Architetture Distribuite per Basi di Dati

Carlo Combi** e **Angelo Montanari

Dipartimento di Matematica e Informatica

Università degli Studi di Udine

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
- Interoperabilità
- Parallelismo
- Data warehouse
- Basi di dati replicate

Introduzione - 1

- Architettura Client/Server vs. Basi di dati distribuite
- Basi di dati distribuite vs. Basi di dati parallele
- Tipologie di applicazione: separazione dei DBMS in due componenti fondamentali:
    * **OLTP** (On-Line Transaction Processing), che gestisce in maniera ottimizzata ed affidabile le transazioni, per garantire la modifica dei dati in tempo reale;
    * **OLAP** (On-Line Analytical Processing), che consente l'analisi dei dati. Essa presuppone la possibilità di esportare i dati dalle componenti/sistemi OLTP ai magazzini dei dati (data warehouse).

Introduzione - 2

I server supportano sia funzioni OLTP (per gestire numerose transazioni al secondo) che OLAP (per ottimizzare interrogazioni complesse).

Il parallelismo può essere usato per ottimizzare sia le prestazioni di componenti/sistemi OLTP (parallelismo fra le interrogazioni, o INTER-QUERY) che le prestazioni di componenti/sistemi OLAP (parallelismo all'interno di una singola interrogazione, o INTRA-QUERY).

La **replicazione dei dati**: intesa come la capacità di costruire copie di collezioni di dati, esportabili da un nodo all'altro di un sistema distribuito, per massimizzare:

- disponibilità dei dati;
- affidabilità dei dati.

Introduzione - 3

La necessità di far interagire fra loro sistemi e prodotti diversi richiede che vengano affrontati/risolti i problemi della:

- **portabilità**: possibilità di trasportare programmi da un ambiente ad un altro (proprietà tipica del tempo di compilazione);

- **interoperabilità**: possibilità di far interagire sistemi eterogenei (proprietà tipica del tempo di esecuzione).

Importanza degli **standard**:

- la portabilità dipende dagli standard relativi ai linguaggi (SQL);

- l'interoperabilità dipende dagli standard relativi ai protocolli di accesso ai dati (ODBC, per far dialogare basi di dati eterogenee, X-Open DTP, per far interagire in una stessa transazione diversi DBMS).

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**

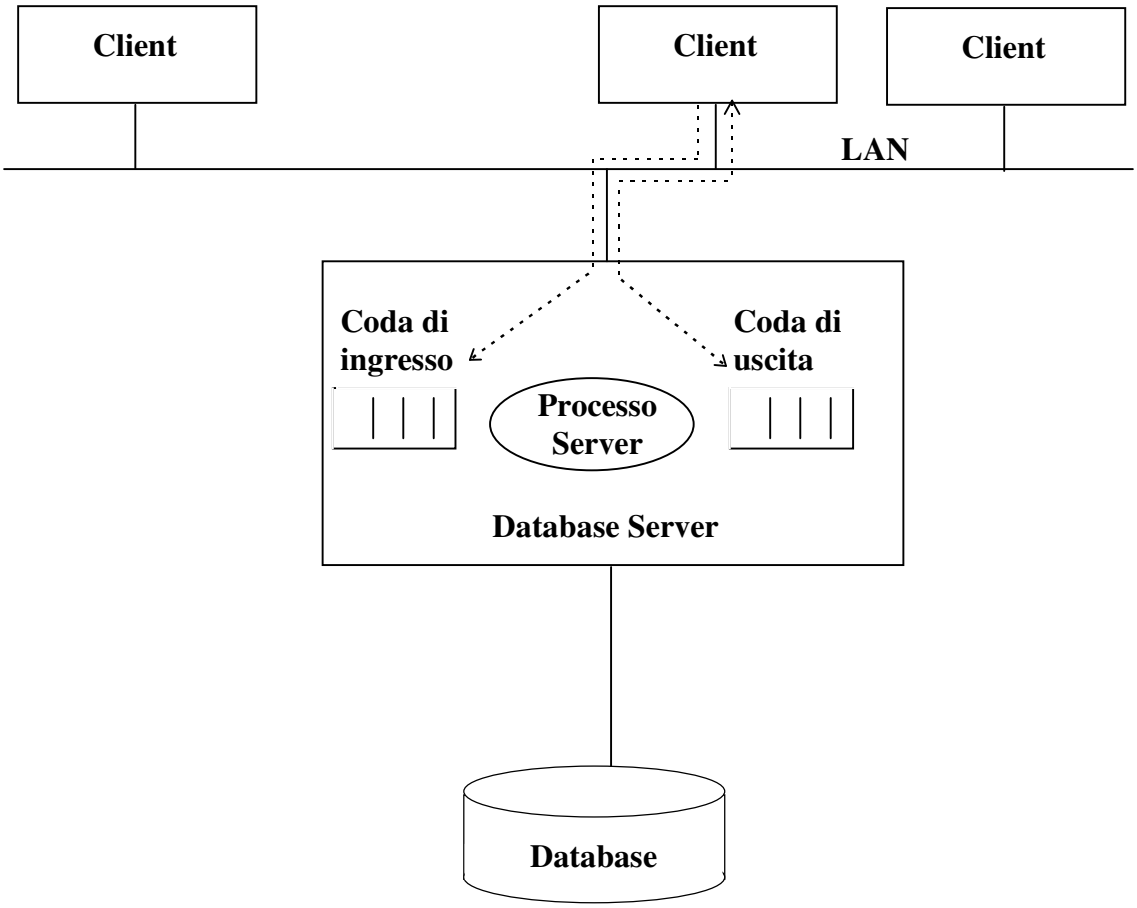
Il paradigma client-server e le basi di dati

- Ruoli ben caratterizzati di client e server nelle basi di dati
  - client: software applicativo
  - server: sistema di basi di dati che supporta più 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 sola 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 più server***

- Gestione distribuita dei dati vs. gestione centralizzata

- Sistemi distribuiti vs. sistemi centralizzati

    * complessità strutturale

    * flessibilità, modularità 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 DBMS	Tipo di rete	
	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
  - Ri è un insieme di tuple con lo stesso schema di R
  - ogni Ri è (logicamente) il risultato di una selezione su R
- Frammentazione verticale
  - lo schema di Ri è sottoinsieme dello schema di R
  - ogni Ri è (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 Ri
  - ∗ *ricostruibilità*: 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 = **Select**$_{\text{Empnum} =< 3}$(Impiegato)

Impiegato2 = **Select**$_{\text{Empnum} > 3}$(Impiegato)

- Ricostruzione della relazione

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

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

- Ricostruzione della relazione

Impiegato = Impiegato1 >< 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 Ri è 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(Fnum, Nome, Città)

Frammentazione

Fornitore1 = **Select**$_{\text{Città=`Milano'}}$Fornitore

Fornitore2 = **Select**$_{\text{Città=`Roma'}}$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.i
    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 più DBMS, dove ogni comando SQL fa riferimento ai dati di un solo DBMS

Richieste distribuite: transazioni arbitrarie, dove ogni query può 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)

con frammentazione

ContoCorrente1=$Select_{CCnum=<10000}$(ContoCorrente)

ContoCorrente2=$Select_{CCnum>10000}$(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 (*isolamento*)

  ⇓ controllo di affidabilità (*atomicità*)

Ottimizzazione di interrogazioni distribuite

Quando e come viene eseguita l'ottimizzazione?

