
Parte IX

Basi di dati distribuite e parallele

Architetture distribuite e parallele

- Diverse soluzioni architetture (sia hardware che software), la cui rilevanza va progressivamente crescendo
- Architetture client-server: accentrimento delle funzioni del DBMS in un modulo *server*, e spostamento su moduli *client* di tutto il carico applicativo e di gestione dell'interfaccia utente
- Basi di dati distribuite: distribuzione paritaria tra più nodi di una rete della base di dati e delle funzioni del DBMS
- Basi di dati parallele: DBMS progettati per sfruttare le particolari caratteristiche di architetture multiprocessore

Tipologie di accesso ai dati

- Due modalità diverse di accesso, cui corrispondono diverse tipologie di sistemi
- On-Line Transaction Processing (OLTP): gestione in linea dei dati con carico transazionale pesante, proveniente da terminali o client distribuiti su rete
- On-Line Analytical Processing (OLAP): strumento di base per un sistema di supporto alle decisioni che richiede analisi complesse, svolte nelle *data warehouse* in cui i dati vengono esportati da sistemi OLTP

Portabilità ed interoperabilità

- Sono i due requisiti fondamentali che devono essere soddisfatti dalle architetture distribuite per basi di dati
- Portabilità: è la possibilità di trasferire le applicazioni da una configurazione di sistema (e di rete) all'altra (viene comunque assicurata nell'ambito di una data famiglia di DBMS (es. Oracle, Informix, etc.))
- Interoperabilità: è la possibilità di far interagire sistemi eterogenei, e di costruire su di essi applicazioni integrate
- Sono stati introdotti standard, sia per il linguaggi (es. SQL), sia per i protocolli di accesso ai dati (ODBC, X-Open DTP)

Architettura client-server

- Si basa su di un paradigma ormai consolidato nella realizzazione di applicazioni distribuite
- Architettura *software* distribuita che risulta da una decomposizione delle applicazioni in moduli di due tipi
- Server : mettono a disposizione una serie di *servizi* la cui esecuzione può essere richiesta da altri moduli
- Client: realizzano le loro funzioni sfruttando i servizi offerti dai server
- Uno stesso modulo si può comportare come server nei confronti di alcuni moduli e come client nei confronti di altri

DBMS client-server

- Il server è un *database server* che provvede alla gestione dei dati e delle transazioni, e accetta richieste di servizi consistenti in comandi SQL
- I moduli *client* contengono la parte relativa alla gestione dell'interfaccia utente ed un'interfaccia di comunicazione verso il database server
- Di fatto il server ed i client sono allocati su macchine diverse, sistema dipartimentale o mainframe per il server e workstation o PC per i client
- Tutta la parte applicativa del carico grava sui client, e ciò consente sia una gestione più sofisticata dell'interfaccia, che una scalabilità del sistema

Basi di dati distribuite

- Sono presenti più database server, su cui girano DBMS che possono essere tra loro sia omogenei che eterogenei, e interconnessi da reti locali o geografiche
- Diverse soluzioni utilizzate nei diversi settori applicativi

TIPO 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

Frammentazione dei dati

- Ha il fine di permettere una distribuzione dei dati che ne razionalizzi l'accesso, e riduca i costi di trasmissione
- Data una relazione **R** essa può essere divisa in un insieme di frammenti $\{R_i\}$, tramite l'applicazione di operatori algebrici
- Frammentazione orizzontale: ciascun frammento R_i è un insieme di tuple con lo stesso schema di **R**, e viene ottenuto da **R** tramite una *selezione*
- Frammentazione verticale: ciascun frammento R_i ha uno schema che è un sottoinsieme dello schema di **R**, e viene ottenuto da **R** tramite una *proiezione*

Proprietà di una frammentazione

- Affinché la frammentazione di una relazione **R** sia *corretta* devono essere verificate due proprietà
 - Completezza: ogni dato di **R** deve essere presente in almeno un frammento **R_i**
 - Ricostruibilità: **R** deve essere completamente ricostruibile a partire dai suoi frammenti **R_i**
- I frammenti orizzontali sono tipicamente disgiunti
- I frammenti verticali includono tutti la chiave primaria di **R** in modo da garantire la ricostruibilità (*decomposizione senza perdita*)

Frammentazione: esempio

Studente(Matricola, Nome, Cognome, Nascita)

- Una *frammentazione orizzontale* è:

Studente1 = SEL_{Matricola < 7000} (Studente)

Studente2 = SEL_{Matricola ≥ 7000} (Studente)

la relazione è ricostruibile tramite una unione:

Studente = Studente1 UNION Studente2

- Una *frammentazione verticale* è:

Studente1 = PROJ_{Matricola, Corso} (Studente)

Studente2 = PROJ_{Matricola, Nome, Cognome} (Studente)

la relazione è ricostruibile tramite un join naturale:

Studente = Studente1 JOIN Studente2

Esempio: la tabella Studente

Studente

Matricola	Cognome	Nome	Corso
27655	Rossi	Mario	Ing Inf
78763	Rossi	Mario	Ing Civile
65432	Neri	Piero	Ing Mecc
87654	Neri	Mario	Ing Inf
67653	Rossi	Piero	Ing Mecc

