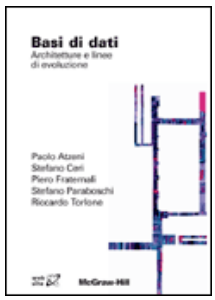
Architetture Distribuite

Capitolo 3



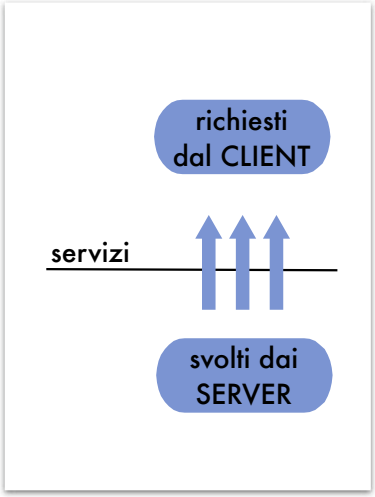
Basi di dati – Architetture e linee di evoluzione
P. Atzeni, S. Ceri, P. Fraternali, S. Paraboschi, R. Torlone

Sommario

- Architetture client-server
- Basi di dati distribuite
- Basi di dati parallele
- Basi di dati replicate

Client–Server

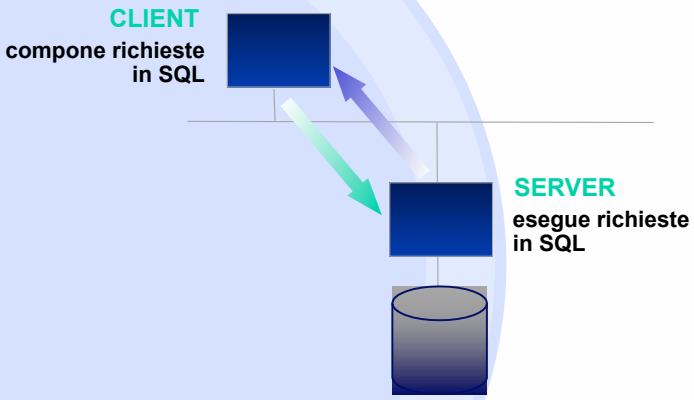
- Tecnica per strutturare sistemi software
- Viene resa "pubblica" una "interfaccia di servizi"
- Due tipologie di sistemi
  - **Client**: richiedono i servizi
  - **Server**: forniscono i servizi



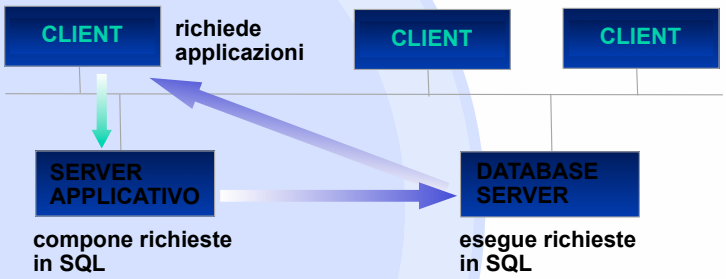
Client-server nei sistemi informativi

- Separazione funzionale ideale
  - **Client**: presentazione dell'informazione
  - **Server**: gestione dei dati
- SQL: il linguaggio ideale per separare gli ambienti
  - **Client**: formula query, elabora risultati
  - **Server**: segue query
  - **Rete**: trasferisce i comandi di attivazione (es. procedure SQL) ed i risultati

Architettura client-server classica



Architettura con server applicativo



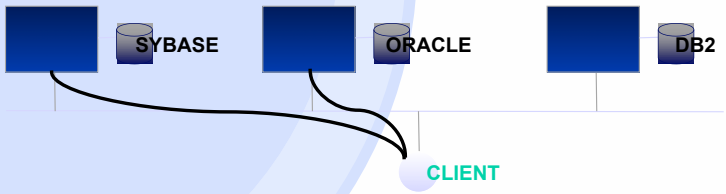
Database distribuiti

Motivazioni

- Natura intrinsecamente distribuita delle organizzazioni
- Evoluzione degli elaboratori
  - aumento della capacità elaborativa
  - riduzione di prezzo
- Evoluzione della tecnologia dei DBMS
- Standard di interoperabilità

Tipologie di basi di dati distribuite

- Si distinguono per tipi di rete:
  - LAN (Local Area Network)
  - WAN (Wide Area Network)
- e per tipi di DBMS:
  - Sistema omogeneo
  - Sistema eterogeneo



