Architetture distribuite per Basi di Dati

Carlo Combi

Dipartimento di Informatica

Universita' degli Studi di Verona

Architetture distribuite per Basi di Dati

- Introduzione
- Architettura client-server (richiami)
- Basi di dati distribuite
- Tecnologia delle basi di dati distribuite
- Protocollo di commit a due fasi
- Interoperabilita’
- Parallelismo
- Data warehouse
- Basi di dati replicate

Il paradigma client-server

- Client: richiesta di servizi
- Server: offerta di servizi
- Interfaccia di servizi, messi a disposizione dal server

◊ Client: ruolo attivo

◊ Server: ruolo reattivo

⇒**richieste di un client ad un solo server**

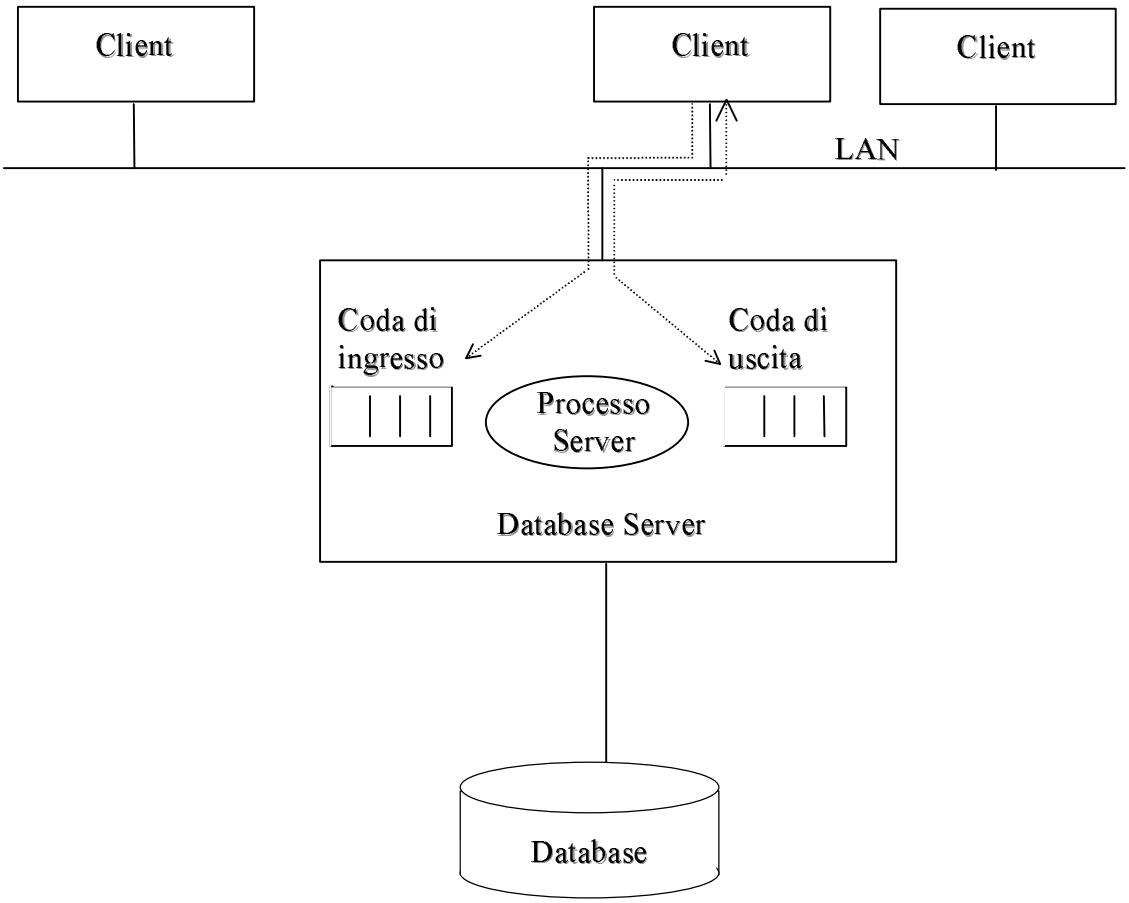
Il paradigma client-server e le basi di dati

- Ruoli ben caratterizzati di client e server nelle basi di dati
  - client: software applicativo
  - server: sistema di basi di dati che supporta piu' applicazioni
- Utilizzo di macchine diverse per client e server
  - client: personal computer
  - server: sistema dimensionato rispetto al carico di lavoro complessivo (carico transazionale)
- SQL: interfaccia di servizi
  - da client a server: interrogazione
  - da server a client: risultati

Il paradigma client-server e le basi di dati

- Interrogazioni compilate staticamente (compile and store)
  - interrogazioni sottomesse una solo volta e richiamate molte volte
  - chiamate a procedure e/o servizi remoti
- Interrogazioni con SQL dinamico (compile and go)
  - invio dell'interrogazione sotto forma di stringhe di caratteri
- Interrogazioni parametriche
  - assegnamento di alcuni parametri e poi esecuzione di una interrogazione o procedura

Architettura client-server



- Server multi-threaded: un unico processo opera per conto di differenti transazioni
- Dispatcher: distribuisce le richieste ai server e restituisce le risposte ai client (gestione delle code)

Basi di dati distribuite

- ***Una transazione coinvolge piu' server***

- Gestione distribuita dei dati vs. gestione centralizzata

- Sistemi distribuiti vs. sistemi centralizzati

  - complessita' strutturale

  - flessibilita', modularita' e resistenza ai guasti

Basi di dati distribuite

- Base di dati
  - *omogenea*: tutti i server usano lo stesso DBMS
  - *eterogenea*: i server utilizzano DBMS diversi
- Rete
  - locale (LAN)
  - geografica (WAN)

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

Basi di dati distribuite: frammentazione e allocazione dei dati

- Frammentazione orizzontale
  - R_i e' un insieme di tuple con lo stesso schema di R
  - ogni R_i e' (logicamente) il risultato di una selezione su R
- Frammentazione verticale
  - lo schema di R_i e' sottoinsieme dello schema di R
  - ogni R_i e' (logicamente) il risultato di una proiezione su R

Basi di dati distribuite: frammentazione e allocazione dei dati

♦ Correttezza della frammentazione

- *completezza*: ogni dato in R deve essere presente in un qualche suo frammento R_i
- *ricostruibilita'*: la relazione deve essere interamente ricostruibile a partire dai suoi frammenti

Basi di dati distribuite: Esempi di frammentazione

♦ Frammentazione orizzontale

Impiegato(Empnum, Nome, Dip, Sal, Tax)

Impiegato1 = $\sigma_{Empnum \leq 3}$Impiegato

Impiegato2 = $\sigma_{Empnum > 3}$Impiegato

♦ Ricostruzione della relazione

Impiegato = Impiegato1 $\cup$ Impiegato2

Basi di dati distribuite: Esempi di frammentazione

♦ Frammentazione orizzontale

Impiegato(Empnum, Nome, Dip, Sal, Tax)

