Università degli Studi di Roma “Tor Vergata”
Facoltà di Ingegneria

Consistenza e replicazione

Corso di Sistemi Distribuiti

Valeria Cardellini
Anno accademico 2009/10

Replicazione: motivazioni e problemi

- Perché replicare i dati?
  - Per aumentare l’affidabilità del sistema
    - Rendendo il sistema tollerante ai guasti
  - Per migliorare le prestazioni del sistema, aumentandone la scalabilità
    - Scalabilità rispetto alla dimensione
    - Scalabilità geografica
- Cosa comporta la replicazione?
  - La presenza di molteplici copie di un dato può causare problemi di consistenza
    - Una replica modificata diventa diversa dalle altre: la modifica deve essere eseguita su tutte le altre repliche

Come e quando eseguire la modifica delle repliche?

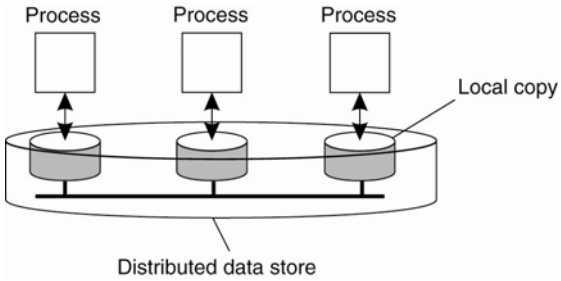
Come evitare un degrado pesante delle prestazioni dovuto alla consistenza, soprattutto nei SD a larga scala?

Consistenza

- Per mantenere le repliche consistenti, occorre garantire in generale che tutte le operazioni conflittuali siano eseguite nello stesso ordine su tutte le repliche
- **Operazioni conflittuali**: nell'ambito delle transazioni
  - *Conflitto read-write*: un'operazione di lettura ed un'operazione di scrittura concorrenti sullo stesso dato
  - *Conflitto scrittura-scrittura*: due operazioni di scrittura concorrenti sullo stesso dato
- Garantire un ordinamento globale delle operazioni può essere costoso e compromettere la scalabilità del sistema
- *Soluzione*: allentare i vincoli di consistenza per evitare una sincronizzazione globale ed ottenere un sistema efficiente
  - *Attenzione*: non esiste un'unica soluzione generale, bensì diverse soluzioni adatte ad applicazioni aventi diversi requisiti di consistenza

Modelli di consistenza

- Un archivio di dati distribuito è un insieme di spazi di memorizzazione, fisicamente distribuiti e replicati su molteplici processi
  - Ad es. una base di dati distribuita o un file system distribuito



- **Modello di consistenza**
  - Un contratto tra un archivio di dati (distribuito) che stabilisce che, se i processi rispettano un certo insieme di regole, l'archivio assicura un funzionamento corretto

Modelli di consistenza (2)

- Processi concorrenti possono aggiornare simultaneamente un archivio di dati
- Modelli di consistenza **data-centrici**
  - Obiettivo: fornire una vista di un archivio di dati consistente a livello di sistema
- Modelli di consistenza **client-centrici**
  - Obiettivo: fornire una vista di un archivio di dati consistente a livello di un singolo client
  - Prevedono l'uso di varie tecniche per garantire una politica di gestione della consistenza più veloce, ma meno accurata

Modelli di consistenza data-centrici

- Consistenza stretta
  - L'aggiornamento viene eseguito su tutte le repliche come singola operazione atomica: è come se ci fosse una copia unica
  - Ordinamento temporale assoluto di tutti gli accessi all'archivio di dati
  - In mancanza di un orologio globale, è difficile stabilire esattamente quale sia l'ultima operazione di scrittura: come indebolire il vincolo della consistenza stretta?
- Modelli di consistenza basati sull'ordinamento consistente delle operazioni sui dati condivisi e replicati
  - Consistenza sequenziale
  - Consistenza causale
- Consistenza continua

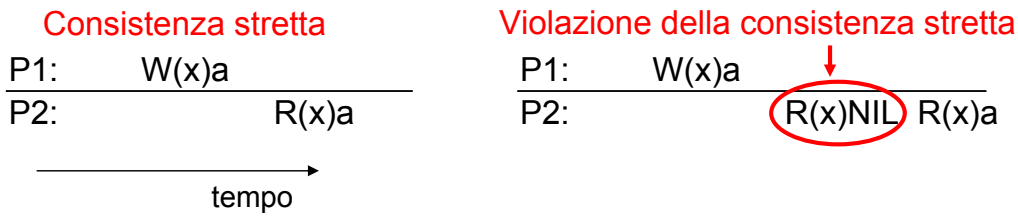
Ordinamento consistente delle operazioni

- Principali modelli di consistenza basati sull'ordinamento delle operazioni sui dati condivisi e replicati
  - Consistenza stretta
  - Consistenza sequenziale
  - Consistenza causale
- Le repliche devono accordarsi su quale sia l'ordinamento globale degli aggiornamenti prima di renderli permanenti

Consistenza stretta

Qualsiasi read su un dato x ritorna un valore corrispondente al risultato della write su x più recente

- Comportamento di 2 processi che operano sullo stesso dato in un modello di consistenza stretta
  - $W_i(x)a$: operazione di scrittura da parte del processo P_i sul dato x con valore a
  - $R_i(x)b$: operazione di lettura da parte del processo P_i sul dato x con valore restituito b



- Richiede un orologio globale
- Le write sono viste istantaneamente da tutti i processi
- E' il modello di consistenza più forte

Consistenza sequenziale

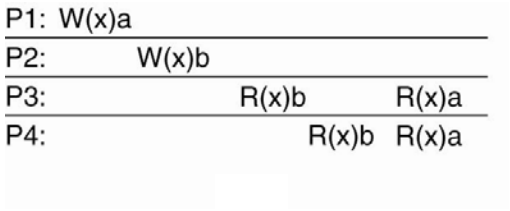
Il risultato di una qualunque esecuzione è uguale a quello ottenuto se le operazioni (di read e write) da parte di tutti i processi sull'archivio di dati fossero eseguite

