)

Replici initiate de clienti (caches)

- Gestionate de clienti folosind informatii de la servere
- Utilizate doar pentru a imbunatati timpul de acces la date
- Date pastrate pentru un timp limitat
- Cache-uri partajate de clienti

Propagarea Actualizarilor

Stare versus operatii

- Propagarea notificarilor
  - Utilizate de protocoalele de invalidare
  - Utilizeaza largimi de banda reduse
- Propagarea schimbarilor
  - Utila cand rata read / write este ridicata
  - Log-urile modificarilor pot fi propagate
- Propagare operatii de actualizare
  - Replicare activa
  - Cost communicare redus

Protocoale Pull versus Push

Push:

replici permanente si "server initiated" care necesita un grad inalt de consistenta

Pull:

folosite pentru cache-uri client; utile cand rata read / write este redusa

Element	Push-based	Pull-based
Stare server	Lista replicilor si a cache-urilor client	Nimic
Message transmise	update sau invalidare plus fetch update ulterior	poll si update
Timp raspuns la client	Imediat (sau timp de fetch update)	Timp fetch update

Comparatie intre protocoale push-based si pull-based in cazul sistemelor cu mai multi clienti si un singur server.

Protocoale de actualizare hibride

Contract – o promisiune ca serverul va transmite actualizari la client pentru un timp specificat.

Timpul poate fi adaptat dinamic in functie de criteriile de contract (lease)

- bazat pe vechime
  - contracte de durata pentru elemente cu sansa de a ramane nemodificate
- bazat pe frecventa de innoire
  - contracte de durata pentru clienti ale caror cache-uri trebuie innoite frecvent
- overhead-ul datorat starii serverului
  - Serverul scade timpul de expirare al noilor contracte daca este supraincarcat (scade numarul de clienti)

Protocoale epidemice

Rol: propagarea actualizarilor la replici cu un numar minim de mesaje

Varianta "anti-entropie"

Categorii de servere

- Infectios – detine un update pe care vrea sa-l propage
- Susceptibil – inca ne-actualizat
- Eliminat – detine un update dar nu vrea sa-l propage

Abordari

- Push based – mai bun pentru putini infectiosi
- Pull based – mai bun pentru multi infectiosi

Varianta Speciala: raspandirea zvonurilor (gossiping)

- Interesul serverului de a propaga actualizarile scade pe masura ce intalneste servere actualizate deja
- Nu garanteaza actualizarea tuturor replicilor

Certificate de deces - probleme

Propagarea stergerii unui element de date este dificila

Solutie: certificate de deces

Cand se sterge certificat de deces?

- Executa procedura pt. a detecta daca stergerea este cunoscuta peste tot

  Apoi colecteaza certificatele de deces (similar garbage collection)
- Asociaza un timp de viata maxim cu fiecare certificat
  - Exista riscul de a nu atinge toate serverele
  - Solutie: un numar mic de servere pastreaza nelimitat certificate "dormante"

Protocoale de consistenta

Un protocol de consistenta descrie o implementare a unui model de consistenta

Protocoale bazate pe o copie primara

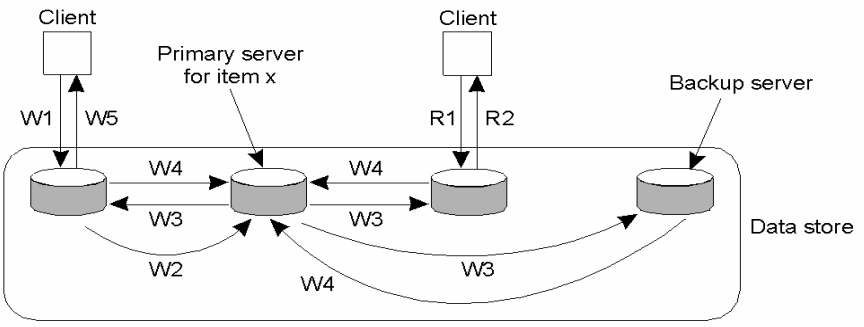
- scriere la distanta (Remote-write)
- scriere locala (Local-write)