Empnum	Nome	Dip	Sal	Tax
1	Roberto	Produzione	3.7 M	1.2
2	Giovanni	Amministrazione	3.5 M	1.1
3	Anna	Produzione	5.3 M	2.1
4	Carlo	Marketing	3.5 M	1.1
5	Alfredo	Amministrazione	3.7 M	1.2
6	Paolo	Direzione	8.3 M	3.6
7	Giorgio	Marketing	4.2 M	1.4

Impiegato1(Empnum, Nome, Dip, Sal, Tax)

Empnum	Nome	Dip	Sal	Tax
1	Roberto	Produzione	3.7 M	1.2
2	Giovanni	Amministrazione	3.5 M	1.1
3	Anna	Produzione	5.3 M	2.1

Impiegato2(Empnum, Nome, Dip, Sal, Tax)

Empnum	Nome	Dip	Sal	Tax
4	Carlo	Marketing	3.5 M	1.1
5	Alfredo	Amministrazione	3.7 M	1.2
6	Paolo	Direzione	8.3 M	3.6
7	Giorgio	Marketing	4.2 M	1.4

Basi di dati distribuite: Esempi di frammentazione

♦ Frammentazione verticale

Impiegato(Empnum, Nome, Dip, Sal, Tax)

Impiegato1 = $\pi_{\text{Empnum, Nome}}$(Impiegato)

Impiegato2 = $\pi_{\text{Empnum, Dip, Sal, Tax}}$(Impiegato)

♦ Ricostruzione della relazione

Impiegato = Impiegato1 $\triangleright\!\triangleleft$ Impiegato2

Basi di dati distribuite: Esempi di frammentazione

♦ Frammentazione verticale

Impiegato(Empnum, Nome, Dip, Sal, Tax)

Empnum	Nome	Dip	Sal	Tax
1	Roberto	Produzione	3.7 M	1.2
2	Giovanni	Amministrazione	3.5 M	1.1
3	Anna	Produzione	5.3 M	2.1
4	Carlo	Marketing	3.5 M	1.1
5	Alfredo	Amministrazione	3.7 M	1.2
6	Paolo	Direzione	8.3 M	3.6
7	Giorgio	Marketing	4.2 M	1.4

Impiegato1(Empnum, Nome)

Empnum	Nome
1	Roberto
2	Giovanni
3	Anna
4	Carlo
5	Alfredo
6	Paolo
7	Giorgio

Impiegato2(Empnum, Dip, Sal, Tax)

Empnum	Dip	Sal	Tax
1	Produzione	3.7 M	1.2
2	Amministrazione	3.5 M	1.1
3	Produzione	5.3 M	2.1
4	Marketing	3.5 M	1.1
5	Amministrazione	3.7 M	1.2
6	Direzione	8.3 M	3.6
7	Marketing	4.2 M	1.4

Basi di dati distribuite: frammentazione e allocazione dei dati

- Ogni frammento R_i e' implementato attraverso un file fisico su uno specifico server (allocazione)
- Schema di allocazione: mapping dai frammenti (o dalle relazioni) ai server che li memorizzano.
  - mapping non ridondante
  - mapping ridondante

Basi di dati distribuite: frammentazione e allocazione dei dati

♦ Livelli di trasparenza

- trasparenza di frammentazione
- trasparenza di allocazione
- trasparenza di linguaggio
- assenza di trasparenza

Basi di dati distribuite: frammentazione e allocazione dei dati

Fornitore(Fpnum, Nome, Città)

- Frammentazione

  Fornitore1 = $\sigma_{\text{Città = 'Milano'}}$ Fornitore

  Fornitore2 = $\sigma_{\text{Città = 'Roma'}}$ Fornitore

- Allocazione

  Fornitore1@ditta.milano.it

  Fornitore2@ditta.roma1.it

  Fornitore2@ditta.roma2.it

- Applicazione: dato un numero di fornitore, viene restituito il nome del fornitore stesso

Basi di dati distribuite: frammentazione e allocazione dei dati

- Trasparenza di frammentazione

```
procedure Query1(:fnum, :nome);
        select Nome into :nome
        from Fornitore
        where Fnum = :fnum;
end procedure;
```

- Trasparenza di allocazione

```
procedure Query2(:fnum, :nome);
        select Nome into :nome
        from Fornitore1
        where Fnum = :fnum;
if :empty then
        select Nome into :nome
        from Fornitore2
        where Fnum = :fnum;
end procedure;
```

Basi di dati distribuite: frammentazione e allocazione dei dati

- Trasparenza di linguaggio

```
procedure Query3(:fnum, :nome);
    select Nome into :nome
    from Fornitore1@ditta.milano.it
    where Fnum = :fnum;
if :empty then
    select Nome into :nome
from Fornitore2@ditta.roma1.it
    where Fnum = :fnum;
end procedure;
```

- Assenza di trasparenza: il programmatore deve indicare esplicitamente frammenti ed allocazioni ed usare dialetti diversi di SQL per ogni DBMS

Basi di dati distribuite: classificazione delle transazioni

- *Richieste remote*: transazioni di sola lettura ad un solo DBMS remoto
- *Transazioni remote*: transazioni (con comandi SQL di qualunque genere - select, insert, delete, update) ad un solo DBMS remoto
-
- *Transazioni distribuite*: transazioni rivolte a piu' DBMS, dove ogni comando SQL fa riferimento ai dati di un solo DBMS
- *Richieste distribuite*: transazioni arbitrarie, dove ogni query puo' far riferimento a dati su un qualunque DBMS

Basi di dati distribuite: classificazione delle transazioni

♦ Esempio di transazione distribuita, a livello di trasparenza di allocazione

ContoCorrente(CCnum, Nome, Saldo)

- frammentazione

ContoCorrente1 = $\sigma_{CCnum \leq 10000}$ ContoCorrente

ContoCorrente2 = $\sigma_{CCnum > 10000}$ ContoCorrente

```
begin transaction

    update ContoCorrente1
    set Saldo = Saldo - 100.000
    where Ccnum = 3154;

    update ContoCorrente2
    set Saldo = Saldo + 100.000
    where Ccnum = 14878;
    commit work;

end transaction
```

Tecnologia delle basi di dati distribuite

- La *consistenza* delle transazioni non dipende dalla distribuzione dei dati (limiti dei DBMS attuali)

- La *persistenza* non dipende dalla distribuzione dei dati

- Vanno considerati, invece:

  ⇒ottimizzazione delle interrogazioni

  ⇒controllo di concorrenza

  ⇒controllo di affidabilita'

Tecnologia delle basi di dati distribuite

- Ottimizzazione di interrogazioni distribuite
  - solo per richieste distribuite
  - ottimizzazione globale
  - ordine delle operazioni
  - metodo di esecuzione delle operazioni
  - strategia di esecuzione per operazioni con operandi su nodi differenti (trasmissione ed allocazione dei risultati)
  - $C_{tot} = C_{I/O} \times n_{I/O} + C_{cpu} \times n_{cpu} + C_{tr} \times n_{tr}$