- ***in qualche ordine sequenziale*** e …
- le operazioni di ogni singolo processo appaiono comunque in questa sequenza ***nell'ordine specificato dal suo programma***

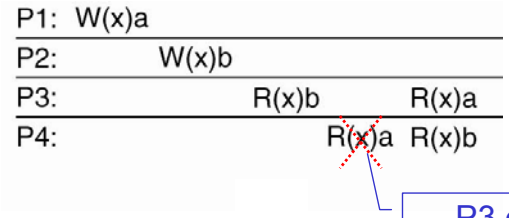
- Quando i processi sono in esecuzione concorrente, qualunque alternanza di operazioni è accettabile, ma *tutti i processi vedono la* ***stessa alternanza di operazioni***
- E' un indebolimento della consistenza stretta

Consistenza sequenziale (2)

- Esempio di sequenza di operazioni valida in un archivio di dati sequenzialmente consistente



- Esempio di sequenza di operazioni non valida in un archivio di dati sequenzialmente consistente



P3 e P4 leggono le scritture fatte da P1 e P2 in ordine diverso

Consistenza sequenziale (3)

- 3 processi in esecuzione concorrente
  - Variabili x, y e z (inizializzate a 0) memorizzate in un archivio di dati sequenzialmente consistente
  - 720 (=6!) possibili sequenze di esecuzione ma solo 90 (=3*120/4) sono valide; 4 di queste sono mostrate in figura

Process P1	Process P2	Process P3
x ← 1;	y ← 1;	z ← 1;
print(y, z);	print(x, z);	print(x, y);

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

000000 e 001001 sono esempi di firme non consentite

Consistenza causale

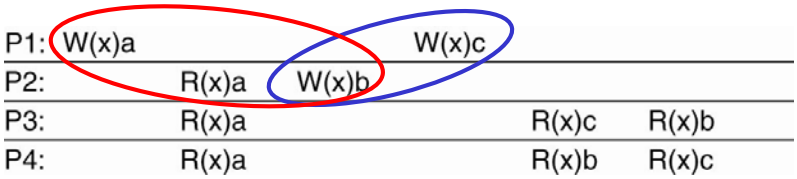
Operazioni di write che sono potenzialmente in relazione di causa/effetto devono essere viste da tutti processi nello stesso ordine. Operazioni di write concorrenti possono essere viste in ordine differente da processi differenti

- Se due processi scrivono simultaneamente su due variabili, le due write non sono causalmente correlate (write concorrenti)
- read seguita da write possono essere operazioni potenzialmente causalmente correlate
  - Se P1 scrive x e P2 legge x e usa il suo valore per scrivere y, la lettura di x e la scrittura di y sono causalmente correlate

- E' un indebolimento della consistenza sequenziale, in quanto distingue tra le operazioni che sono potenzialmente in relazione di causa/effetto e quelle che non lo sono

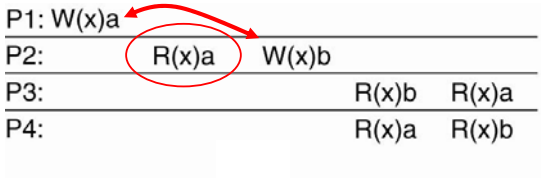
Consistenza causale (2)

- Esempio di sequenza valida in un archivio di dati causalmente consistente, ma non in un archivio sequenzialmente consistente
  - $W_2(x)b$ e $W_1(x)c$ sono write concorrenti: possono essere viste dai processi in ordine differente
  - $W_1(x)a$ e $W_2(x)b$ sono in relazione di causa/effetto

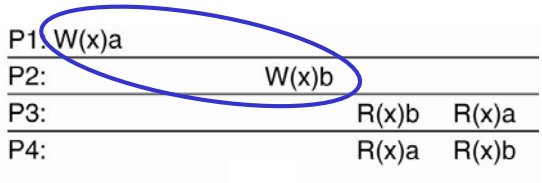


Consistenza causale (3)

- Esempio di sequenza di operazioni non valida in un archivio di dati causalmente consistente
  - $W_1(x)a$ e $W_2(x)b$ sono in relazione di causa/effetto: devono essere viste da tutti i processi nello stesso ordine



- Esempio di sequenza di operazioni valida in un archivio di dati causalmente consistente
  - Ma non valida in un archivio sequenzialmente consistente



Sintesi dei modelli di consistenza

- Modelli di consistenza data-centrici basati sull'ordinamento delle operazioni e che non usano variabili di sincronizzazione

Consistenza	Descrizione
Stretta	Tutti i processi vedono gli accessi condivisi nello stesso ordine assoluto di tempo
Sequenziale	Tutti i processi vedono gli accessi condivisi nello stesso ordine; gli accessi non sono ordinati temporalmente
Causale	Tutti i processi vedono gli accessi condivisi correlati causalmente nello stesso ordine

Consistenza continua

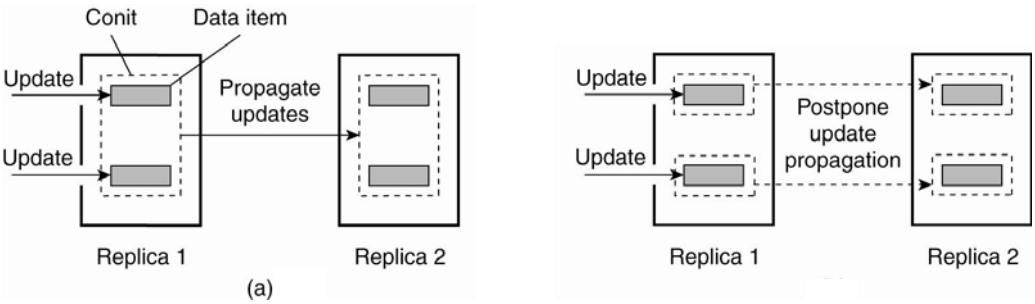
- Come definire una metrica per misurare la consistenza tra repliche?
  - Metrica basata *sul valore*: deviazione nei valori numerici delle repliche
  - Metrica basata *sul tempo*: deviazione nell'età delle repliche
    - Differenza tra tempo di aggiornamento su A e su B (essendo A e B due repliche)
  - Metrica basata *sull'ordinamento*: deviazione relativa all'ordine di esecuzione delle operazioni di scrittura
- Modello di consistenza continua: imporre dei limiti a tutti e 3 i tipi di deviazione
  - Deviazione pari a 0 per tutte e 3 le metriche → consistenza stretta

Riferimento: H. Yu, A. Vahdat, "Design and evaluation of a conit-based continuous consistency model for replicated services", *ACM TOCS*, 2002.