Protocoale cu scriere replicata

- replicare activa
- bazate pe cvorum

Protocoale de coerenta a cache-urilor

Protocoale Remote-Write



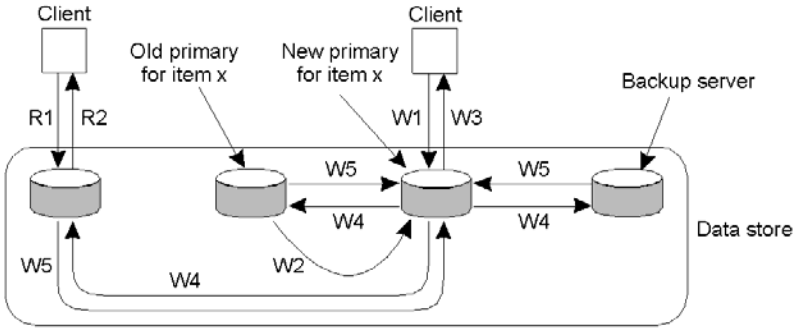
W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

Principiul protocolului primary-backup.

Implementeaza consistenta secventiala (copia primara poate ordona toate operatiile de scriere primite)

Protocoale Local-Write

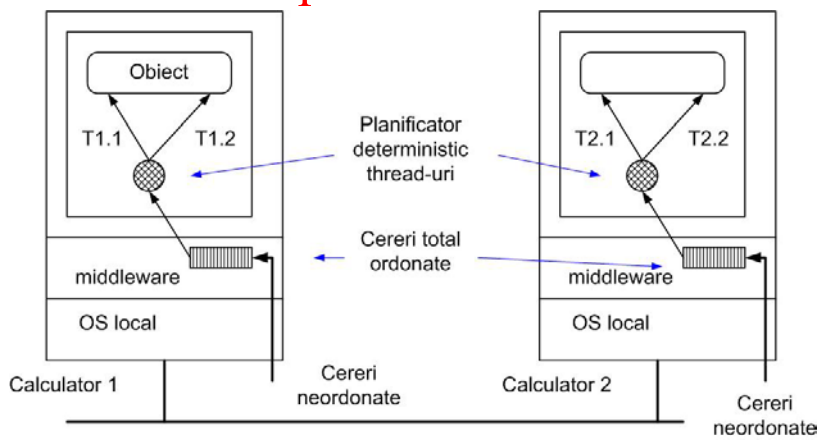


W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

Protocol "primary-backup" in care copia primara migreaza la procesul care vrea sa faca actualizarea

Replicare activa



Doua probleme:

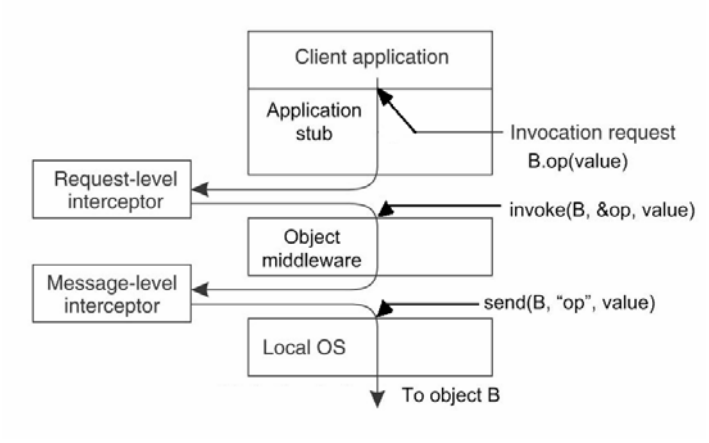
- Prevenirea executiei concurente a invocarilor aceluiasi obiect (lock-uri locale obiectului)
- Operatiile trebuie executate in aceeasi ordine in toate replicile
  - Solutia 1:
    - folosirea unei scheme primary-based la nivel aplicatie
      - » serializeaza cererile
      - » efort al dezvoltatorului de aplicatii