Tecnologia delle basi di dati distribuite

♦ Controllo di concorrenza

- transazione t_i; sottotransazioni t_{ij} su diversi nodi j

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 serializzabilita' locale presso gli scheduler non e' garanzia sufficiente per la serializzabilita'**

S_1 $r_{11}(x)w_{11}(x)r_{21}(x)w_{21}(x)$

S_2 $r_{22}(y)w_{22}(y)r_{12}(y)w_{12}(y)$

- sul nodo 1, t_1 precede t_2 ed e' in conflitto con t_2
- sul nodo 2, t_2 precede t_1 ed e' in conflitto con t_1

Tecnologia delle basi di dati distribuite

♦ Serializzabilita' globale

- deve esistere un unico schedule seriale S, che coinvolga tutte le transazioni del sistema, equivalente a tutti gli schedule locali S_i
- per ogni nodo i, la proiezione $S[i]$ di S con le sole operazioni svolte su i, deve essere equivalente a S_i

Tecnologia delle basi di dati distribuite

♦ Serializzabilita' globale

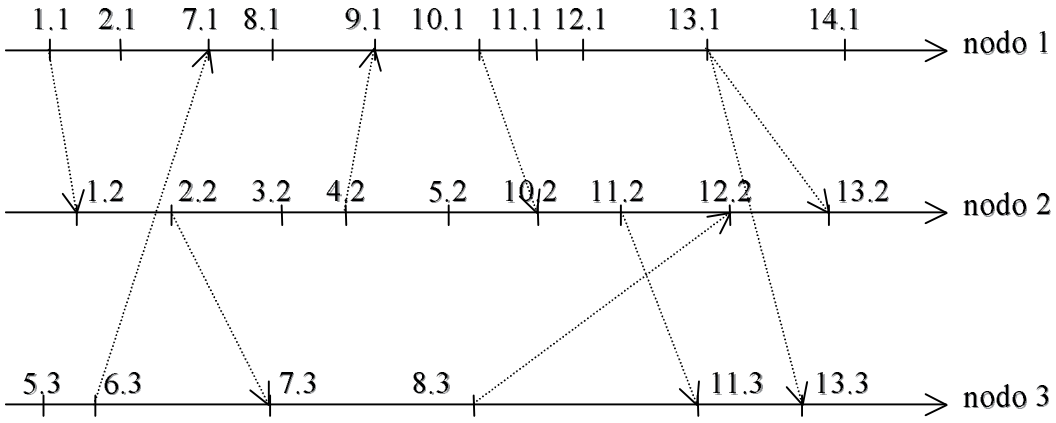
- se ciascun scheduler della base di dati distribuita usa su ciascun nodo il metodo di locking a due fasi e svolge l'azione di commit in modo atomico in un istante in cui le sotto-transazioni ai vari nodi detengono tutte le risorse, gli schedule risultanti sono globalmente serializzabili rispetto ai conflitti

- se un insieme di sotto-transazioni distribuite acquisisce un unico timestamp e lo usa nelle sue richieste a tutti gli scheduler che usano il controllo di concorrenza basato su timestamp, gli schedule risultanti sono globalmente seriali in base all'ordinamento indotto dai timestamp.

Tecnologia delle basi di dati distribuite

♦ Metodo di Lamport per assegnare i timestamp

- i timestamp devono riflettere le relazioni di precedenza fra eventi in un sistema distribuito
- ogni timestamp e' formato da due gruppi di cifre: le meno significative identificano un nodo, le piu' significative identificano gli eventi su ciascun nodo

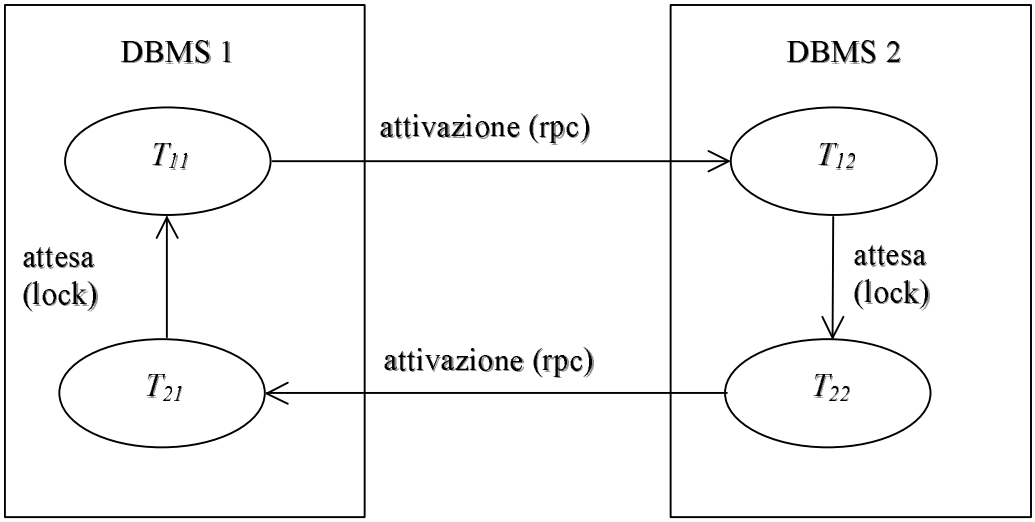


Tecnologia delle basi di dati distribuite

- Rilevazione distribuita dei deadlock
  - attesa circolare che coinvolge due o piu' nodi
  - impiego dei time-out
- Un algoritmo di rilevazione dei deadlock
  - una transazione e' composta di sotto-transazioni
  - due sotto-trasanzioni della stessa transazione in attesa su DBMS distinti (una attende la fine dell'altra)
  - due sotto-transazioni di diverse transazioni sono in attesa sullo stesso DBMS (una blocca i dati a cui vuole accedere l'altra)

Tecnologia delle basi di dati distribuite

♦ Esempio di deadlock distribuito



- T_{11} attende T_{12}, attivata con chiamata a procedura remota
- T_{12} attende una risorsa controllata da T_{22}
- T_{22} attende T_{21}, attivata con chiamata a procedura remota
- T_{21} attende una risorsa controllata da T_{11}

Tecnologia delle basi di dati distribuite

♦ Un algoritmo di rilevazione distribuita dei deadlock

- Condizioni di attesa:

al DBMS 1: $EXT \rightarrow T_{21} \rightarrow T_{11} \rightarrow EXT$

al DBMS 2: $EXT \rightarrow T_{12} \rightarrow T_{22} \rightarrow EXT$

- Sequenza di attesa: $EXT \rightarrow T_i \rightarrow T_j \rightarrow EXT$

♦ L'algoritmo e' distribuito e viene attivato dai vari DBMS

- analisi della sequenza di attesa locale
- comunicazione ad altre istanze dell'algoritmo delle sequenze di attesa