Esempi tipici

	LAN	WAN
Omogeneo	Applicazioni gestionali e finanziarie	Sistemi di prenotazione, applicazioni finanziarie
Eterogeneo	Applicazioni gestionali interfunzionali	Sistemi di prenotazione integrati, sistemi interbancari

Problemi delle basi di dati distribuite

- Autonomia e cooperazione
- Trasparenza
- Efficienza
- Affidabilità

Autonomia e cooperazione

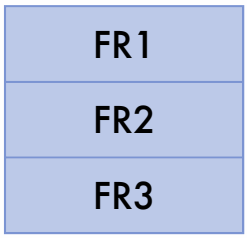
- Autonomia
  - Una reazione ai "Centri EDP"
  - Portare competenze e controllo laddove vengono gestiti i dati
  - Rendere la maggior parte delle applicazioni NON distribuite (!)
- Cooperazione
  - Alcune applicazioni sono intrinsecamente distribuite e richiedono l'accesso a più basi di dati

Frammentazione dei dati

- Scomposizione delle tabelle in modo da consentire la loro distribuzione
- Proprietà:
  - Completezza
  - Ricostruibilità

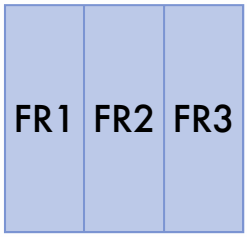
Frammentazione orizzontale

- Frammenti:
  insiemi di tuple
- Completezza:
  presenza di tutte le tuple
- Ricostruzione:
  unione



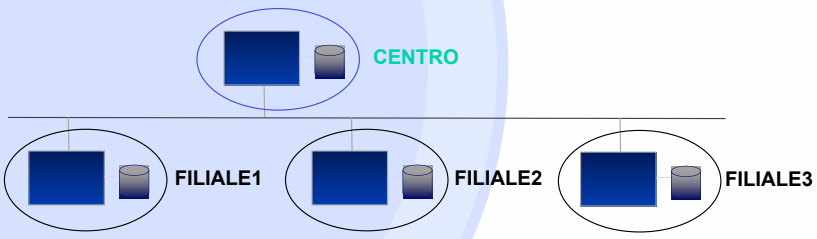
Frammentazione verticale

- Frammenti:
  insiemi di attributi
- Completezza:
  presenza di tutti gli attributi
- Ricostruzione:
  join sulla chiave



Esempio

- Conti correnti bancari:
  - CONTO-CORRENTE (NUM-CC, NOME, FILIALE, SALDO)
  - TRANSAZIONE (NUM-CC, DATA, PROGR, AMMONTARE, CAUSALE)



Frammentazione orizzontale principale

- Basata su selezioni:
  - $R_i = \sigma_{P_i}(R)$
- esempio:
  - CONTO1 = $\sigma_{Filiale=1}$ (CONTO-CORRENTE)
  - CONTO2 = $\sigma_{Filiale=2}$ (CONTO-CORRENTE)
  - CONTO3 = $\sigma_{Filiale=3}$ (CONTO-CORRENTE)

Frammentazione orizzontale derivata

- Basata su join con frammenti principali
  - $S_i = S >< R_i$
- esempio:
  - TRANS1 = TRANSAZIONE >< CONTO1
  - TRANS2 = TRANSAZIONE >< CONTO2
  - TRANS3 = TRANSAZIONE >< CONTO3

Frammentazione verticale

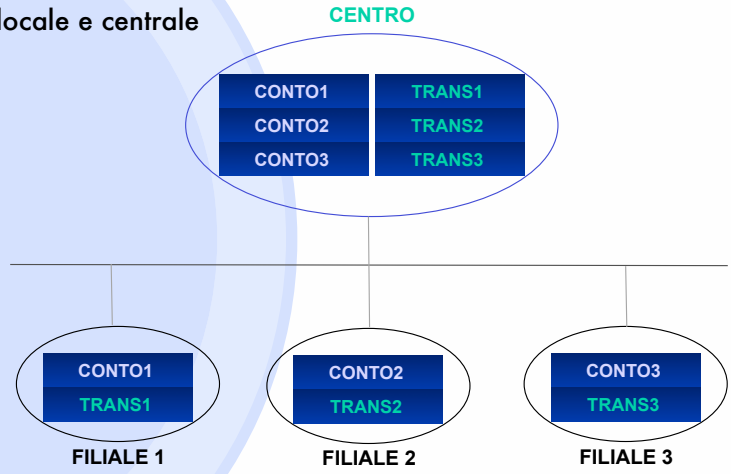
- Basata su proiezioni
  - $R_i = \pi_{A_i}(R)$
- esempio:
  - CONTO1 = $\pi_{\text{NUM-CC, NOME}}$ (CONTO-CORRENTE)
  - CONTO2 = $\pi_{\text{NUM-CC, FILIALE}}$ (CONTO-CORRENTE)
  - CONTO3 = $\pi_{\text{NUM-CC, SALDO}}$ (CONTO-CORRENTE)