* 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)
* $C_{tot} = C_{I/O} \times n_{I/O} + C_{cpu} \times n_{cpu} + C_{tr} \times n_{tr}$

Proprietà di isolamento - 1

- Controllo della concorrenza

  * transazione *ti*; sottotransazioni *tij* su diversi nodi *j*

  t1 $r11(x)w11(x)r12(y)w12(y)$

  t2 $r22(y)w22(y)r21(x)w21(x)$

  ⇓ **La serializzabilità locale presso gli scheduler non è garanzia sufficiente per la serializzabilità**

  S1 $r11(x)w11(x)r21(x)w21(x)$

  S2 $r22(y)w22(y)r12(y)w12(y)$

  * sul nodo 1, *t1* precede *t2* ed è in conflitto con *t2*

  * sul nodo 2, *t2* precede *t1* ed è in conflitto con *t1*

Proprietà di isolamento - 2

- Serializzabilità globale
  - deve esistere un unico schedule seriale *S*, che deve coinvolgere tutte le transazioni del sistema, equivalente a tutti gli schedule locali *Si*, ossia tale che
  - per ogni nodo i, la proiezione *S*[*i*] di *S*, contenente le sole operazioni svolte su *i*, deve essere equivalente a *Si*

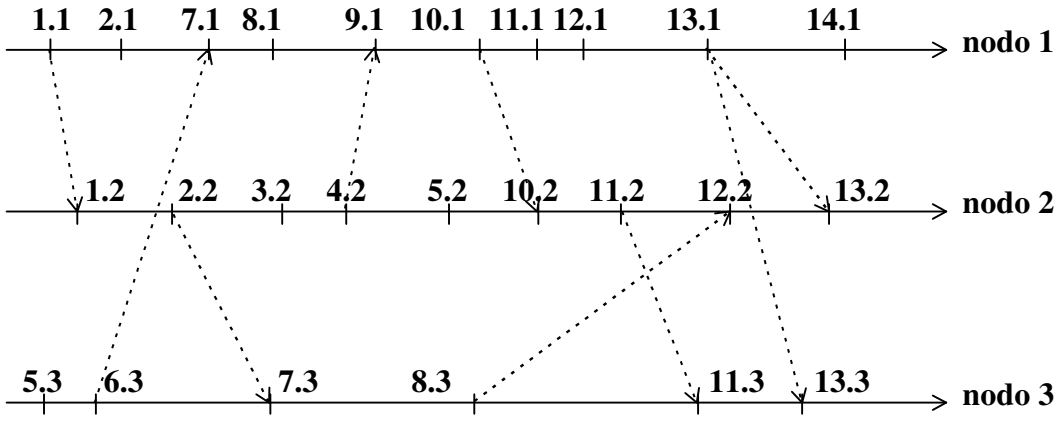
Proprietà di isolamento - 3

- Come garantire la serializzabilità globale?
  - se ciascuno 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.

Il metodo di Lamport - 1

- Metodo di Lamport per assegnare i timestamp:
  - i timestamp devono riflettere le relazioni di precedenza fra eventi in un sistema distribuito
  - ogni timestamp è formato da due gruppi di cifre: le meno significative identificano un nodo, le più significative identificano gli eventi su ciascun nodo
  - ogni qual volta due nodi comunicano scambiandosi un messaggio, i loro timestamp si sincronizzano: l'evento ricezione deve avere un timestamp successivo a quello dell'evento di invio.

Il metodo di Lamport - 2

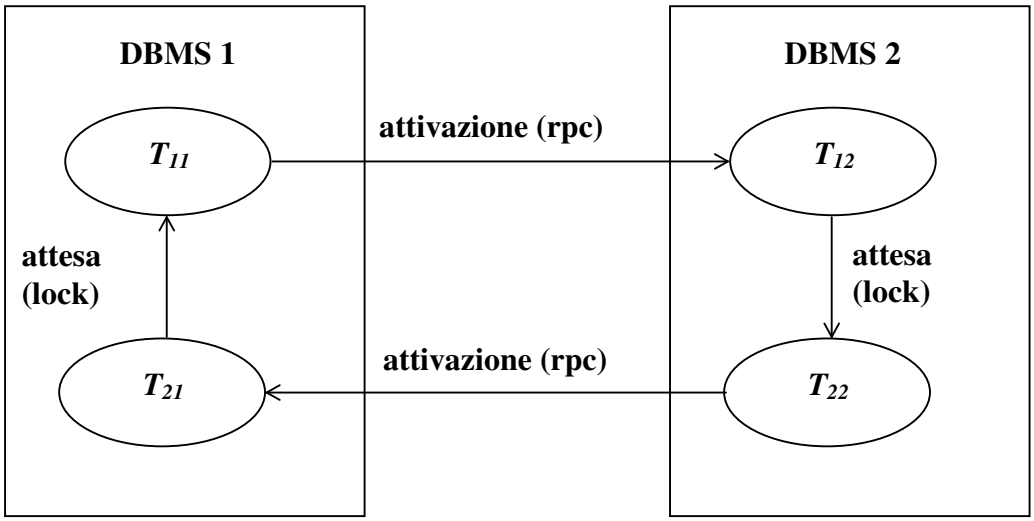


Rilevazione dei deadlock

- Rilevazione distribuita dei deadlock
  - attesa circolare che coinvolge due o più nodi
  - impiego dei time-out
- Un algoritmo di rilevazione dei deadlock
  - una transazione è consiste di sotto-transazioni
  - due sotto-transanzioni 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)

Rilevazione dei deadlock: un esempio

- Esempio di deadlock distribuito



  - *T11* attende *T12*, attivata con chiamata a procedura remota
  - *T12* attende una risorsa controllata da *T22*
  - *T22* attende *T21*, attivata con chiamata a procedura remota
  - *T21* attende una risorsa controllata da *T11*