Tecnologia delle basi di dati distribuite

♦ Un algoritmo di rilevazione distribuita dei deadlock

- ricevimento delle sequenze di attesa dagli altri DBMS
- composizione di tali sequenze nel grafo di attesa locale (ogni nodo una transazione)
- ricerca locale di deadlock, con abort di una delle transazioni coinvolte nel deadlock
- trasmissione “in avanti” delle sequenze di attesa

Tecnologia delle basi di dati distribuite

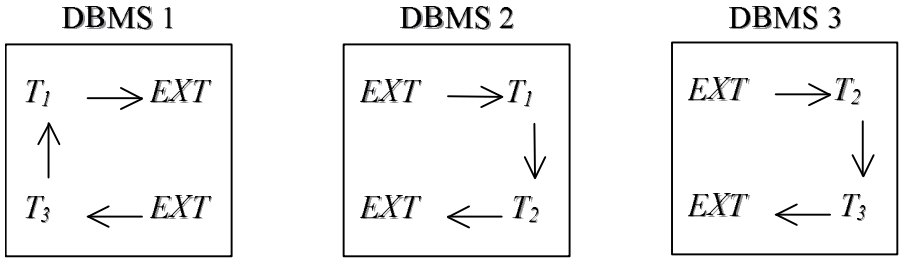
♦ Un esempio di funzionamento dell'algoritmo (1)

- Condizione iniziale

S1: EXT → T3 → T1 → EXT

S2: EXT → T1 → T2 → EXT

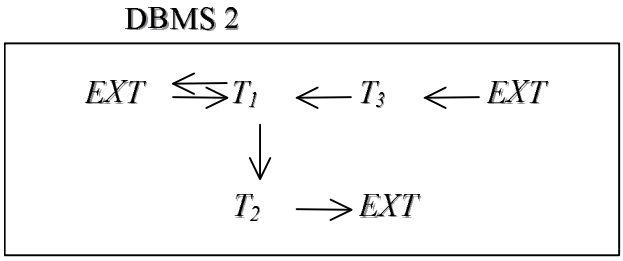
S3: EXT → T2 → T3 → EXT



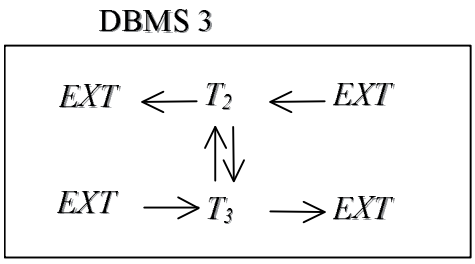
Tecnologia delle basi di dati distribuite

♦ Un esempio di funzionamento dell'algoritmo (2)

- il DBMS 1 comunica la sequenza di attesa al DBMS 2



- il DBMS 2 comunica la sequenza di attesa al DBMS 3



- Il deadlock e' individuato ed una delle due transazioni e' scelta per il deadlock

Tecnologia delle basi di dati distribuite

♦ Atomicita' di transazioni distribuite

- tutti i nodi che partecipano alla transazione devono pervenire alla stessa decisione (commit o abort)
- *protocolli di commit*
- molteplici cause di guasto: caduta di un nodo, perdita di un messaggio, partizionamento della rete
- uso di messaggi di *ack*

Protocollo di commit a due fasi

- Analogia con il matrimonio
  - sposi = server (resource manager: RM)
  - celebrante = processo coordinatore (transaction manager: TM)
  - numero arbitrario di partecipanti al matrimonio

Protocollo di commit a due fasi

- Nuovi record nel log
  - sia TM che RM sono dotati di propri *log*
- Il TM scrive:
  - record di *prepare* con l'identita' dei processi RM (partecipazioni)
  - record di *global commit* o di *global abort*
  - record di *complete*
- L'RM scrive:
  - *begin*, *insert*, *delete*, *update*
  - record di *ready*: disponibilita' di partecipare al protocollo di commit a due fasi

Protocollo di commit a due fasi

♦ Protocollo in assenza di guasti - I fase

- il TM scrive il record di *prepare* ed invia un messaggio di *prepare* per informare dell'inizio del protocollo
- gli RM, che sono in uno stato affidabile, attendono il messaggio di *prepare*; non appena ricevono il messaggio, scrivono il record di *ready* e mandano il messaggio di *ready* al TM (oppure *not-ready* nel caso di guasto di transazione)
- il TM colleziona i messaggi di risposta: se tutti sono positivi, scrive sul suo log un record di *global commit*; in caso contrario scrive un record di *global abort*

Protocollo di commit a due fasi

♦ Protocollo in assenza di guasti - II fase

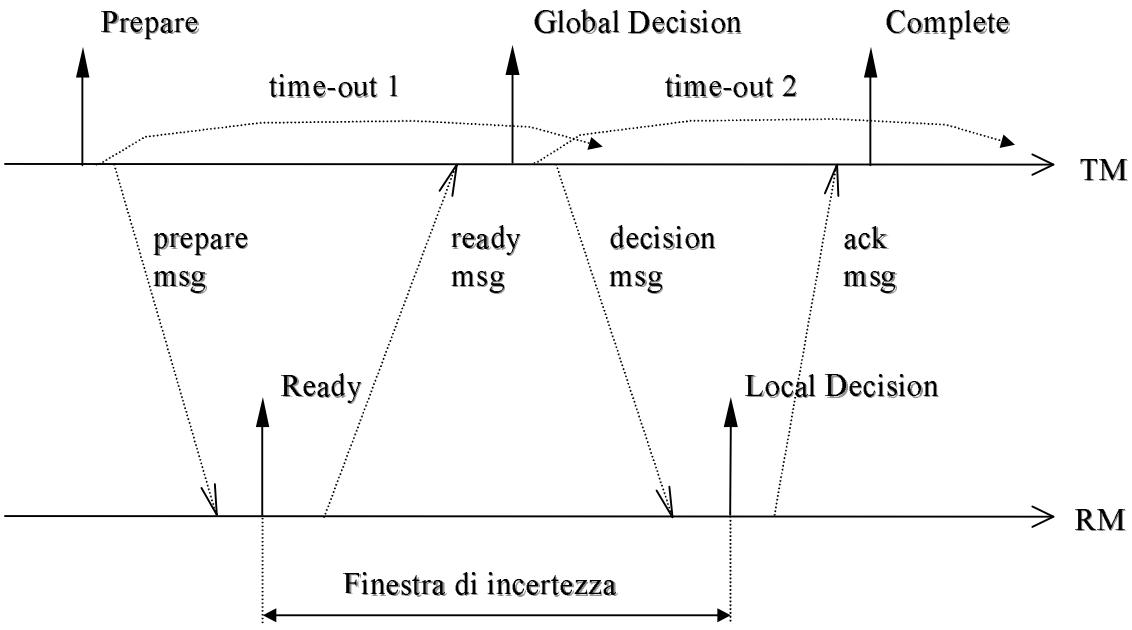
- il TM invia la sua decisione globale agli RM ed imposta un time-out per la ricezione dei messaggi di risposta dagli RM