- Solutia 2: implementare in middelware
  - multicast total ordonat pentru invocari
  - trebuie asigurat si ca threadurile trateaza cererile in ordinea corecta

– Problema

- functionare servere multithread
  - preiau o cerere
  - o paseaza unui thread disponibil
  - trec la urmatoarea cerere
  - planificatorul de threaduri aloca procesorul threadurilor executabile –ordinea nu este aceeasi in toate replicile
- solutie: planificator determinist
  - are la baza o varianta de replica primara
    - » copia primara determina ordinea threadurilor
    - » o comunica celorlalte replici
  - frecvente comunicari intre replici - ineficient

Separarea replicare de functionalitate obiecte

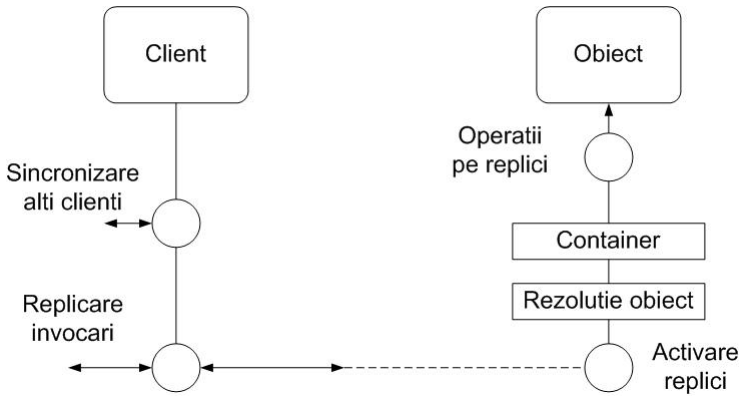


Bazata pe mecanismul interceptorilor

- constructie software care intrerupe fluxul de control normal si permite executia altor functii
- ex.: inserare interceptori in invocarea unor metode la distanta

Java IDL and Java RMI-IIOP Technologies: Using Portable Interceptors (PI)
http://java.sun.com/j2se/1.4.2/docs/guide/idl/PI.html

Un cadru complet de replicare



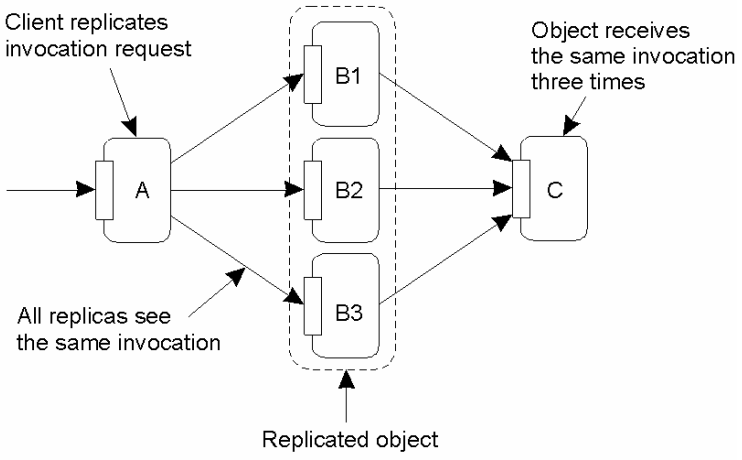
Interceptori client

- Sincronizare cu alti clienti, cand clientul este replicat
- Replicare invocari, cand invocarea trebuie transmisa mai multor replici