Allocazione dei frammenti

- Ogni frammento R_i viene implementato tramite un file fisico ed installato presso un server: allocazione dei frammenti sui server
- La relazione (R) è presente solo in modo virtuale (vista) mentre i frammenti sono memorizzati
- Lo **schema di allocazione** contiene il **mapping** tra frammenti (e relazioni non frammentate) e server su cui sono allocati
  - **non ridondante**: ciascun frammento e/o relazione viene allocato su di un solo server
  - **ridondante**: qualche frammento e/o relazione viene allocato su più di un server

Allocazione dei frammenti (2)

- Esempio:
  - 3 siti periferici ed 1 sito centrale
  - Allocazione locale e centrale
  - Ridondante?



Livelli di trasparenza

- La distinzione tra frammentazione ed allocazione consente di individuare vari livelli di trasparenza nelle applicazioni
- Livelli di trasparenza:
  - Modalità per esprimere interrogazioni offerte dai DMBS a seconda dell'astrazione dal modello fisico utilizzato per la frammentazione e l'allocazione dei dati
  - Trasparenza di **frammentazione**
  - Trasparenza di **allocazione**
  - Trasparenza di **linguaggio**
  - **Assenza** di trasparenza

Trasparenza di frammentazione

- La più "trasparente"!
- Non ci si deve preoccupare del fatto che la base di dati sia o meno distribuita e frammentata (o replicata)
  - L'interrogazione è identica a quella che verrebbe scritta per una relazione non frammentata
- Esempio:
  - QUERY: estrarre il saldo del conto corrente 45
  - SELECT SALDO
    FROM CONTO-CORRENTE
    WHERE NUM-CC=45

Trasparenza di allocazione

- Si conosce la struttura dei frammenti ma non si deve indicare l'allocazione
  - L'interrogazione deve indicare i frammenti (e se è il caso scansionarli sequenzialmente)
  - In caso di copie replicate (allocazione ridondante) non è però necessario specificare quale copia utilizzare (trasparenza di replicazione)
- Esempio:
  - QUERY: Estrarre il saldo del conto corrente 45
  - Due casi: Si conosce o meno a quale filiale appartiene il conto

Trasparenza di allocazione (2)

- QUERY: Estrarre il saldo del conto corrente 45
  1. Conto corrente 45 c/o filiale 1
     - **SELECT** SALDO FROM CONTO1 WHERE NUM-CC=45
  2. Allocazione incerta (probabilmente filale 1...)
     - **SELECT** SALDO FROM CONTO1 WHERE NUM-CLI=45
       IF (NOT FOUND) THEN (
       SELECT SALDO FROM CONTO2 WHERE NUM-CLI=45
       UNION
       SELECT SALDO FROM CONTO3 WHERE NUM-CLI=45
)

Trasparenza di linguaggio

- Si deve conoscere sia la struttura dei frammenti sia la loro allocazione
  - L'interrogazione deve indicare entrambi
- Esempio:
  - QUERY: estrarre il saldo del conto corrente 45 (non si sa la filiale, ma si sa dove sono tutte le filiali)
  - **SELECT** SALDO FROM CONTO1@1 WHERE NUM-CLI=45
    IF (NOT FOUND) THEN (
    SELECT SALDO FROM CONTO2@C WHERE NUM-CLI=45
    UNION
    SELECT SALDO FROM CONTO3@C WHERE NUM-CLI=45
)

Ipotesi: CONTO1 allocato su server 1, CONTO2 e CONTO3 allocati su server C

Query complesse

- Come ci si comporta con query complesse, ad esempio con join?
  - QUERY: Estrarre tutti i movimenti dei conti con saldo negativo