- gli RM, che sono in uno stato di ready, attendono il messaggio di decisione; non appena ricevono il messaggio, scrivono il record di *commit* o *abort* (locale) e mandano il messaggio di *acknowledgement* al TM

- il TM colleziona i messaggi di ack: se tutti i messaggi arrivano, scrive sul suo log un record di *complete*; in caso contrario reimposta il time-out e ripete la trasmissione, finche' tutti gli RM non hanno risposto

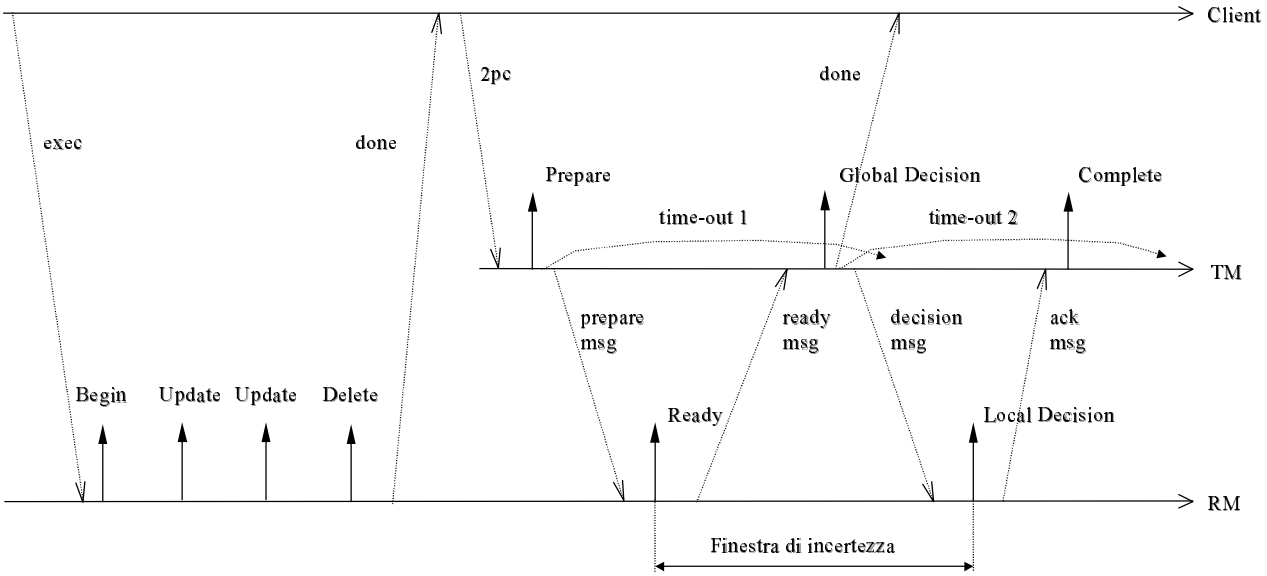
Protocollo di commit a due fasi

- Assenza di comunicazione tra TM e RM nella prima fase: abort globale
- Assenza di comunicazione tra TM e RM nella seconda fase: ripetizione della trasmissione



Protocollo di commit a due fasi

♦ Interazione tra client, server e TM



♦ Il protocollo di commit e' rapido: TM e RM scrivono nel file di log e mandano messaggi

- minimizzazione della finestra di incertezza

Protocollo di commit a due fasi

- Protocolli di ripristino
  - Caduta di un partecipante
  - Caduta del coordinatore
  - Perdita di messaggi e partizionamenti delle rete

Protocollo di commit a due fasi

♦ Caduta di un partecipante

- ripresa a caldo:

  ⇒se l'ultimo record nel log e' relativo ad una azione o ad un *abort*, le azioni vanno *disfatte*

  ⇒se l'ultimo record e' un *commit*, le azioni vanno *rifatte*

- caso critico: l'ultimo record nel log e' di *ready*

  ⇒per le transazioni in dubbio, si richiede l'esito finale della transazione

Protocollo di commit a due fasi

♦ Caduta del coordinatore

- se l'ultimo record nel log e' un *prepare*, alcuni RM possono essere in situazione di blocco

  ⇒il TM decide un *global abort*

  ⇒il TM riprova, per decidere un *global commit*

- se l'ultimo record nel log e' un *global commit* o un *global abort*, alcuni RM possono essere in situazione di blocco

  ⇒il TM ripete la seconda fase del protocollo

- se l'ultimo record nel log e' un *complete*, la caduta del coordinatore non ha effetto sulla transazione

Protocollo di commit a due fasi

♦ Perdita di messaggi e partizionamenti delle rete

- perdita di un *prepare* o del successivo *ready*

  ⇒time-out sulla prima fase e *global abort*

- perdita di un *global abort*/*global commit* o del successivo *ack*

  ⇒time-out sulla seconda fase e ripetizione

- partizionamento della rete: una transazione ha successo solo se, durante le fase critiche del protocollo, il TM e tutti gli RM appartengono alla stessa partizione

Protocollo di commit a due fasi

♦ Ottimizzazione del commit a due fasi

- assunzione di scritture nel log *sincrone* (tramite una *force*)
- scelta per default dell'esito di una transazione, in assenza di informazione circa alcuni partecipanti

  ⇒protocollo di abort presunto

  ⇒protocollo di commit presunto

Protocollo di commit a due fasi

♦ Protocollo di abort presunto

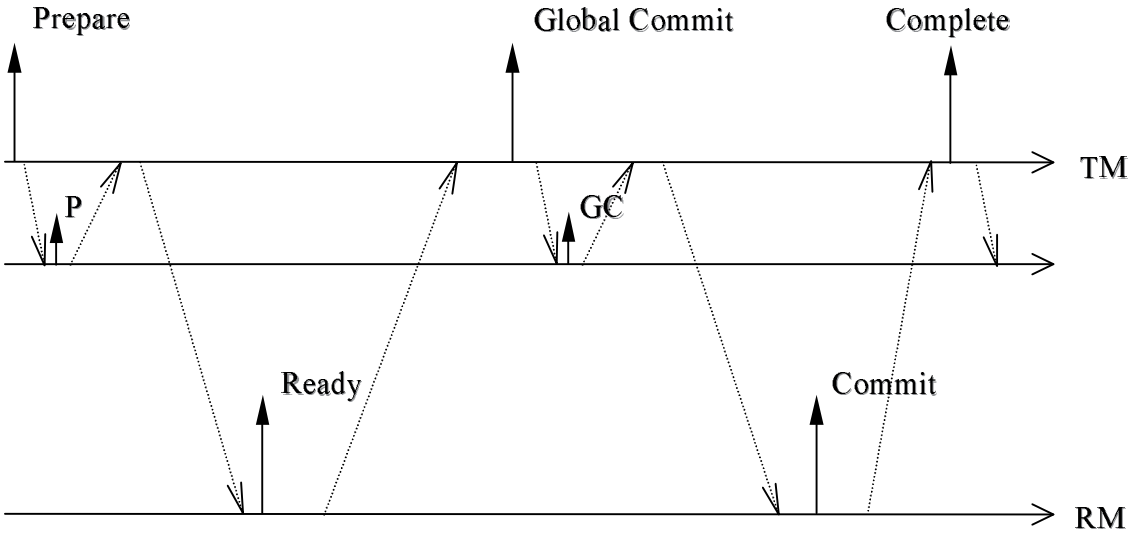
- ad ogni richiesta di *remote recovery* da parte di un partecipante in dubbio, sulla cui transazione il TM non abbia informazione, viene restituita la decisione di *abort*
- i record di *prepare* e *global abort* non sono piu' critici (si evita il force), cosi' come il record di *complete*
- devono essere scritti in modo sincrono i record di *ready* e *commit*, nel log dell'RM, ed il record di *global commit* nel log del TM