Un algoritmo per la rilevazione dei deadlock -1

- Un algoritmo di rilevazione distribuita dei deadlock
  - ∗ Condizioni di attesa:

    al DBMS 1: *EXT T21 T11 EXT*

    al DBMS 2: *EXT T12 T22 EXT*
  - ∗ Sequenza di attesa: *EXT Ti Tj EXT*
- L'algoritmo è distribuito e viene attivato dai vari DBMS
  - ∗ analisi della sequenza di attesa locale
  - ∗ comunicazione ad altre istanze dell'algoritmo delle sequenze di attesa

Un algoritmo per la rilevazione dei deadlock -2

- Come opera l'algoritmo di rilevazione distribuita dei deadlock?
  - ricezione delle sequenze di attesa dagli altri DBMS
  - composizione di tali sequenze nel grafo di attesa locale (un nodo per ogni transazione)
  - ricerca locale di deadlock, con abort di una delle transazioni coinvolte nel deadlock
  - trasmissione "in avanti" delle sequenze di attesa

Un esempio -1

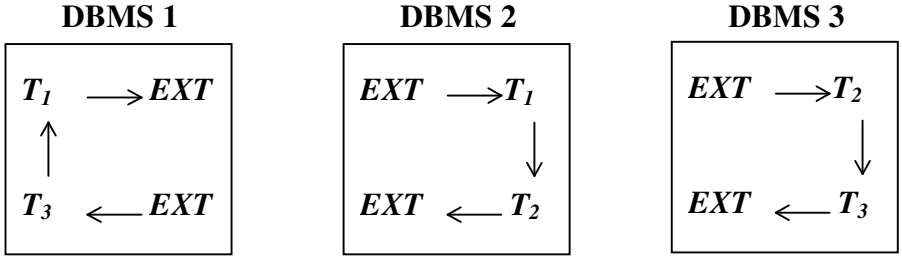
- Un esempio di funzionamento dell'algoritmo (1)

  * Condizione iniziale

  S1: EXT T3 T1 EXT

  S2: EXT T1 T2 EXT

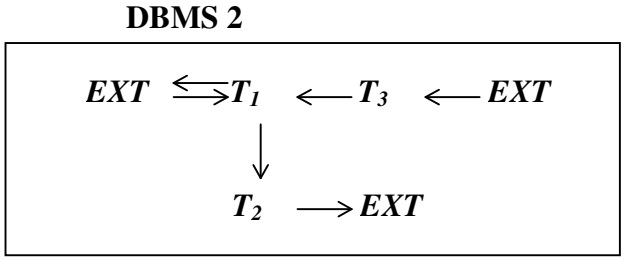
  S3: EXT T2 T3 EXT



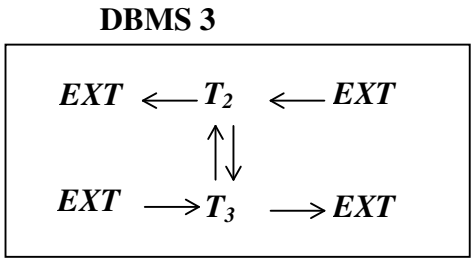
Un esempio -2

- 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 è individuato ed una delle due transazioni è scelta per il deadlock

Proprietà di atomicità

- Atomicità 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 che partecipano ad una transazione (resource managers: RMs)
  - celebrante = processo coordinatore (transaction manager: TM)
  - numero arbitrario di partecipanti al matrimonio

Nuovi record nel log - 1

- Il protocollo di commit a due fasi si svolge tramite un rapido scambio di messaggi fra TM e RM. Per rendere il protocollo resistente ai guasti, TM e RM arricchiscono di alcuni nuovi record i loro file di log (sia TM che RM sono dotati di propri log).

- Il TM scrive:

  * record di *prepare* con l'identità dei processi RM, specificata attraverso un identificatore di nodo e di processo (partecipazioni al matrimonio);

  * record di *global commit* o di *global abort* (l'istante in cui TM scrive nel suo file di log il record di *global commit* o *abort* determina l'esito finale dell'intera transazione);

  * record di *complete*, scritto alla conclusione del protocollo di commit a due fasi).

Nuovi record nel log - 2

- Ogni RM rappresenta una sotto-transazione. Esso scrive, come nel caso centralizzato, i record di:
  - *begin*, *insert*, *delete*, *update*
- La partecipazione al protocollo di commit a due fasi si caratterizza per la presenza di:
  - record di *ready*: disponibilità a partecipare al protocollo di commit a due fasi. In tal record, è riportato anche l'identificatore (nodo e processo) del TM.

Protocollo in assenza di guasti

In questo caso il protocollo consiste in una sequenza di scritture sul log e di scambi di messaggi tra TM e RM.

TM può usare sia meccanismi di *broadcast* per comunicare con diversi RM (stesso messaggio a molti nodi) che meccanismi di *comunicazione seriale*.

Deve essere inoltre in grado di collezionare risposte provenienti da vari nodi. Il protocollo si articola in due fasi.

Protocollo in assenza di guasti: fase 1

- La prima fase del protocollo di commit a due fasi in assenza di guasti si articola nelle seguenti operazioni:

  * il TM scrive il record di *prepare* ed invia un messaggio di *prepare* per informare dell'inizio del protocollo (inoltre, imposta il time out che scatterà nel caso di ritardo eccessivo nella ricezione dei messaggi di risposta)

  * gli RM, che sono in uno stato affidabile, attendono il messaggio di *prepare* e, 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*