Conit

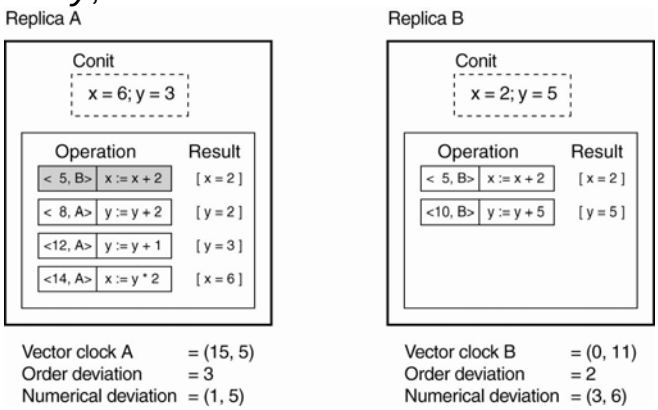
- Un'applicazione può specificare il livello desiderato di consistenza continua tramite *conit*
  - Conit: unità di consistenza → specifica l'unità di dati su cui misurare la consistenza

- Tradeoff nella scelta della granularità opportuna della conit
  - Conit troppo fine: lavoro aggiuntivo per gestire le conit
  - Conit troppo grossa: falsa condivisione



Conit: esempio

- Consideriamo il seguente esempio
  - 2 repliche A e B che operano su una conit contenente le variabili *x* e *y*, inizializzate a 0



  - *B* invia ad A l'operazione [<5,*B*>: *x* := *x* + 2]; A ha reso questa operazione permanente
  - A ha 3 operazioni provvisorie → deviazione *nell'ordinamento*=3
  - A non ha visto 1 operazione da B, che determina una differenza massima tra i valori pari a 5 → deviazione numerica *nel valore* pari a (1,5)

Raggruppare le operazioni

- I modelli di consistenza basati sull'ordinamento delle operazioni ed analizzati finora sono stati definiti inizialmente per sistemi multiprocessore a memoria condivisa
  - Granularità fine, a livello delle operazioni di lettura e scrittura
- Modelli di consistenza più deboli aumentano la granularità, considerando intere serie di operazioni di lettura e scrittura
  - Assumono che ogni serie di operazioni sia raggruppata da operazioni accompagnatorie su *variabili di sincronizzazione* (lock)
  - L'effetto della serie di operazioni viene reso noto a tutti i processi
- Modelli di consistenza che usano variabili di sincronizzazione:
  - Consistenza debole
  - Consistenza release (non esaminata)
  - Consistenza entry

Consistenza debole

- I processi usano variabili di sincronizzazione che permettono di sincronizzare tutte le copie locali dell'archivio di dati tramite l'operazione synchronize (S)
  - Quando viene effettuata una sincronizzazione, tutte le write precedenti alla sincronizzazione vengono esportate verso gli altri processi e tutte le write di questi ultimi vengono importate
- Proprietà:
  - Gli accessi a variabili di sincronizzazione associate ad un archivio di dati sono sequenzialmente consistenti
  - Nessun accesso ad una variabile di sincronizzazione è permesso fino a che tutte le write precedenti non sono state completate su tutte le copie
  - Nessun accesso ai dati (operazione di read o write) è permesso fino a che non sono state effettuati tutti i precedenti accessi a variabili di sincronizzazione

Consistenza debole (2)

- *In pratica*: non interessa che le read o le write di una sequenza di operazioni siano immediatamente note agli altri processi; si vuole soltanto che sia noto l'effetto della serie di operazioni
- Esempio di sequenza di operazioni valida in un archivio di dati debolmente consistente
  - P2 e P3 leggono prima della sincronizzazione: l'ordine può essere diverso

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

- Esempio di sequenza di operazioni non valida in un archivio di dati debolmente consistente
  - P2 si sincronizza prima di leggere: quindi deve leggere b!

P1: W(x)a	W(x)b	S		
P2:			S	R(x)a

Consistenza entry

- Sono definite due operazioni di sincronizzazione:
  - acquire (Acq, acquisizione): per segnalare l'ingresso in una sezione critica e aggiornare tutte le copie dei dati condivisi
  - release (Rel, rilascio): per segnalare l'uscita da una sezione critica
- Queste operazioni sostituiscono e specializzano l'operazione synchronize della consistenza debole, differenziando tra sincronizzazione prima di leggere i dati o dopo averli scritti

Consistenza entry (2)

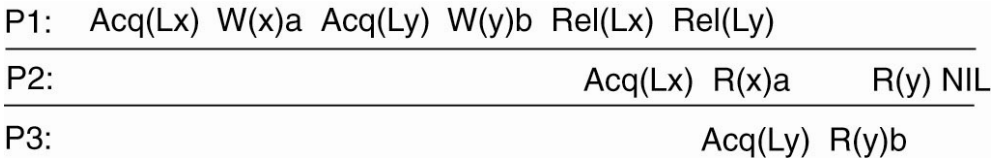
- Regole da rispettare per garantire la consistenza entry di un archivio di dati:
  1. Un processo non può eseguire un'operazione di acquire di una variabile di sincronizzazione finché non sono stati effettuati tutti gli aggiornamenti dei dati protetti condivisi per quel processo
     - Significa che quando viene eseguita una acquire, tutte le modifiche remote ai dati protetti condivisi dalla variabile associata alla acquire devono essere rese visibili
  2. Prima che sia possibile l'accesso esclusivo ad una variabile di sincronizzazione da parte di un processo, nessun altro processo può possedere la variabile di sincronizzazione, neanche in modo non esclusivo
     - Significa che prima di aggiornare un dato condiviso, un processo deve entrare in una sezione critica in modo esclusivo per evitare che un altro processo possa aggiornare il dato condiviso nello stesso istante