Protocollo di commit a due fasi

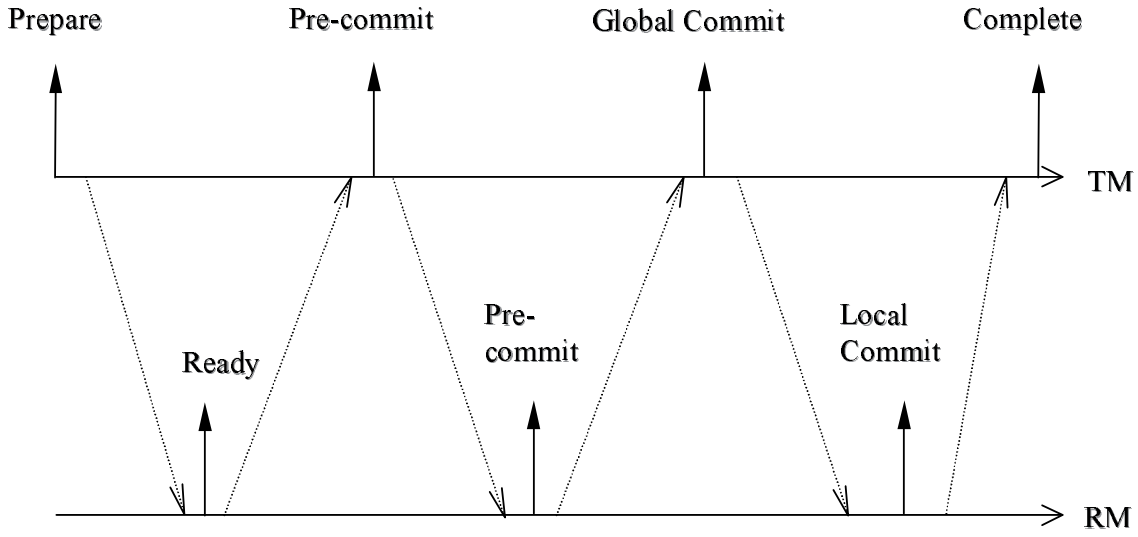
- Ottimizzazione sola lettura
  - se un partecipante scopre di essere di “sola lettura”, al messaggio di *prepare* avvisa il TM, che lo ignorera’ nella seconda fase del protocollo

Altri protocolli di commit

♦ Protocollo di commit a quattro fasi



♦ Protocollo di commit a tre fasi



Interoperabilita’

- Interazione di sistemi di basi di dati (distribuiti) eterogenei
- Standard (numerosi e contrapposti)
  - ODBC
  - X-OPEN

Interoperabilita’

- Open Database Connectivity: ODBC
  - proposta da Microsoft
  - attraverso un’interfaccia ODBC, applicazioni SQL possono accedere a dati remoti
  - SQL “ristretto”

Interoperabilita'

♦ Architettura ODBC

- tra applicazione e server deve esserci un *driver*
- il *driver* maschera le differenze di interazione dovute a DBMS, protocollo di rete e sistema operativo
- (*Sybase*, *Windows/NT*, *Novell*) → driver specifico

Interoperabilita'

♦ Accesso a database remoto tramite ODBC

- l'*applicazione* richiama le funzioni SQL per interrogare ed acquisire i risultati
- il *driver manager* carica i driver necessari all'applicazione (inizializzazione dei driver; fornito da Microsoft)
- i *driver* sono responsabili dell'esecuzione di funzioni ODBC (interrogazione attraverso SQL specifici e bufferizzazione dei risultati)
- la fonte di informazione, *data source*, e' il sistema remoto che esegue quanto richiesto dal client

Interoperabilita’

- Transazioni e ODBC
  - comandi transazionali commit-work e rollback-work rivolti esplicitamente ad un server
  - ODBC non supporta il protocollo di commit a due fasi
  - codici di errore standardizzati
  - possibilita’ di interrogazioni statiche e dinamiche

Interoperabilita'

- Commit standard: X-OPEN DTP (Distributed Transaction Processing)
  - interoperabilita' di computazioni transazionali su DBMS di fornitori differenti
  - architettura composta di un processo client, vari processi RM ed un processo TM
- Il protocollo X-OPEN DTP
  - interfaccia tra client e TM: *TM-interface*
  - interfaccia tra TM ed RM: *XA-interface*

Interoperabilita’

♦ X-OPEN DTP

- RM passivi; il controllo e’ completamente nel TM
- commit a due fasi con ottimizzazione di *abort presunto* e *sola lettura*
- decisioni euristiche, per il controllo dell’operatore sulle transazioni in presenza di guasti

Interoperabilita’

- TM-interface
  - tm_init e tm_exit per iniziare e terminare il dialogo client-TM
  - tm_open, alla quale segue l’apertura di una sessione del TM con i vari RM; tm_term, chiusura della sessione
  - tm_begin, per iniziare una transazione
  - tm_commit, per richiedere un commit globale

Interoperabilita'

♦ XA-interface

- xa_open e xa_close per inizializzare e concludere il dialogo TM-RM
- xa_start e xa_end, per far partire una nuova transazione RM e per completarla
- xa_precom, per richiedere all'RM di svolgere la prima fase del protocollo di commit
- xa_commit e xa_abort, per comunicare la decisione globale relativa alla transazione
- xa_recover, per iniziare una procedura di ripristino dopo una caduta di un processo (TM o RM); xa_forget, per far dimenticare ad un RM transazioni decise in modo euristico

Interoperabilita’

♦ Esempio di interazione tra client, TM e RM

	TM-Interface	XA-Interface
Dialogo client-TM	tm_init()	
Apertura sessione	tm_open()	xa_open()
Inizio transazione	tm_begin()	xa_start()
Fine transazione (2PC)	tm_commit()	xa_precom()
		xa_commit()
		xa_abort()
		xa_end()
Chiusura sessione	tm_term()	xa_close()
Recovery guidata dal TM		xa_recover()
		xa_commit()
		xa_abort()

		xa_forget()

Interoperabilita'

♦ X-OPEN DTP

- Se un RM e' bloccato per la caduta di un TM, l'operatore puo' imporre una decisione euristica (*abort*)
- Procedura di ripristino guidata dal TM che chiama l'RM