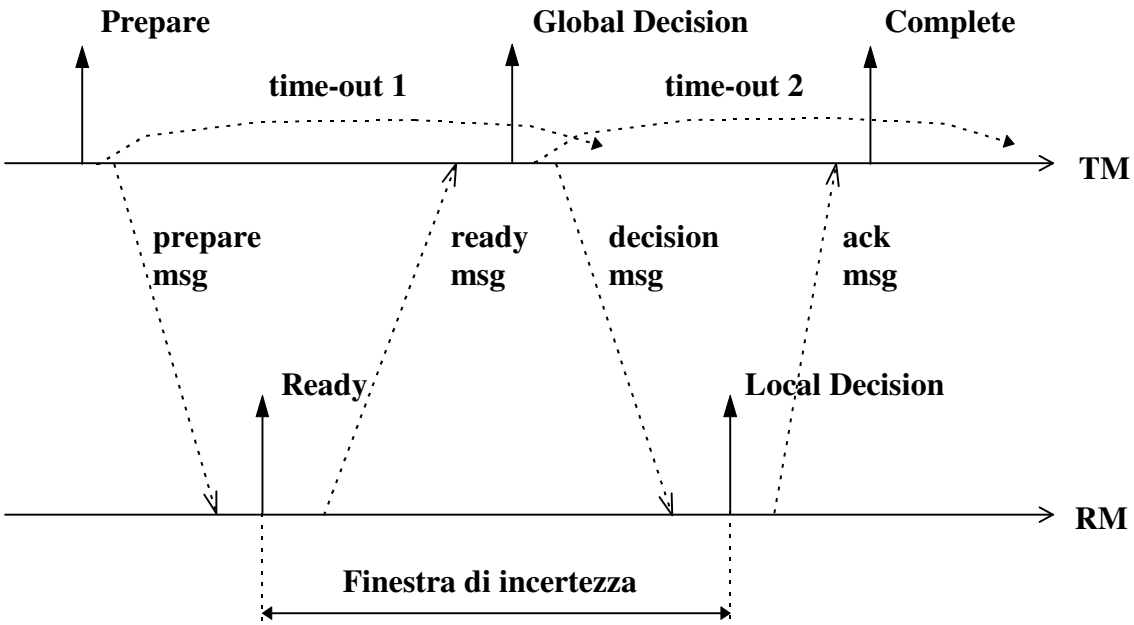
Protocollo in assenza di guasti: fase 2

- La seconda fase del protocollo di commit a due fasi in assenza di guasti si articola nelle seguenti operazioni:
    - 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, finché tutti gli RM non hanno risposto

Abort globale vs. ripetizione della trasmissione

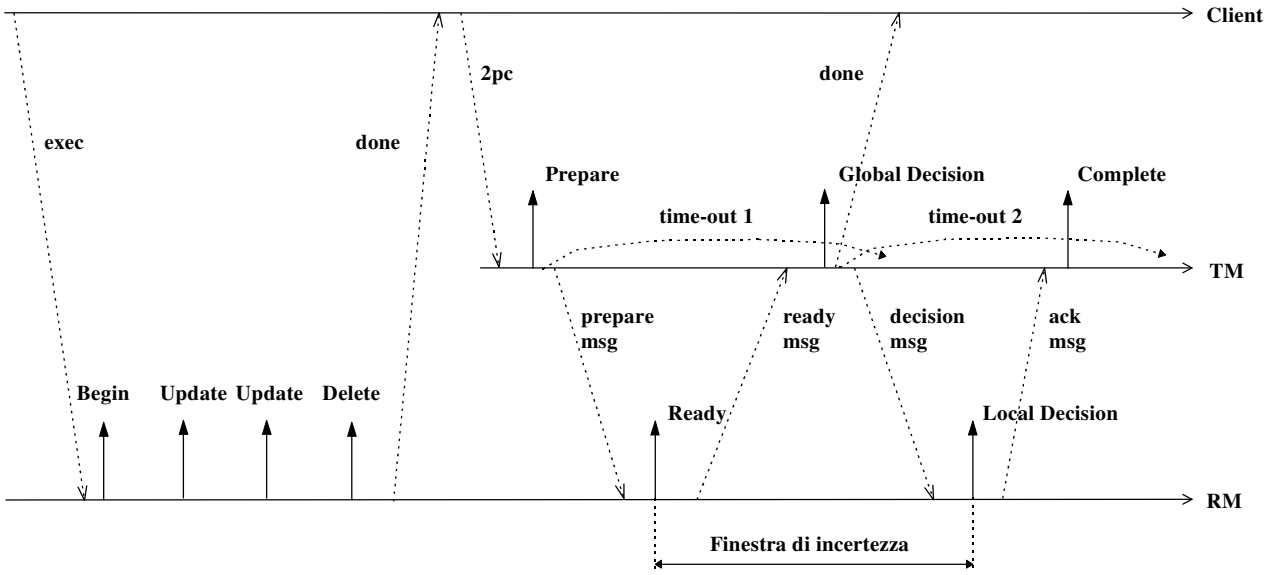
- Assenza di comunicazione tra TM e RM nella prima fase: abort globale

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



Un esempio (semplificato)

- Interazione tra client, server e TM



- Il protocollo di commit è rapido: TM e RM scrivono nel file di log e mandano messaggi

  * minimizzazione della finestra di incertezza

Protocolli di ripristino

- Possibili cause di errore:
    - Caduta di un partecipante
    - Caduta del coordinatore
    - Perdita di messaggi e partizionamenti delle rete

Caduta di un partecipante-1

- La caduta di un partecipante provoca la perdita del contenuto del buffer e può quindi lasciare la base di dati in uno stato inconsistente. Lo stato del partecipante è recuperato dal relativo file di log.

  * Ripresa a caldo:

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

    ⇓ se l'ultimo record è un *commit*, le azioni vanno *rifatte*.

Caduta di un partecipante-2

∗ Caso critico: l'ultimo record nel log è di *ready*.

⇓ Per tutte le transazioni in dubbio (collezionate nell'insieme READY), si richiede l'esito finale della transazione al processo master:

- richiesta diretta del nodo RS al nodo TM, o remote recovery;

- ripetizione della seconda fase del protocollo, che ad intervalli regolari ri-invia la decisione;

- esplicita richiesta di effettuare la recovery (è il caso, ad esempio, del protocollo X-open).

Caduta del coordinatore-1

- La caduta del coordinatore avviene durante la trasmissione dei messaggi e comporta la loro possibile perdita. Dal log è possibile ricavare informazioni sullo stato di avanzamento del protocollo, ma non su quali messaggi siano stati inviati correttamente. Distinguiamo tre possibili stati di TM:

    * se l'ultimo record nel log è un *prepare*, alcuni RM possono essere in situazione di blocco. Normalmente,

        ⇓ il TM decide un *global abort* e poi svolge la seconda fase del protocollo. In alternativa,

        ⇓ il TM ripete anche la prima fase, sperando che tutti gli RM siano ancora in attesa nello stato di ready, al fine di poter decidere un *global commit*.

Caduta del coordinatore-2

- se l'ultimo record nel log è un *global commit* o un *global abort*, alcuni RM possono essere in situazione di blocco (quelli cui TM non è riuscito a comunicare la decisione presa). In tal caso,

  ⇓ il TM ripete la seconda fase del protocollo.

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

• Si osservi che la ripetizione della seconda fase del protocollo (a causa della scadenza dei time-out o durante la recovery dopo la caduta di RM o TM) può causare molteplici ricezioni del medesimo messaggio da parte di un partecipante. Questi può ignorare il messaggio, ma deve sempre inviare un ack per consentire al TM di completare il protocollo.