Consistenza entry (3)

- Regole da rispettare per garantire la consistency entry di un archivio di dati distribuito *(continua)*
  3. Dopo che è stato effettuato l'accesso esclusivo ad una variabile di sincronizzazione, nessun altro processo può accedere a quella variabile di sincronizzazione in modo non esclusivo finché il proprietario di quella variabile non abbia registrato le modifiche
     - Significa che se un processo vuole entrare in una sezione critica in modo non esclusivo, deve controllare con il proprietario della variabile di sincronizzazione di accedere alle copie più aggiornate dei dati condivisi

Consistenza entry (4)

- Esempio di sequenza di operazioni valida in un archivio di dati con consistenza entry
  - Ad ogni dato condiviso è associata una variabile di sincronizzazione



Sintesi dei modelli di consistenza (2)

- Modelli di consistenza data-centrici che usano variabili di sincronizzazione

Consistenza	Descrizione
Debole	I dati condivisi possono essere considerati consistenti solo dopo una sincronizzazione
Entry	I dati condivisi riguardanti una sezione critica possono essere considerati consistenti all'ingresso della sezione critica

Modelli di consistenza client-centrici

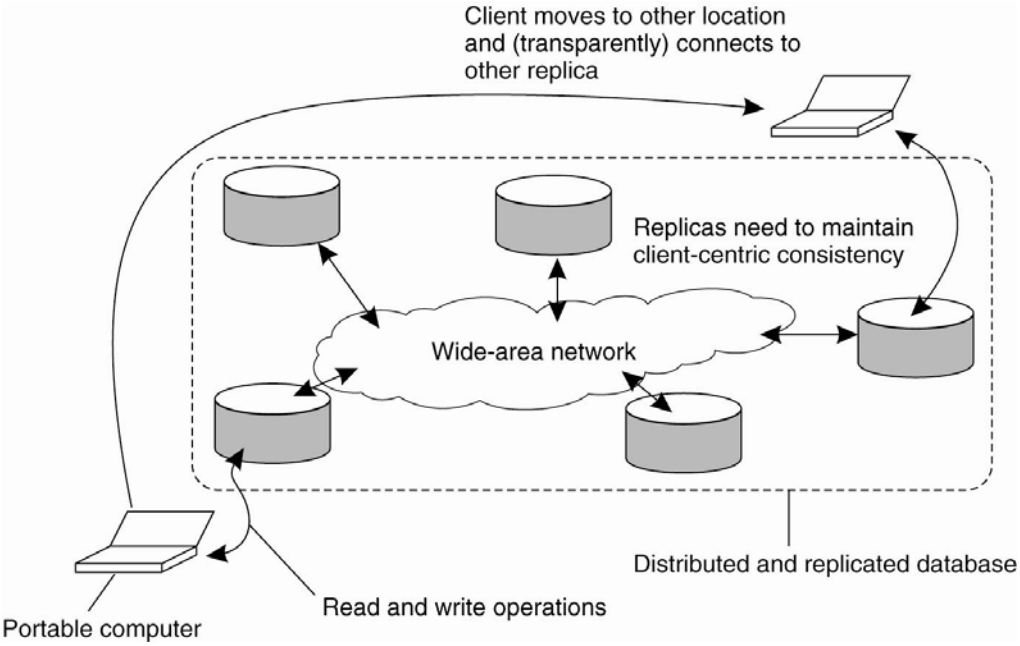
- Consistenza monotonic-read
- Consistenza monotonic-write
- Consistenza read-your-write
- Consistenza writes-follow-reads

- *Obiettivo*: mostrare come è possibile evitare la consistenza a livello di sistema, concentrandosi su cosa vuole un client specifico (invece di cosa dovrebbe essere mantenuto dai server)
  - Nessuna garanzia di consistenza relativamente agli accessi concorrenti da parte di altri client

Consistenza per utenti mobili

- Esempio: consideriamo una base di dati distribuita alla quale un utente accede tramite un portatile ed assumiamo che il portatile agisca da front end verso la base di dati
  - Nella posizione A l'utente accede al server locale della base di dati, eseguendo letture ed aggiornamenti
  - Nella posizione B l'utente continua a lavorare ma, a meno che non acceda allo stesso server della posizione A, può notare inconsistenze
    - Gli aggiornamenti in A possono non essere stati ancora propagati a B
    - L'utente sta leggendo entry più nuove di quelle disponibili in A
    - Gli aggiornamenti in B possono essere in conflitto con quelli in A
- *Osservazione*: l'utente desidera solo che le entry aggiornate e/o lette precedentemente in A siano in B nello stesso modo in cui le ha lasciate in A ➡ la base di dati apparirà consistente all'utente

Architettura di base



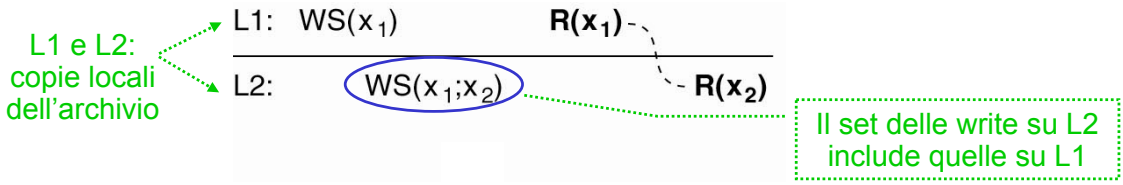
Notazione

- Nei modelli di consistenza client-centrici usiamo la seguente notazione:
  - $x_i[t]$: versione di x sulla copia locale L_i al tempo t
  - $WS(x_i[t])$: sequenza di operazioni di scrittura su L_i che hanno portato come risultato a $x_i[t]$
  - $WS(x_i[t_1]; x_j[t_2])$: le operazioni in $WS(x_i[t_1])$ sono state eseguite anche su L_j in un tempo successivo t_2
    - Significa che $WS(x_i[t_1])$ è parte di $WS(x_j[t_2])$