Livello di trasparenza		
Frammentazione	**Allocazione**	**Linguaggio**
SELECT CC-NUM, PROGR, AMMONTARE FROM CONTO-CORRENTE AS C **JOIN** TRANSAZIONE AS T ON C.NUM-CC=T.NUM-CC WHERE SALDO < 0	**SELECT** CC-NUM, PROGR, AMMONTARE FROM CONTO1 **JOIN** TRANS1 ON ... WHERE SALDO < 0 **UNION** **SELECT** CC-NUM, PROGR, AMMONTARE FROM CONTO2 **JOIN** TRANS2 ON ... WHERE SALDO < 0 **UNION** **SELECT** CC-NUM, PROGR, AMMONTARE FROM CONTO3 **JOIN** TRANS3 ON ... WHERE SALDO < 0	**SELECT** CC-NUM, PROGR, AMMONTARE FROM CONTO1@1 **JOIN** TRANS1@1 ON ... WHERE SALDO < 0 **UNION** **SELECT** CC-NUM, PROGR, AMMONTARE FROM CONTO2@C **JOIN** TRANS2@C ON ... WHERE SALDO < 0 **UNION** **SELECT** CC-NUM, PROGR, AMMONTARE FROM CONTO3@C **JOIN** TRANS3@C ON ... WHERE SALDO < 0

Ipotesi: CONTO1 allocato su server 1, CONTO2 e CONTO3 allocati su server C

Update

- Come ci si comporta in caso di update?
  - QUERY: Spostare il conto corrente 45 dalla filiale 1 alla filiale 2

Livello di trasparenza		
Frammentazione	**Allocazione**	**Linguaggio**
UPDATE CONTO-CORRENTE **SET** FILIALE = 2 WHERE NUM-CC=45 AND FILIALE=1	**INSERT** INTO CONTO2 **SELECT** * FROM CONTO1 WHERE NUM-CC=45 **INSERT** INTO TRANS2 **SELECT** * FROM TRANS1 WHERE NUM-CC=45 **DELETE** FROM CONTO1 WHERE NUM-CC=45 **DELETE** FROM TRANS1 WHERE NUM-CC=45	**INSERT** INTO CONTO2@2 **SELECT** * FROM CONTO1 WHERE NUM-CC=45 **INSERT** INTO CONTO2@C **SELECT** * FROM CONTO1 WHERE NUM-CC=45 **INSERT** INTO TRANS2@2 **SELECT** * FROM TRANS1 WHERE NUM-CC=45 **INSERT** INTO TRANS2@C **SELECT** * FROM TRANS1 WHERE NUM-CC=45 (in modo analogo: 2 coppie di **DELETE**)

Ipotesi: CONTO1 allocato su server 1, CONTO2 e CONTO3 allocati su server C

Classificazione Transazioni

- Classificazione basato sulla composizione degli statement SQL che compongono una transazione
  - **Richieste remote**: transazioni si sola lettura (SELECT) ad un solo DBMS remoto
  - **Transazioni remote**: transazioni (SELECT, INSERT, UPDATE, DELETE) ad un solo DMBS remoto
  - **Transazioni distribuite**: transazioni ad un numero generico di DBMS remoti in cui ciascun comando SQL fa riferimento a dati memorizzati su di un solo DBMS
  - **Richieste distribuite**: transazioni arbitrarie su di un numero arbitrario di DMBS remoti

Classificazione Transazioni (2)

- Livelli progressivi di complessità nell'interazione con DBMS
  - **Richieste remote**: un solo DBMS remoto può essere interrogato
  - **Transazioni remote**: estende il caso precedente alle scritture, ma ciascuna transazione scrive su di un solo DBMS
  - **Transazioni distribuite**: estende il precedente, scritture su più nodi possibili, ma ciascun comando SQL è indirizzato ad uno specifico DBMS; richiede **commit a due fasi**
  - **Richieste distribuite**: il più generale e complesso da gestire; richiede **Distributed Query Optimizer**
- Tutte le query al livello di trasparenza di frammentazione sono classificabili come richieste distribuite!