Perdita di messaggi e partizionamento della rete

- Rispetto alla perdita di messaggi e ai partizionamenti delle rete, distinguiamo i seguenti casi:

  * perdita di un *prepare* o del successivo *ready* (indistinguibili dal TM)

    $\Downarrow$ time-out sulla prima fase e *global abort*;

  * perdita di un *global abort*/*global commit* o del successivo *ack* (indistinguibili dal TM)

    $\Downarrow$ time-out sulla seconda fase e ripetizione;

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

Ottimizzazione del commit a due fasi

- Il protocollo descritto è abbastanza oneroso. In particolare,
  - assunzione di scritture nel log *sincrone* (tramite una *force*) per garantire la persistenza.
  - Per ottimizzare il protocollo, scelta per default dell'esito di una transazione, in assenza di informazione circa alcuni partecipanti

    $\Downarrow$ protocollo di abort presunto

    $\Downarrow$ protocollo di commit presunto

Protocollo di abort presunto

- Il protocollo di abort presunto, adottato dalla maggior parte dei DBMS commerciali, si basa sulla seguente regola:

  - 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*.

- Vantaggi:

  - i record di *prepare* e *global abort* non sono più critici (si evita il force), così come il record di *complete* (in genere, una sua perdita causa la ripetizione della seconda fase);

  - devono essere scritti in modo sincrono solo i record di *ready* e *commit*, nel log dell'RM, ed il record di *global commit*, nel log del TM.

Ottimizzazione sola lettura

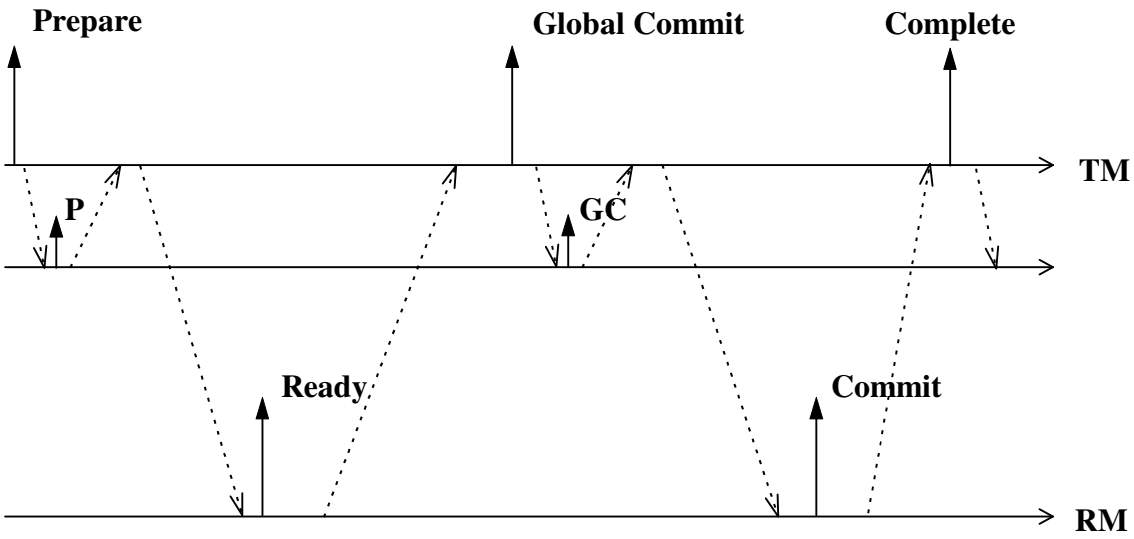
- Se un partecipante scopre di aver svolto solo operazioni di lettura, non deve influenzare l'esito finale della transazione. (I partecipanti di cui si conosce a priori la partecipazione "solo lettura" vanno esclusi dal protocollo.)

- Al messaggio di *prepare*, ogni partecipante "solo lettura" avvisa il TM, che lo ignorerà nella seconda fase del protocollo

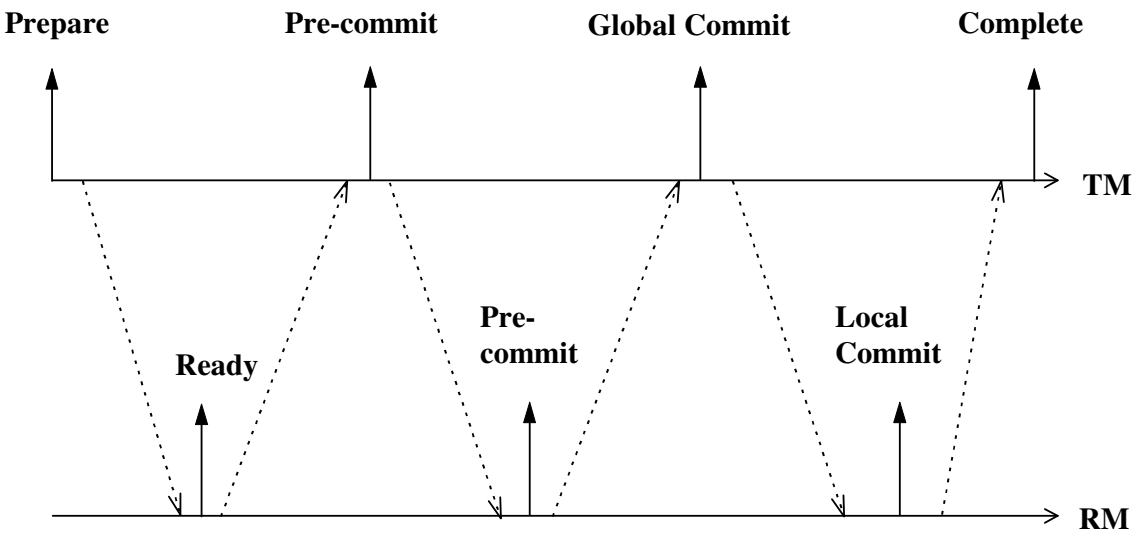
Altri protocolli di commit

- Per evitare che un RM resti bloccato a causa del TM,

  * Protocollo di commit a quattro fasi



  * Protocollo di commit a tre fasi


Prepare
Pre-commit
Global Commit
Complete
TM
Ready
Pre-
commit
Local
Commit
RM

Interoperabilità

- Interazione di sistemi di basi di dati (distribuiti) *eterogenei*
- Standard (numerosi e contrapposti)
    - ODBC (per garantire accessi remoti senza commit a due fasi)
    - X-OPEN (standard relativo al protocollo di commit)

Lo standard ODBC

- Open DataBase Connectivity: ODBC
  - interfaccia proposta da Microsoft nel 1991
  - attraverso un'interfaccia ODBC, applicazioni SQL possono accedere a dati remoti
  - SQL "ristretto" (minimo insieme di istruzioni)
- Architettura ODBC:
  - tra applicazione e server deve esserci un *driver* (libreria collegata dinamicamente alle applicazioni e da esse invocata);
  - il *driver* maschera le differenze di interazione dovute a DBMS, protocollo di rete e sistema operativo
  - ad esempio, la tripla (*Sybase*, *Windows/NT*, *Novell*) identifica uno specifico driver

Accessi tramite ODBC

- Accesso a database remoto tramite ODBC

  * l'*applicazione* richiama le funzioni SQL per interrogare ed acquisire i risultati (l'applicazione ignora il protocollo di comunicazione, il server DBMS e il sistema operativo del nodo ove il DBMS è installato);

  * il *driver manager* carica i driver necessari all'applicazione (software fornito da Microsoft, gestisce anche le corrispondenze fra nomi e funzioni di inizializzazione);

  * i *driver* sono responsabili dell'esecuzione di funzioni ODBC (interrogazione attraverso SQL specifici e bufferizzazione dei risultati);

  * la fonte di informazione (*data source*) è il sistema remoto che esegue quanto richiesto dal client.