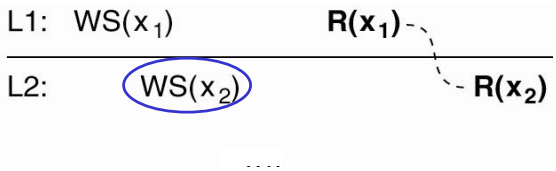
Consistenza monotonic-read

Se un processo legge il valore di un dato *x*, qualunque successiva operazione di lettura su *x* da parte di quel processo restituirà sempre quello stesso valore o un valore più recente

- Esempio di archivio di dati che fornisce la consistenza monotonic-read



- Esempio di archivio di dati che non fornisce la consistenza monotonic-read



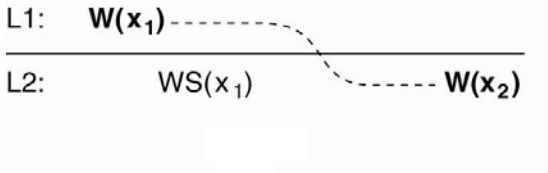
Esempi monotonic-read

- Esempio 1: leggere automaticamente gli aggiornamenti al proprio calendario personale da diversi server
  - La consistenza monotonic-read garantisce che l'utente veda tutti gli aggiornamenti, senza importare da quale server avvenga la lettura automatica
- Esempio 2: leggere (senza modificare) la posta in arrivo mentre ci si sposta
  - Ogni volta che l'utente si connette ad un diverso mail server, il server carica (almeno) tutti gli aggiornamenti relativi alla mailbox dell'utente dal server usato precedentemente

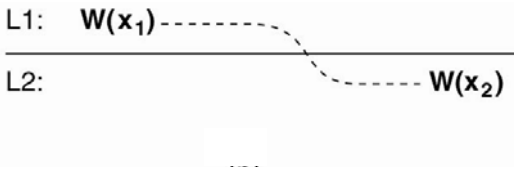
Consistenza monotonic-write

Un'operazione di scrittura da parte di un processo su un dato *x* viene completata prima di qualunque operazione di scrittura successiva su *x* da parte dello stesso processo

- Esempio di archivio di dati che fornisce la consistenza monotonic-write



- Esempio di archivio di dati che non fornisce la consistenza monotonic-write



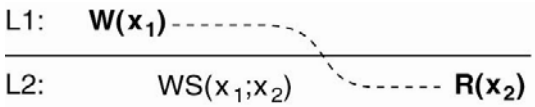
Esempi monotonic-write

- Esempio 1: aggiornare un programma sul server *S*2 ed assicurare che anche tutti i componenti da cui dipendono la compilazione ed il linking siano su *S*2
- Esempio 2: mantenere versioni di file replicati nell'ordine corretto su ogni server (propagando la versione precedente sul server dove è installata la nuova versione)

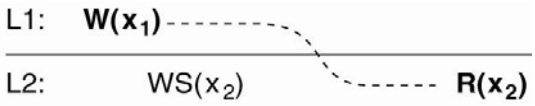
Consistenza read-your-writes

L'effetto di un'operazione di scrittura da parte di un processo su un dato *x* sarà sempre visto da una successiva operazione di lettura di *x* da parte dello stesso processo

- Esempio di archivio di dati che fornisce la consistenza read-your-writes



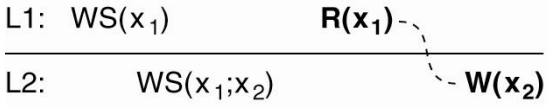
- Esempio di archivio di dati che non fornisce la consistenza read-your-writes



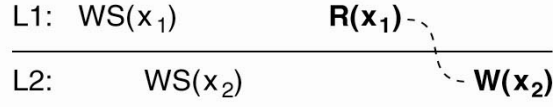
Consistenza writes-follow-reads

Un'operazione di scrittura da parte di un processo su un dato *x* che segue una precedente operazione di lettura di *x* da parte dello stesso processo ha luogo sullo stesso valore di *x* che è stato letto o su un valore più recente

- Esempio di archivio di dati che fornisce la consistenza writes-follow-reads



- Esempio di archivio di dati che non fornisce la consistenza writes-follow-reads



Esempi read-your-writes e writes-follow-reads

- Esempio di consistenza read-your-writes: aggiornare la propria pagina Web e garantire che il proprio browser mostri la versione più recente anziché la copia in cache
- Esempio di consistenza writes-follow-reads: vedere i commenti ad un articolo inserito in un newsgroup solo se si ha visto l'articolo originale (una lettura "tira" la corrispondente operazione di scrittura)

Protocolli di distribuzione

- Posizionamento dei server replica
  Dove posizionare le repliche
- Replica e posizionamento dei contenuti
  Dove propagare i contenuti
- Distribuzione dei contenuti
  Che cosa propagare