Proprietà transazioni in sistemi distribuiti

- Come estendere ai sistemi distribuiti le tecnologie e le proprietà sinora viste?
- Proprietà ACID
  - ✗ Atomicità: problema principale; sono necessari protocolli di commit specifici (commit a due fasi, a quattro fasi, ...)
  - ✓ Consistenza: vincoli di integrità locali a ciascun DBMS. E' garantita in modo automatico (limite degli attuali DBMS)
  - ✗ Isolamento: controllo di concorrenza con metodo 2PL o TS a ciascuno nodo (vedremo...)
  - ✓ Persistenza: ciascun sistema si occupa di garantire la persistenza dei dati in esso contenuti

Proprietà transazioni in sistemi distribuiti (2)

- Ottimizzatore di interrogazioni distribuite
  - E' necessario solo quando si tratta di processare richieste distribuite. Negli altri casi sono sufficienti gli ottimizzatori dei singoli DBMS remoti
  - Occorre un'ottimizzazione globale che tenga in conto della presenza di vari nodi remoti e della trasmissione via rete dei risultati intermedi
  - Occorre calcolare oltre al resto il costo delle trasmissioni
  - Occorre scegliere su quale copia dei frammenti operare in caso di allocazione ridondante

Concorrenza

- In un sistema distribuito una transazione t_i può eseguire varie sotto-transazioni t_{ij}, dove l'indice j indica il nodo del sistema su cui la sotto-transazione opera. Ad esempio t_1 e t_2 operano su due oggetti x ed y allocati sui nodi 1 e 2:
  - t_1: $r_{11}(x)$ $w_{11}(x)$ $r_{12}(y)$ $w_{12}(y)$
  - t_2: $r_{22}(y)$ $w_{22}(y)$ $r_{21}(x)$ $w_{21}(x)$
- La serializzabilità locale non è una garanzia sufficiente per la serializzabilità a livello distribuito! Ad esempio:
  - S_1: $r_{11}(x)$ $w_{11}(x)$ $r_{21}(x)$ $w_{21}(x)$
  - S_2: $r_{22}(y)$ $w_{22}(y)$ $r_{12}(y)$ $w_{12}(y)$
  - Non sono CSR!

Serializzabilità globale

- La **serializzabilità globale** di due transazioni distribuite richiede l'esistenza di un **unico schedule seriale S**, che coinvolga tutte le transazioni del sistema, e che sia equivalente a tutti gli schedule locali Si
  - per ogni nodo i, la proiezione S[i] di S contenente le sole azioni che avvengono sul nodo i deve essere perciò equivalente e Si
  - Proprietà difficile da verificare con view- o conflict- equivalenza
  - Automaticamente verificata con 2PL o TS (Metodo di Lamport)!

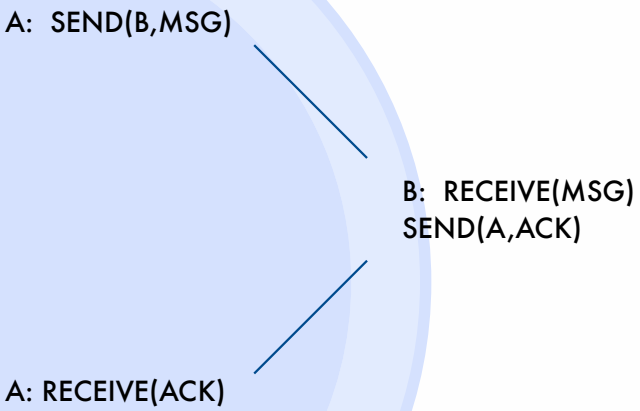
Atomicità

- Problema nasce dal fatto che tutti i nodi che partecipano ad una transazione devono giungere alla stessa decisione circa la transazione: commit o abort
  - Servono quindi dei protocolli di commit che consentano ad una transazione di raggiungere correttamente la decisione finale
  - Difficoltà portate dalla possibile presenza di guasti nelle trasmissioni (anche dello stesso protocollo di commit!)