  ⇒transazioni in dubbio

  ⇒transazioni decise con un *commit euristico*

  ⇒transazioni decise con un *abort euristico*
- il TM comunica alle transazioni in dubbio il loro esito effettivo, verifica che le decisioni empiriche siano coerenti (correzioni con xa_forget)

Parallelismo

- Parallelismo *inter-query*: diverse interrogazioni in parallelo
  - numerose transazioni semplici
  - OLTP: On Line Transaction Processing
- Parallelismo *intra-query*: parti della stessa interrogazione in parallelo
  - poche interrogazioni complesse, suddivise su piu' processori
  - OLAP: On Line Analytical Processing

Parallelismo

♦ Frammentazione dei dati

ContoCorrente(CCNum, Nome, Saldo)

Movimento(CCNum, Datam Progr, Caus, Amm)

- frammentazione in base ad intervalli predefiniti di numero di conto corrente

- Interrogazione OLTP

```
procedure Query5(:cc-num, :saldo);
        select Saldo into :saldo
        from ContoCorrente
        where CCNum = :cc-num;
end procedure;
```

Parallelismo

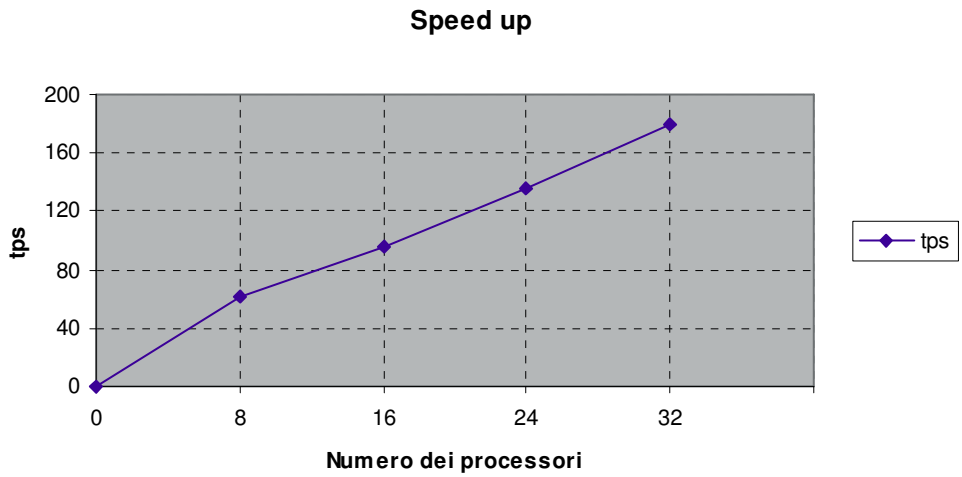
♦ Frammentazione dei dati

- Interrogazione OLAP

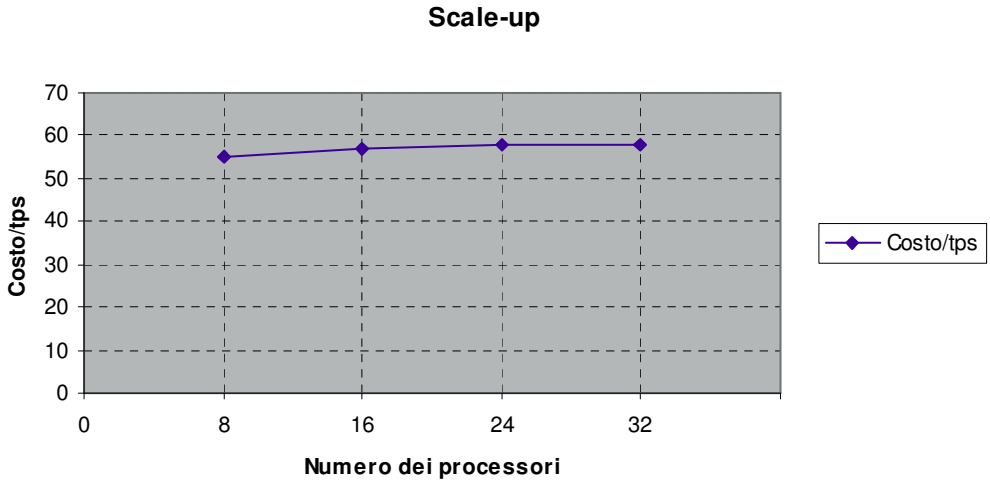
```
procedure Query6();
select Nome, sum(Amm)
from ContoCorrente join Movimento on
        ContoCorrente.CCNum =    Movimento.CCNum
where Data > 1.1.96
group by CCNum, Nome
having sum(abs(Ammontare)) > 100 M;
      end procedure;
```

Parallelismo

♦ Speed-up e Scale-up



♦ Scale-up



Parallelismo

♦ Benchmark delle transazioni

- valutazione delle prestazioni delle architetture di basi di dati

- TPC (Transaction Processing Performance Council)

  ⇒TPC-A, per applicazioni OLTP

  ⇒TPC-B, per applicazioni miste

  ⇒TPC-C per applicazioni OLAP

- Specifiche del benchmark

  ⇒codice della transazione

  ⇒dimensione della base di dati e metodo da usare per generare i dati in modo casuale

  ⇒distribuzione degli arrivi delle transazioni (tps)

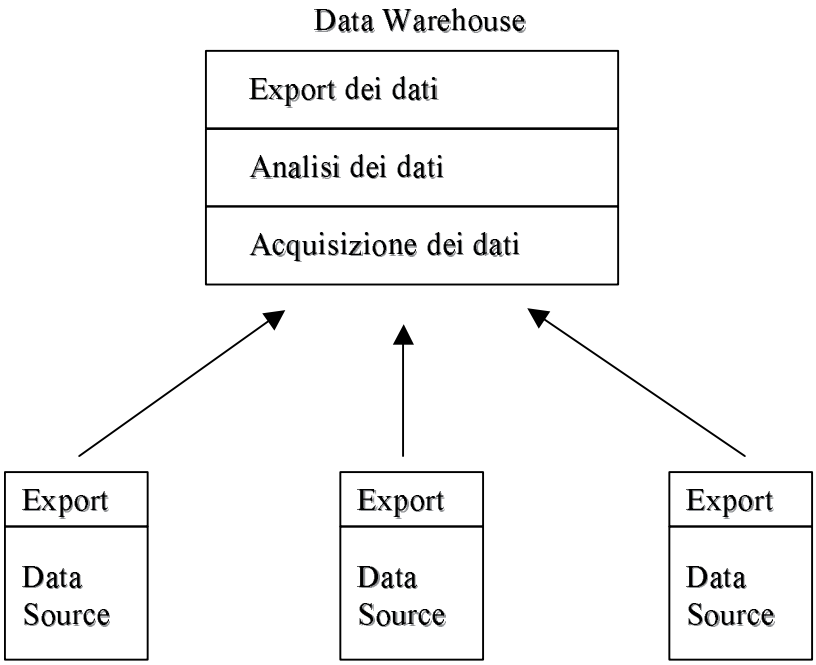
  ⇒modalita' di misurazione e di certificazione della validita' dei benchmark