Posizionamento dei server replica

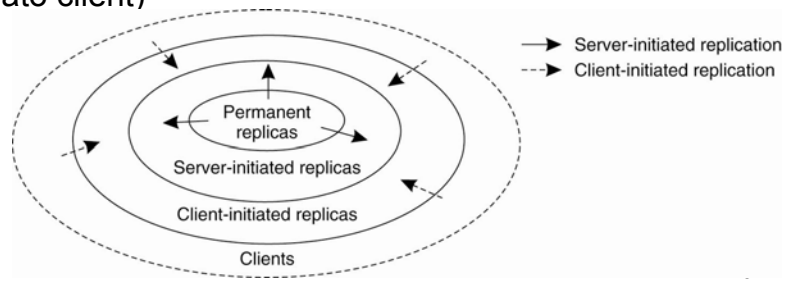
- *Obiettivo*: risolvere un problema di ottimizzazione in cui scegliere le migliori K posizioni tra N ($K < N$)
  - Problema NP-completo (teoria dei grafi: facility location e minimum K-median problem): occorre individuare euristiche
- Soluzione Greedy: selezionare la posizione migliore tra le N-k disponibili che minimizza la distanza media tra quel server replica ed i client; posizionare poi un altro server replica, considerando già fissati i precedenti
  - Il primo server replica posizionato (k=0) è quello che minimizza la distanza media verso tutti i client
  - Elevato costo computazionale (superiore a $O(N^2)$) e conoscenza dettagliata della topologia e dei client
- Soluzione max-AS/max-router fanout placement: selezionare il k-esimo sistema autonomo (AS) più grande e posizionare il server replica sul router dell'AS con la migliore connettività
  - Elevato costo computazionale (superiore a $O(N^2)$)

Posizionamento dei server replica (2)

- Soluzione HotZone: dopo aver posizionato i nodi in uno spazio geometrico M-dimensionale, in modo che la distanza tra due nodi rispecchi la loro latenza
  - Passo 1: selezionare le K regioni con la maggiore densità
    - Una regione è un gruppo di nodi con una bassa latenza tra loro
  - Passo 2: posizionare un server in ognuna delle K regioni, scegliendo come replica server il nodo della regione che minimizza la distanza con gli altri nodi
  - Minore costo computazionale (pari a $O(N * \max\{\log(N), K\})$)

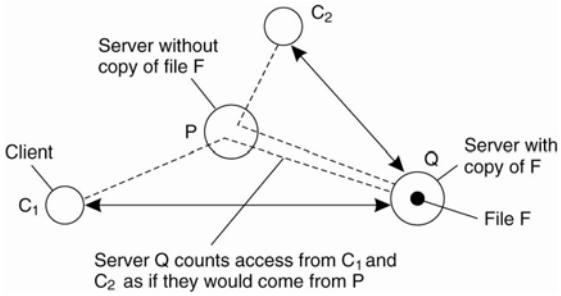
Replica e posizionamento dei contenuti

- Modello: i contenuti possono essere dati, codice o entrambi
- 3 diversi tipi di replica:
  - Repliche permanenti
    - I server replica su cui posizionare i contenuti sono statici
  - Repliche server-initiated
    - I server replica vengono scelti dinamicamente su richiesta di un altro server per migliorare le prestazioni, ad es. per gestire un flash crowd nell'ambito di servizi di Web hosting
  - Repliche client-initiated
    - I server replica vengono scelti dinamicamente su richiesta di un client (cache lato client)



Repliche server-initiated

- Analizziamo un algoritmo per replicare risorse Web (statiche) su server replica posizionati vicino a client che hanno inviato molte richieste per quelle risorse
- Per ogni risorsa *F*, il server *Q* mantiene un contatore di accessi $cnt_Q(P,F)$, aggregato considerando il server *P* più vicino ai client richiedenti la risorsa



  - $cnt_Q(P,F)$ <= soglia D → cancellare la risorsa da Q
  - $cnt_Q(P,F)$ >= soglia R → replicare la risorsa su P
  - *soglia* $D < cnt_Q(P,F) <$ soglia R → migrare la risorsa su P

Riferimento: M. Rabinovich *et al*, "A dynamic object replication and migration protocol for an Internet hosting service", *Proc. ICDCS 1999*.

Distribuzione dei contenuti

- *Obiettivo*: propagare i contenuti aggiornati alle repliche
- Considerando solo una combinazione client-server, ci sono 3 possibili approcci:
  1. Propagare solo la notifica/invalidazione di un aggiornamento usando un *protocollo di invalidazione* (approccio usato spesso per le cache)
  2. Trasferire i dati da una copia all'altra
  3. Propagare le operazioni di aggiornamento alle altre copie (anche detta *replicazione attiva*)
- *Osservazione*: nessun approccio è il migliore, ma la scelta dipende fortemente dalla banda disponibile (B) e dal rapporto (R) tra operazioni di lettura e di scrittura
  - Notifica: conviene se B scarsa e R basso
  - Dati: conviene se R alto
  - Operazioni: conviene se B scarsa

Distribuzione dei contenuti: push e pull

- Push o pull degli aggiornamenti?
- Approccio push
  - Iniziato dal server (anche approccio *basato sul server*)
  - Spesso usati tra repliche permanenti e repliche server-initiated, ma utilizzabili anche per repliche client-initiated
  - Aggiornamenti propagati indipendentemente dal fatto che i client li richiedano
  - Usato quando le repliche hanno la necessità di mantenere un grado di consistenza relativamente alto
  - Efficiente quando il rapporto tra operazioni di lettura e scrittura *per replica* è relativamente alto
- Approccio pull
  - Iniziato dal client (anche approccio *basato sul client*)
  - Spesso usati dalle repliche client-initiated
  - Efficiente quando il rapporto tra operazioni di lettura e scrittura *per replica* è relativamente basso

Distribuzione dei contenuti: push e pull (2)

- Confronto tra approccio push e approccio pull
  - Sistema con un singolo server e molteplici client

Fattore	Push	Pull
Stato sul server	Elenco delle repliche (server stateful)	Nessuno (server stateless)
Messaggi inviati	Aggiornamento (più eventuale prelievo successivo degli aggiornamenti)	Interrogazione ed aggiornamento
Tempo di risposta sul client	Immediato (o tempo di prelievo dell'aggiornamento)	Tempo di prelievo dell'aggiornamento

- Unicast o multicast per gli aggiornamenti?
  - Con approccio push: multicast può essere efficiente
  - Con approccio pull: unicast

Distribuzione dei contenuti: lease

- Una forma ibrida di propagazione degli aggiornamenti derivante dalla combinazione degli approcci push e pull è basata sui lease
  - Lease: un contratto in cui il server si impegna ad inviare (*push*) gli aggiornamenti dei contenuti al client per un certo tempo
  - Scaduto il tempo, il client deve interrogare il server e prelevare i dati aggiornati (*pull*)
- Lease adattativi: evoluzione in cui il tempo di scadenza dipende dal comportamento del sistema
  - Lease basati sull'età: un dato che non è stato modificato per un lungo periodo, non cambierà presumibilmente a breve → lease di lunga durata
  - Lease basati sulla frequenza di rinnovamento: più spesso un client richiede un dato, più duraturo è il relativo lease concesso
  - Lease basati sull'overhead di stato sul server: maggiore è il carico sul server, minore è la durata dei lease concessi

Lease adattativi proposti nell'ambito del Web caching: con quale obiettivo?