Transazioni ODBC

- In ODBC è possibile richiedere i comandi transazionali commit-work e rollback-work, che garantiscono l'atomicità delle transazioni
    - tali comandi vanno rivolti esplicitamente ad un server, in quanto l'ODBC non supporta direttamente il protocollo di commit a due fasi
    - codici di errore standardizzati, per consentire il controllo delle condizioni di errore a tempo di esecuzione
    - possibilità di interrogazioni statiche (compilate) e dinamiche (generate ed eseguite dinamicamente)

Il protocollo X-open DTP

- Commit standard: X-OPEN DTP (Distributed Transaction Processing)
    - garantisce l'interoperabilità di computazioni transazionali su DBMS di fornitori differenti
    - architettura composta da un processo client, vari processi RM ed un processo TM
- Il protocollo X-OPEN DTP consta di due interfacce:
    - interfaccia tra client e TM: *TM-interface*
    - interfaccia tra TM ed RM: *XA-interface*
- Per garantire che i TM possano accedere ai loro server, i costruttori di DBMS devono mettere a disposizione delle XA-interface. Vari produttori (Informix, Oracle) forniscono sia una realizzazione proprietaria del

protocollo di commit a due fasi, che un'implementazione dell'intefaccia XA.

Caratteristiche di X-open DTP

- X-OPEN DTP

  * RM totalmente passivi; il controllo è concentrato interamente completamente nel TM che attiva le funzioni degli RM, rese disponibili come libreria di primitive richiamabili remotamente

  * commit a due fasi, con ottimizzazione tramite *abort presunto* e *sola lettura*

  * decisioni euristiche, che consentono il controllo dell'operatore sulle transazioni in presenza di guasti (tali decisioni possono causare una perdita di atomicità; in tale eventualità, il protocollo garantisce la notifica ai processi client)

L'interfaccia TM

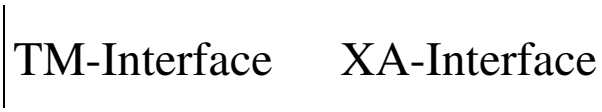
- L’interfaccia TM consta delle seguenti procedure:
  - tm_init e tm_exit, per iniziare e terminare il dialogo client-TM;
  - tm_open, cui segue l’apertura di una sessione del TM con i vari RM, e tm_term, chiusura della sessione;
  - tm_begin, per iniziare una transazione;
  - tm_commit, per richiedere un commit globale

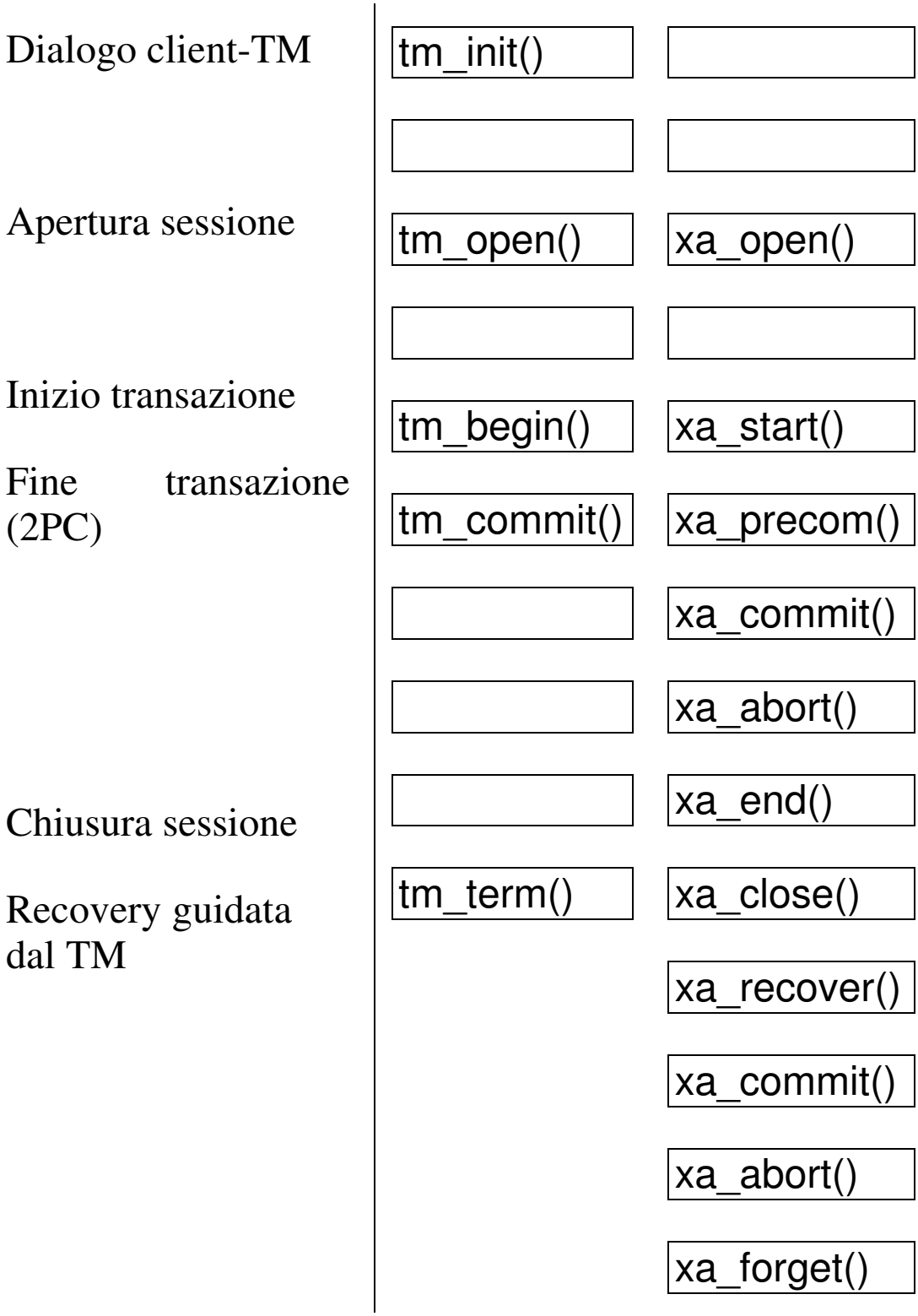
L'interfaccia XA

- L’interfaccia XA consta delle seguenti procedure:
  - 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.