Guasti in un sistema distribuito

- Caduta di nodi
- Perdita di messaggi
- Interruzione della rete

Perdita di messaggi



Commit a due fasi

- Protocollo per garantire l'atomicità di sotto-transazioni distribuite
- Protagonisti del protocollo
  - Un coordinatore (Transaction Manager, **TM**)
  - Due o più partecipanti (Resource Manager, **RM**)
- Analogia con matrimonio
  - Prima fase: dichiarazione di intenti
  - Seconda fase: dichiarazione di matrimonio

Commit a due fasi

- Il protocollo di commit a due fasi si svolge tramite un rapido scambio di messaggi tra TM e RM
- Per rendere il protocollo resistente ai guasti, RM e TM scrivono alcuni nuovi record nei loro log

Record nei log

- TM (coordinatore)
  - PREPARE: identità dei partecipanti
  - GLOBAL COMMIT/ABORT: decisione
  - COMPLETE: termine del protocollo
- RM (Partecipanti)
  - READY: disponibilità al commit
  - LOCAL COMMIT/ABORT: decisione ricevuta

Fase 1

TM: WRITE-LOG(PREPARE)
SET TIME-OUT SEND
(**RM$_i$**,PREPARE)

RM$_i$: RECEIVE(**TM**,PREPARE)
IF OK THEN WRITE-LOG(READY)
MSG=READY
ELSE MSG=NO
SEND (**TM**,READY)

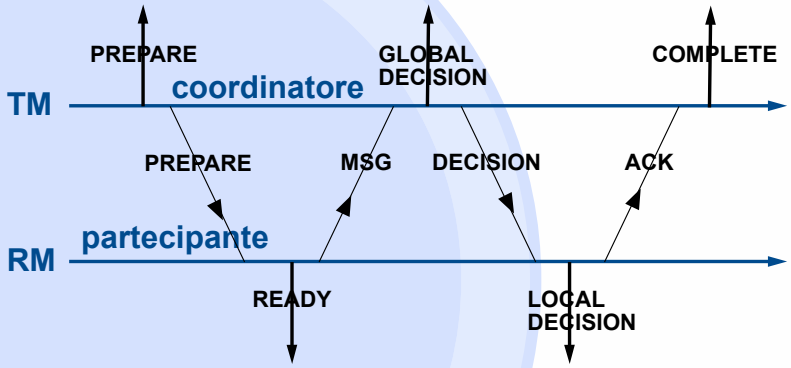
Fase 2

TM: RECEIVE(**RM$_i$**,MSG)
IF TIME-OUT OR ONE(MSG)=NO
THEN WRITE-LOG(GLOBAL-ABORT)
DECISION=ABORT
ELSE WRITE-LOG(GLOBAL-COMMIT)
DECISION=COMMIT
SEND(**RM$_i$**,DECISION)

RM$_i$: RECEIVE(**TM**,DECISION)
IF COMMIT THEN WRITE-LOG(COMMIT)
ELSE WRITE-LOG(ABORT)
SEND (TM,ACK)

TM: RECEIVE(**RM$_i$**,ACK)
WRITE-LOG(COMPLETE)

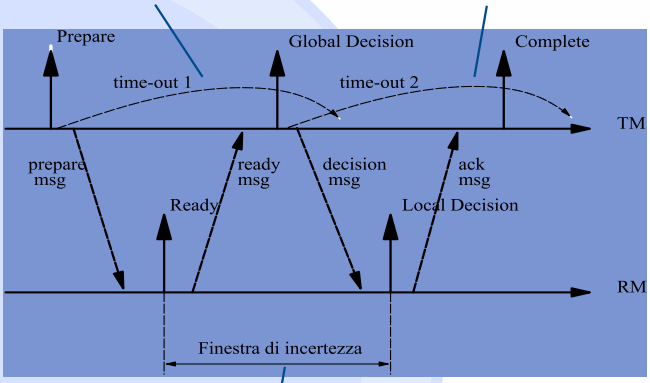
Diagramma 2PC



Protocollo 2PC

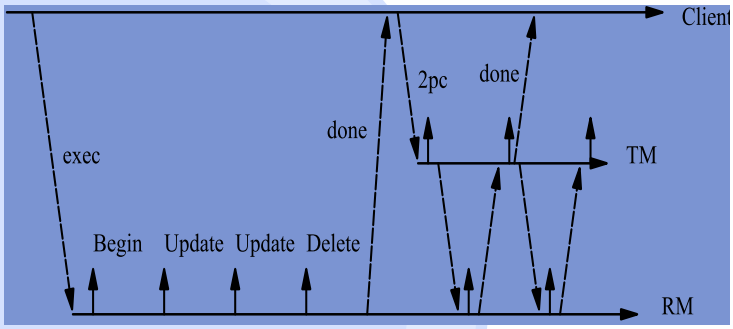
Abort globale

Ripetizione trasmissioni



Dato il Ready il RM perde autonomia
Deve aspettare decisione TM

Protocollo 2PC



Complessità del protocollo

- Il protocollo di commit a due fasi deve essere in grado di gestire tutti i guasti
  - Caduta del coordinatore
  - Caduta di uno o più partecipanti
  - Perdite di messaggi