Protocolli di consistenza

- Protocollo di consistenza: implementazione di uno specifico modello di consistenza
- Protocolli per la consistenza continua
- Protocolli per la consistenza sequenziale
  - Protocolli primary-based
    - Operazioni di scrittura eseguite su una sola replica, che successivamente assicura che gli aggiornamenti siano opportunamente ordinati ed inoltrati alle altre repliche
    - Replicazione **passiva**
  - Protocolli replicated-write
    - Operazioni di scrittura eseguite su molteplici repliche
    - Replicazione **attiva** *oppure*
    - Protocolli quorum-based

Protocolli per consistenza continua

- Analizziamo come limitare la deviazione numerica nei valori delle repliche
- Consideriamo un singolo dato x ed indichiamo con $weight(W)$ il cambiamento numerico nel suo valore dopo un'operazione di scrittura W
  - Assumiamo che $\forall\ W$: $weight(W) > 0$
- W è inizialmente sottomessa ad una tra le N repliche, indicata da $origin(W)$. $TW[i, j]$ sono le operazioni di scrittura eseguite dal server S_i originate dal server S_j:

  $$TW[i, j] = \Sigma\{weight(W) \mid origin(W)=S_j \ \&\ W \in logfile(S_i)\}$$
- Il valore reale $v(t)$ di x al tempo t è:

  $$v(t) = v_{init} + \Sigma_{k=1}^{N} TW[k, k]$$
- Il valore v_i di x sulla replica S_i è:

  $$v_i = v_{init} + \Sigma_{k=1}^{N} TW[i, k] \qquad \text{con } v_i <= v(t)$$

Protocolli per consistenza continua (2)

- *Problema*: occorre assicurare che $v(t)$-$v_i < \delta_i$ per ogni server S_i
- *Approccio*: ogni server S_k mantiene una vista $TW_k[i, j]$ di quello che crede sia il valore di $TW[i, j]$ per S_i
  - Questa informazione può essere diffusa tramite gossiping quando viene propagato un aggiornamento
- *Osservazione*: $0 \leq TW_k[i, j] \leq TW[i, j] \leq TW[j, j]$
- *Soluzione*: S_k invia ad S_i le operazioni nel suo logfile quando si accorge che $TW_k[i, k]$ si sta allontanando troppo da $TW[k, k]$, in particolare quando

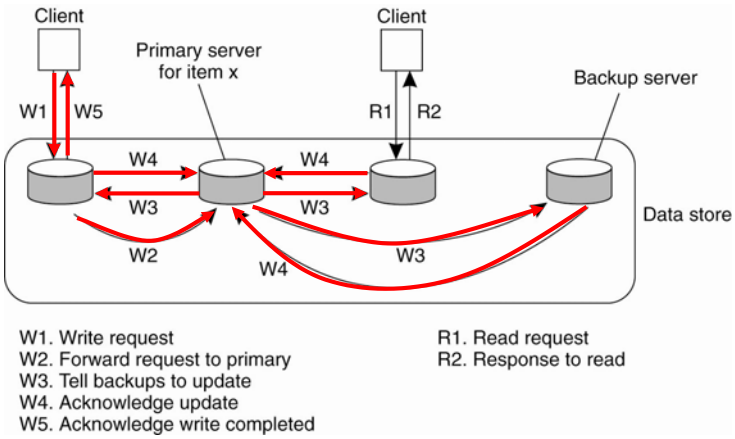
  $$TW[k, k] - TW_k[i, k] > \delta_i/(N\text{-}1)$$

- Per limitare la deviazione nella scadenza, si può seguire un approccio simile, tenendo traccia di quello che è stato visto da S_i

Protocolli primary-based

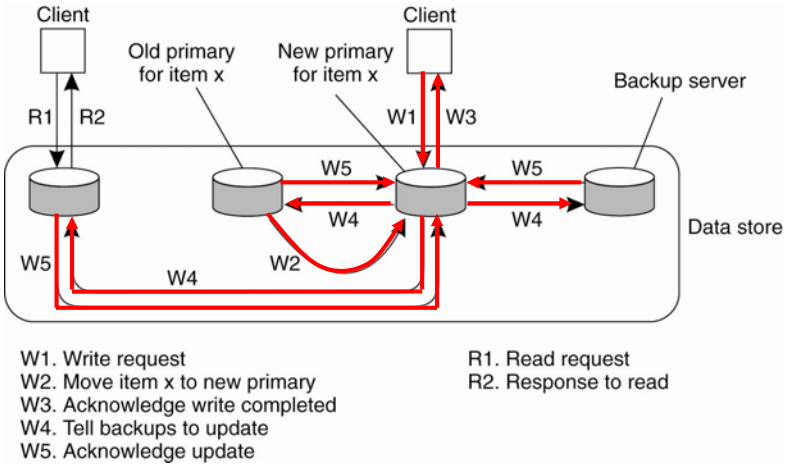
- Ad ogni dato *x* è associata una replica primaria che ha il compito di coordinare le operazione di scrittura su *x*
  - Anche detti protocolli primary-backup
- Protocolli primary-based di tipo remote-write
  - Tutte le operazioni di scrittura su *x* devono essere inoltrate alla replica primaria
  - Le operazioni di lettura possono essere eseguite localmente
- Protocolli primary-based di tipo local-write
  - La copia primaria di x migra verso il processo che vuole eseguire un'operazione di scrittura