Interceptor server

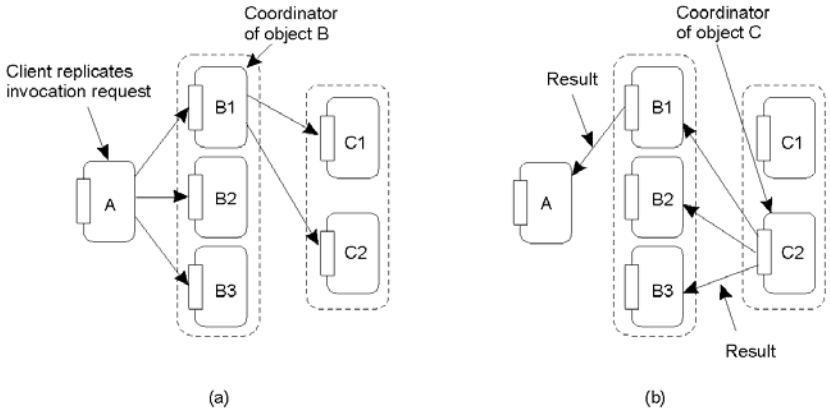
- Activare replici, daca invocarea se adreseaza mai multor replici
- Multiplicarea operatiilor daca trebuie aplicate mai multor replici

Invocari Replicate



Problema: la C ajung invocari replicate; doar una este necesara

Solutie middleware: Communicatii constiente de replicare

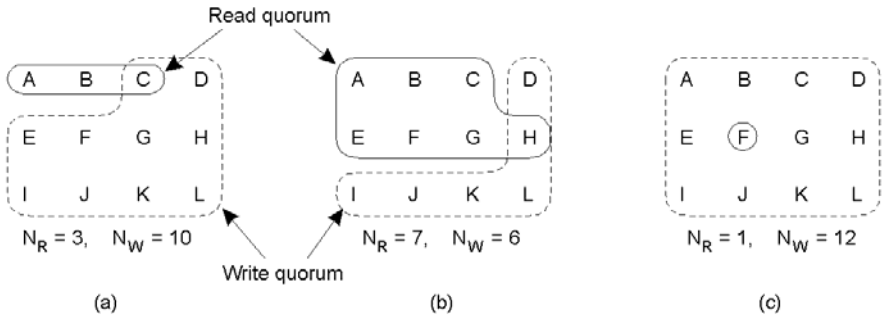


Solutie: bazata pe transmiterea multicast a invocarilor/raspunsurilor

- replicile asigneaza acelasi identificator invocarilor
- doar coordonatorul transmite invocarea
- toate replicile primesc copii ale aceluiasi raspuns

Alternativa: identificarea si eliminarea invocarilor duplicate la un obiect

Protocoale bazate pe cvorum



Trei exemple pentru algoritmul de votare:

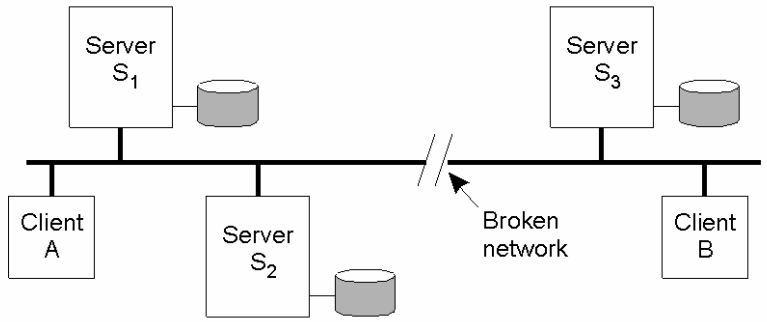
a) Alegere corecta a seturilor de citire si scriere

b) Alegere ce poate conduce la conflicte write-write

- ex. un proces alege setul de scriere {A,B,C,E,F,G} iar altul {D,H,I,J,K,L}

c) Alegere corecta, cunoscuta ca ROWA (read one, write all)

Aplicatie: replicarea serverelor in CODA



Unitatea de replicare = volum

VSG (Volume Storage Group) = grupul de servere care au copia volumului

AVSG (Accessible Volume Storage Group) = serverele la care un client are acces; Ex.