Data warehouse

- Magazzini di dati per OLAP
  - dati provenienti da sistemi transazionali (OLTP)
  - dati non gestiti da DBMS
  - dati gestiti da DBMS di vecchia concezione (legacy system)
- Dati storico-temporali, spesso non perfettamente aggiornati

Data warehouse

♦ Architettura di una Data warehouse



Data warehouse

♦ Architettura di una Data warehouse

- acquisizione dei dati

  ⇒trasferimento dei dati

  ⇒filtraggio dati inesatti

  ⇒conversione di formato

- accesso ai dati

  ⇒interrogazioni complesse

  ⇒analisi dei dati

- esportazione dei dati

  ⇒organizzazione gerarchica di warehouse

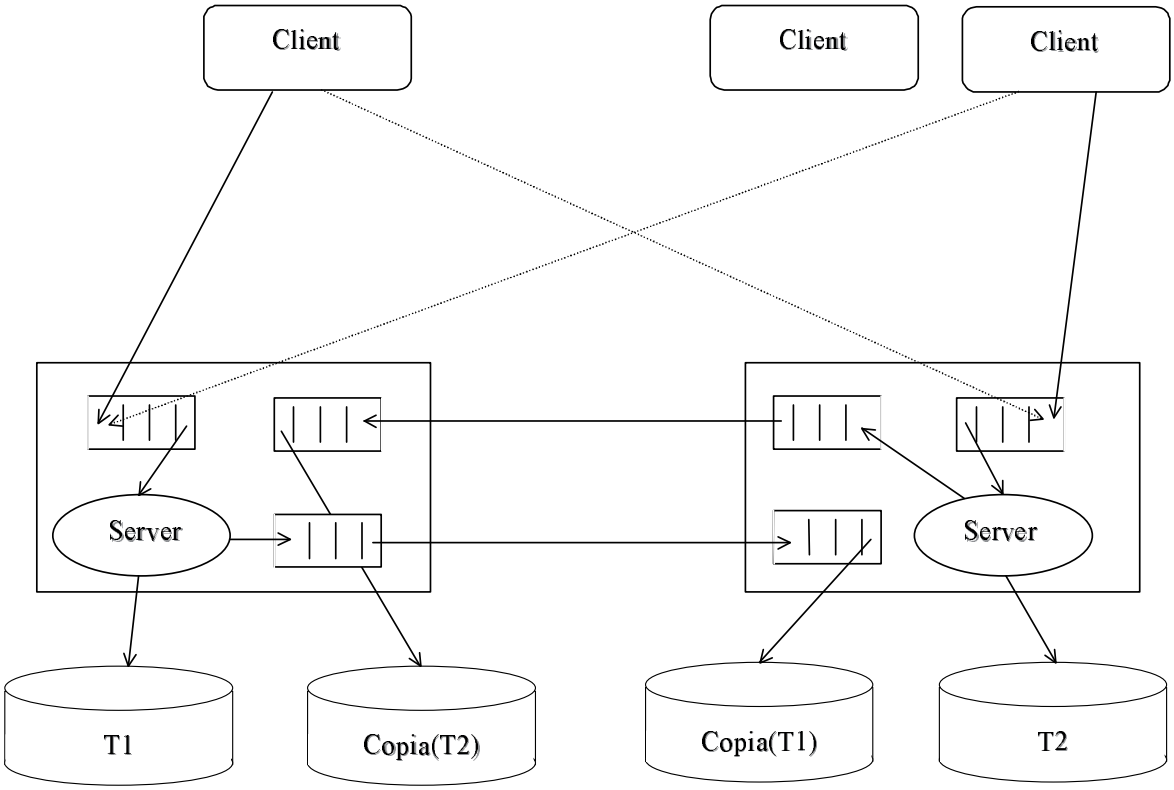
- moduli ausiliari

  ⇒assistenza allo sviluppo

  ⇒descrizione del contenuto della warehouse

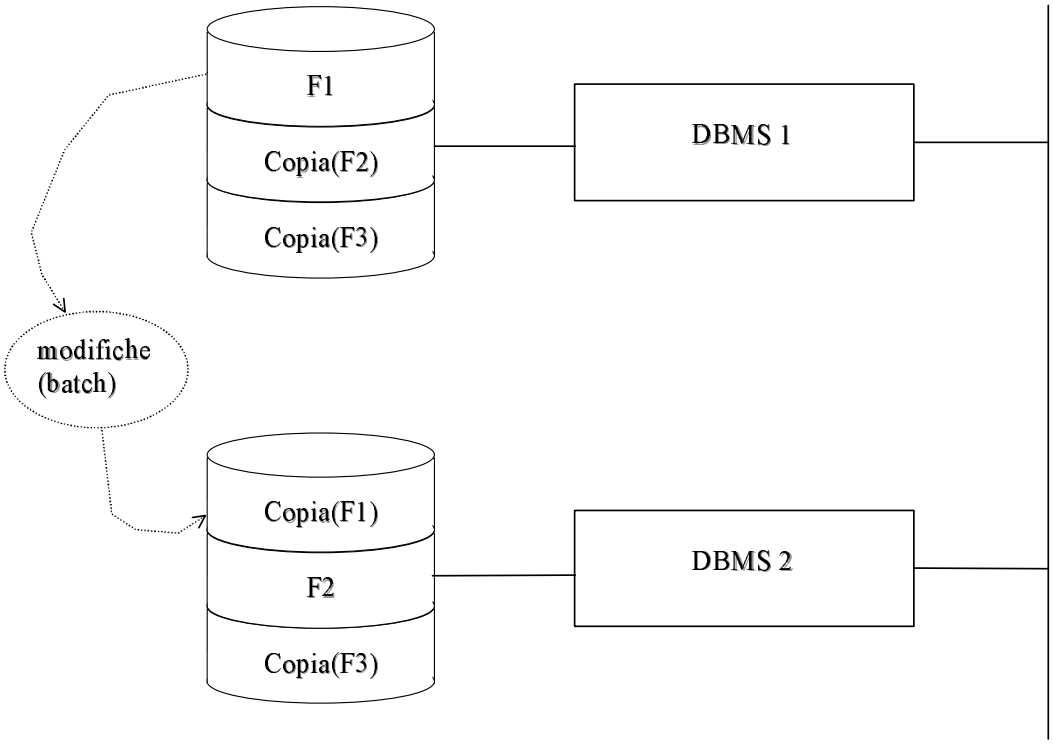
Basi di dati replicate

- Solidita' ai guasti
- Forma sofisticata di backup



Basi di dati replicate

♦ Replicazione, distribuzione e frammentazione



♦ Replicazione simmetrica

- peer-to-peer (sistemi distribuiti mobili)