Primary-based: protocolli remote-write



- Protocolli tipicamente usati nei DB distribuiti e nei file system distribuiti che richiedono un elevato grado di tolleranza ai guasti
  - Repliche posizionate sulla stessa LAN
- Aggiornamento della replica primaria in modo bloccante o non bloccante
  - Non bloccante: maggiori prestazioni, minore tolleranza ai guasti

Primary-based: protocolli local-write



- Applicabile anche in ambito mobile computing
  - Invio di tutti i dati rilevanti all'utente prima della disconnessione e successivo aggiornamento alla riconnessione

Replicazione attiva

- Ogni replica esegue le operazioni di scrittura
- Per eseguire le operazioni di scrittura su ogni replica nello stesso ordine occorre un meccanismo di multicast totalmente ordinato
  - Implementabile in modo distribuito con gli orologi logici di Lamport (già esaminato come)
  - Oppure introducendo un coordinatore centralizzato, detto *sequencer*, che assegna ad ogni operazione di scrittura un numero di sequenza univoco

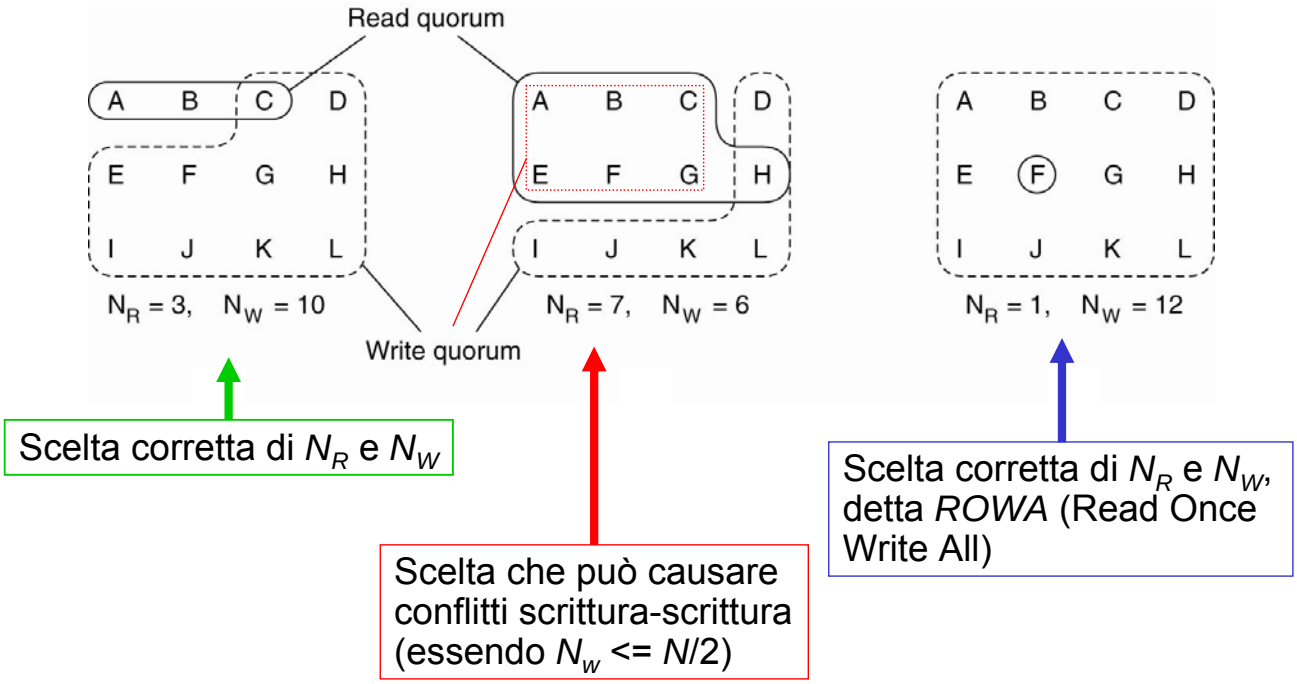
Protocolli quorum-based

- Protocolli basati su un meccanismo di votazione
- Supponiamo che esistano N repliche
- Prima di effettuare un'operazione di lettura è richiesto un quorum per la lettura N_R
- Prima di effettuare un'operazione di scrittura è richiesto un quorum per la scrittura N_W
- Per N_R e N_W valgono le condizioni

  $N_R + N_W > N$ (per impedire conflitti lettura-scrittura)

  $N_w > N/2$ (per impedire conflitti scrittura-scrittura)

Protocolli quorum-based (2)



Protocolli per la consistenza client-centrica

- Per implementare la consistenza client-centrica
  - Ad ogni operazioni di scrittura viene assegnato un identificatore globale univoco
  - Ogni client mantiene
    - Un *read set*: scritture rilevanti per le sue operazioni di lettura
    - Un *write set*: operazioni di scrittura eseguite dal client
  - I client passano ai server il proprio read o write set quando richiedono un'operazione
  - Se al server manca qualche scrittura, contatta gli altri server per essere aggiornato prima di eseguire l'operazione richiesta dal client
  - Dopo ogni operazione, il read set ed il write set sono aggiornati
- Poiché il read set ed il write set possono diventare molto grandi, è possibile definire un'implementazione della consistenza client-centrica più efficiente
  - Basata su timestamp vettoriali

Efficienza della consistenza client-centrica

- Per efficienza i write set sono rappresentati implicitamente
- Ogni server S_i mantiene un timestamp vettoriale WVC_i
- $WVC_i[j]$ è il timestamp dell'operazione di scrittura più recente originata da S_j ed eseguita da S_i
- Anche i client mantengono i read e write set come timestamp vettoriali SVC che sono passati ai server insieme alla richiesta
- Il server S_i confronta SVC con WVC_i per capire se ha bisogno di eseguire delle operazioni di scrittura mancanti
- Dopo un'operazione, il server S_i restituisce al client il proprio timestamp WVC_i, che è usato dal client per aggiornare il proprio timestamp SVC

$$SVC[j] \leftarrow max\{SVC[j], WVC_i[j]\}$$