Un esempio

- Esempio di interazione tra client, TM e RM




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()

Procedura di ripristino

- Se un RM è bloccato per la caduta di un TM, l'operatore può 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 e 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 più processori
    - OLAP: On Line Analytical Processing

Parallelismo e frammentazione dei dati - 1

- 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 e frammentazione dei dati - 2

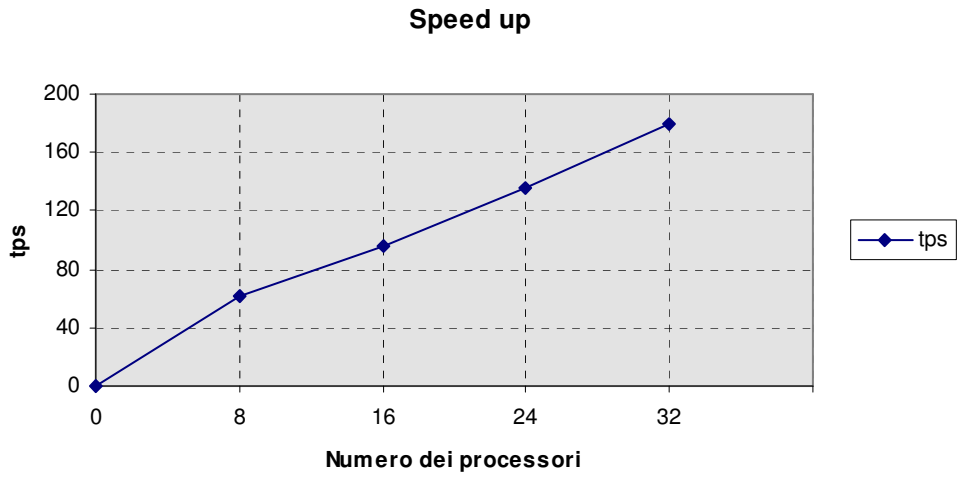
- Frammentazione dei dati

  * Interrogazione OLAP

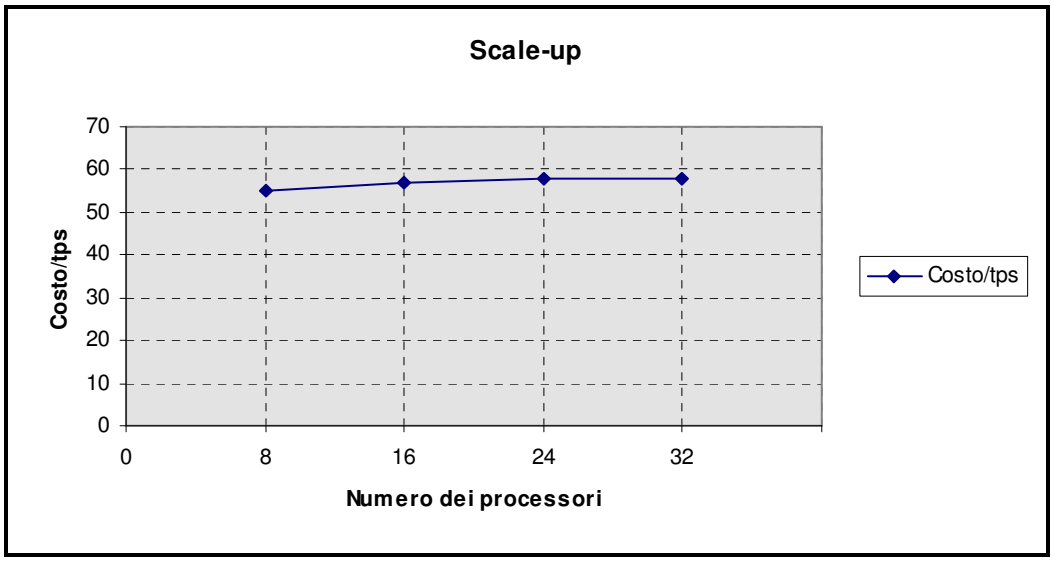
```
procedure Query6();
select Nome, sum(Amm)
from ContoCorrente join Movimento on
        ContoCorrente.CCNum =
        Movimento.CCNum
where Data > 1.1.96
group by CCNum, Nome
having sum(abs(Ammontare)) > 100 M;
      end procedure;
```

Speed-up e Scale-up

- Speed-up e Scale-up



- Scale-up



Benchmark delle transazioni

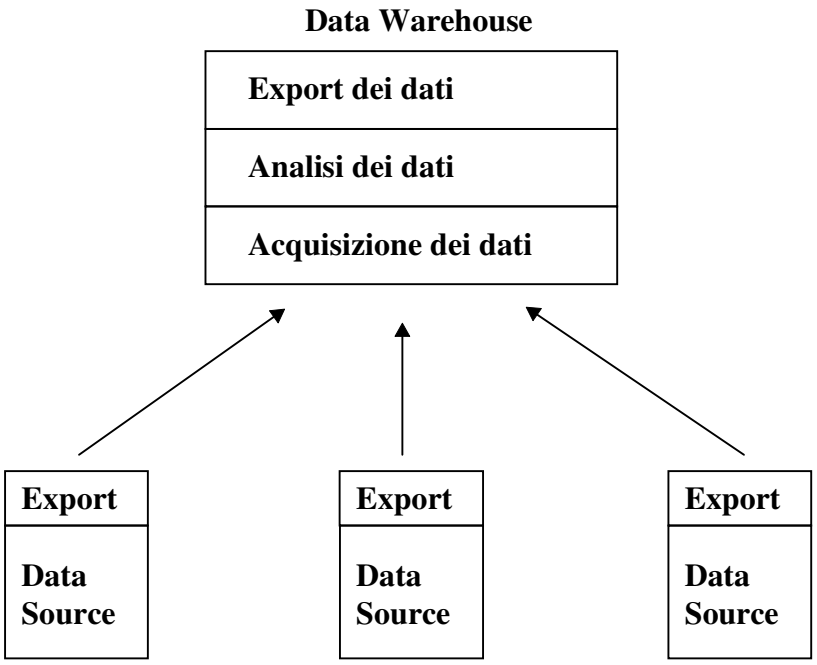
- Benchmark delle transazioni
  - ∗ valutazione delle prestazioni delle architetture di basi di dati
  - ∗ TPC (Transaction Processing Performance Council)
    - ⇓ TPC-A, per applicazioni OLTP
    - ⇓ TPC-B, per applicazioni miste
    - ⇓ TPC-C per applicazioni OLAP
  - ∗ Specifiche del benchmark:
    - ⇓ 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);
    - ⇓ modalità di misurazione e di certificazione della validità 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