- S1, S2 pentru clientul A
- S3 pentru clientul B

CODA foloseste ROWA pentru consistenta

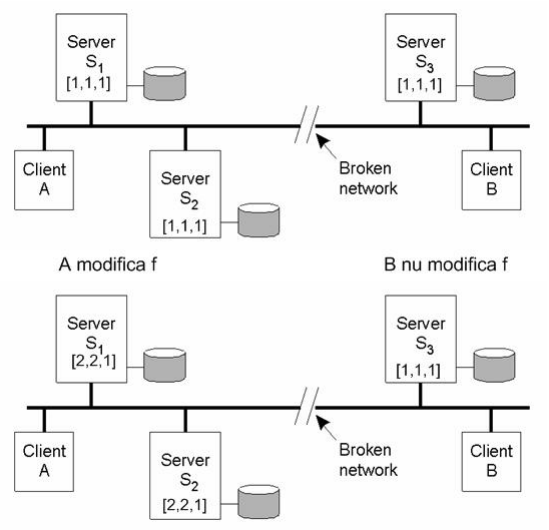
- deschide un fisier pe un membru AVSG
- la inchidere scrie toate replicile din AVSG

Varianta optimista la partitionare retea

- fiecare client lucreaza pe AVSG-ul sau

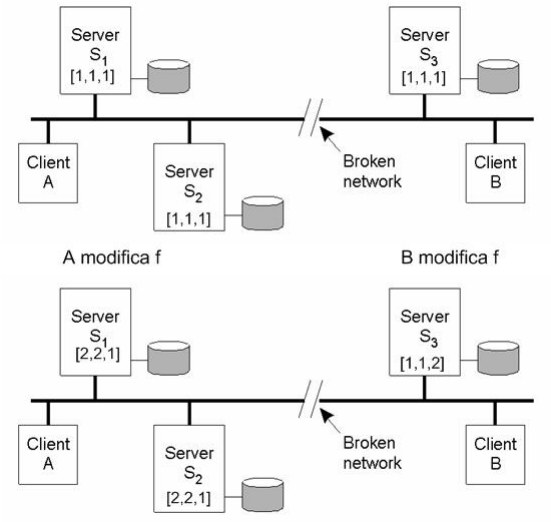
Detectie inconsistente

- CODA pastreaza pentru fiecare fisier **f** un vector de versiune (Coda Version Vector) **$CVV_k(f)$**
- $CVV_k(f)[i]$ este numarul versiunii curente a lui f pe serverul Si, pastrat de k
- la rezolvarea defectului de partitionare a retelei, se compara CVV-urile si se detecteaza conflictele



Fara conflict

Modificarile sunt acceptate de S3



Conflict

Rezolvare

- solutie dependenta de aplicatie, posibil automata
- solutie manuala

Protocoale de coerenta cache-urilor

Caz special de replicare controlata de client

Cand sunt detectate inconsistentele? (coherence detection strategy)

- Static, in compilatoare
  - determina ce date pot conduce la inconsistente
  - insereaza cod pentru ocolirea inconsistentei
- Dinamic, la executie – trei politici
  - La accesarea unei date din cache se face mai intai verificarea
  - Lasa tranzactia sa se faca in paralel cu verificarea coerentei
  - Verifica coerenta datelor cand tranzactia se comite

Cum sunt cache-urile pastrate consistente? (coherence enforcement strategy)

- Nu permite cache-ul datelor partajate
- Lasa serverul sa transmita invalidarea sau propagarea actualizarii

Ce se intampla cand un proces modifica datele din cache?

- Serverul modifica si propaga actualizarile la cache-uri
- Clientul capata drept de acces exclusiv, modifica si paseaza actualizarile la servere (write-through caches)