Recovery dei partecipanti

- Viene svolto durante la loro ripresa a caldo, per ogni transazione dipende dall'ultimo record nel log:
  - Se è una azione generica oppure un "abort" tutte le azioni sono disfatte; se è un "commit", le azioni vengono rifatte; la recovery non dipende dal protocollo.
  - Se è un "ready", il guasto è accaduto durante il commit a due fasi; il partecipante è in dubbio circa il suo esito.
- Durante il protocollo di ripresa a caldo, si collezionano tutte le transazioni in dubbio; di esse si chiede l'esito ai rispettivi TM (richiesta di recovery remota).

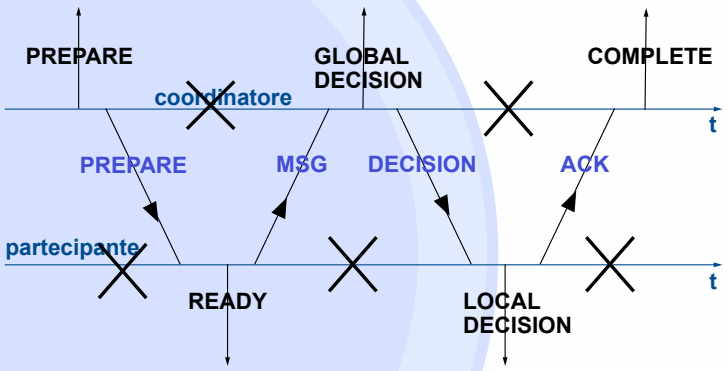
Recovery del coordinatore

- Quando l'ultimo record nel log è un "prepare", il guasto del TM può essere la causa di un blocco di qualche RM. Il TM ha due opzioni:
  - Scrivere "global abort" sul log, e ripetere la seconda fase del protocollo di commit a due fasi
  - Ripetere la prima fase, provando a raggiungere un commit globale
- Quando l'ultimo record è la una "decisione globale", alcuni RMs potrebbero essere bloccati. Il TM deve ripetere la seconda fase del protocollo.

Perdita di messaggi e partizione di rete

- La perdita dei messaggi di "prepare" o "ready" sono indistinguibili; in entrambi i casi scatta il time-out sul TM e viene deciso un "global abort".
- La perdita dei messaggi "decision" or "ack" sono pure indistinguibili. in entrambi i casi scatta il time-out sul TM e viene ripetuta la seconda fase.
- Un partizionamento della rete non causa problemi: si perviene a "global commit" solo se RM e TM appartengono alla stessa partizione.

Recovery del protocollo 2PC



Protocollo di "abort presunto"

- E' una ottimizzazione, usata da tutti i sistemi commerciali:
  - Se un TM riceve una richiesta di "remote recovery" da una transazione in dubbio che non gli è nota, risponde per default che la transazione ha fatto un "global abort"
- Come conseguenza, se vengono persi "prepare" e "global abort" si ottiene comunque un corportamento corretto => non è necessario scriverli in modo sincrono sul log.
- Inoltre, il record "complete" non può essere omesso.
- In conclusione, gli unici record che devono essere scritti in modo sincrono sono: ready, global commit e commit locale.

Ottimizzazione "read-only"

- Quando un RM ha svolto solo operazioni di lettura,
  - Risponde read-only al messaggio di prepare message e esce dal protocollo.
  - Il TM ignora tutti gli RM "read-only" nella seconda fase del protocollo.

Blocking, incertezza, protocolli di recovery

- Un RM dopo essersi dichiarato “ready” perde la sua autonomia e attende la decisione del TM. Un guasto del TM lascia l’RM in uno stato di incertezza, in cui tutte le risorse acquisite con lock sono bloccate.
- L’intervallo tra la scrittura dei record ready e commit o abort è chiamato **finestra di incertezza**. Il protocollo è progettato per minimizzare la sua durata.
- I protocolly di recovery sono svolti dai TM o RM dopo i guasti; ristabiliscono uno stato finale corretto che dipende dalla decisione globale del TM