Architettura - 1

- Architettura di una Data warehouse

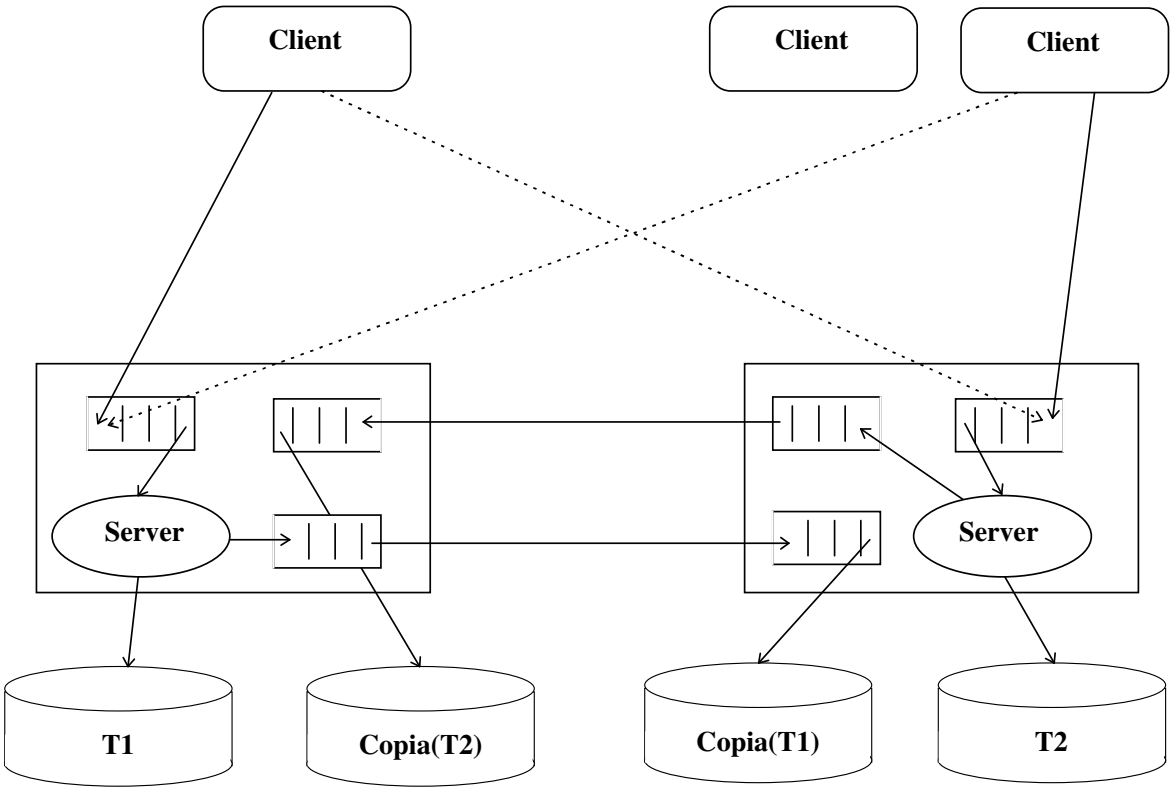


Architettura - 2

- 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

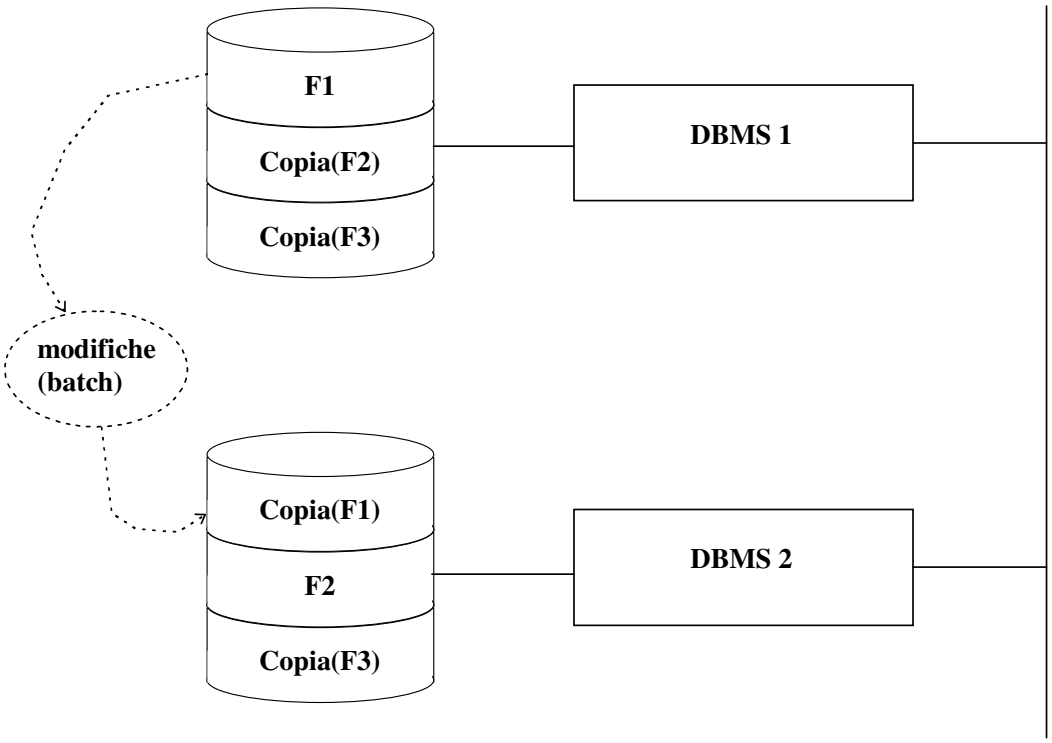
Basi di dati replicate - 1

- Solidità ai guasti
- Forma sofisticata di backup



Basi di dati replicate - 2

- Replicazione, distribuzione e frammentazione



- Replicazione simmetrica

  * peer-to-peer (sistemi distribuiti mobili)