Esempio: frammentazione orizzontale

Studente1

Matricola	Cognome	Nome	Corso
27655	Rossi	Mario	Ing Inf
65432	Neri	Piero	Ing Mecc
67653	Rossi	Piero	Ing Mecc

Studente2

Matricola	Cognome	Nome	Corso
78763	Rossi	Mario	Ing Civile
87654	Neri	Mario	Ing Inf

Esempio: frammentazione verticale

Studente1

Matricola	Corso
27655	Ing Inf
78763	Ing Civile
65432	Ing Mecc
87654	Ing Inf
67653	Ing Mecc

Studente2

Matricola	Cognome	Nome
27655	Rossi	Mario
78763	Rossi	Mario
65432	Neri	Piero
87654	Neri	Mario
67653	Rossi	Piero

Allocazione dei frammenti

- Uno *schema di allocazione* definisce il mapping fra frammenti e server, che può essere:
 - *Non ridondante*: ciascun frammento (tabella) è allocato su un solo server
 - *Ridondante*: qualche frammento (tabella) è allocato su più server
- La ridondanza ha vantaggi dal punto di vista delle prestazioni e della disponibilità, ma pone problemi per il mantenimento della consistenza fra copie dello stesso frammento
- Frammentazione e allocazione dei frammenti possono essere più o meno visibili a livello applicativo (*livelli di trasparenza*)

Allocazione dei frammenti: esempio

- Data la tabella

Fornitore(Fnum, Nome, Città)

frammentata orizzontalmente:

Fornitore1 = SEL_{Città=Milano} (Fornitore)

Fornitore2 = SEL_{Città=Roma} (Fornitore)

- Un possibile schema di allocazione sui tre server **milano, roma1 e roma2** è:

Fornitore1@ditta.milano.it

Fornitore2@ditta.roma1.it

Fornitore2@ditta.roma2.it

Trasparenza di frammentazione

Il programmatore non vede né la frammentazione né l'allocazione dei frammenti. Le query possono essere espresse come se le relazioni non fossero frammentate.

Esempio

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

Trasparenza di allocazione

Il programmatore vede la frammentazione ma non l'allocazione dei frammenti. Le query fanno riferimento esplicito ai frammenti ma non a dove sono allocati (trasparenza di relicazione)

Esempio

```
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;
```

Trasparenza di linguaggio

Il programmatore fa esplicito riferimento alla allocazione dei segmenti. L'unica trasparenza è data dalla presenza di un linguaggio di interrogazione omogeneo per tutti i server

Esempio

```
procedure Query3(:fnum,:nome);
select Nome into :nome
from Fornitore1ditta.milano.it
where Fnum = :fnum;
if :empty then
select Nome into :nome
from Fornitore2@ditta.roma1.it
where Fnum = :fnum;
end procedure;
```

Ottimizzazione degli accessi

Nei livelli di trasparenza di allocazione e frammentazione le scelte sono fatte dall'ottimizzatore; ma possono essere anche guidate dal programmatore:

Esempio

```
procedure Query4(:fnum,:nome, :citta);
case :citta of
"Milano":
select Nome into :nome
from Fornitore1
where Fnum = :fnum;
"Roma":
select Nome into :nome
from Fornitore2
where Fnum = :fnum;
end procedure;
```

Tipi di transazioni

- Ogni client è connesso ad un solo DBMS al quale sottomette le sue transazioni, e questo provvede a *distribuire* la transazione
- Possiamo classificare le transazioni in:
 - *Richieste remote* (*remote request*): transazioni di sola lettura indirizzate ad un solo DBMS remoto.
 - *Transazioni remote* (*remote transaction*): transazioni costituite da un numero qualsiasi di comandi SQL (select, insert, delete, update) dirette ad un solo DBMS remoto
 - *Transazioni distribuite* (*distributed transaction*): transazioni rivolte a più DBMS ma in cui ciascun comando SQL fa riferimento a dati memorizzati su di un solo DBMS

Transazioni distribuite: esempio

In questo esempio la transazione distribuita fa riferimento a una relazione **ContoCorrente(CCNum, Nome, Saldo)** partizionata orizzontalmente, ed è scritta al livello della trasparenza di allocazione:

Esempio

```
begin transaction
update ContoCorrente1
set Saldo = Saldo - 100
where CCNum = 3154;
update ContoCorrente2
set Saldo = Saldo + 100
where CCNum = 14878;
commit work;
end transaction
```

Gestioni delle transazioni distribuite

- Il problema è quello del mantenimento delle *proprietà acide* delle transazioni
- La *persistenza* non pone nuovi problemi dato che ciascun sistema garantisce la persistenza dati presso di esso allocati
- La *consistenza* pone problemi nel caso di vincoli di integrità che fanno riferimento a dati distribuiti. Ciò comunque *non è consentito* dagli attuali DBMS
- Problemi ci sono invece per la *atomicità* e per l'*isolamento*

Serializzabilità globale

- Ciascuna *transazione distribuita (globale)* T_i consiste nella esecuzione di più *sottotransazioni* $T_{i,k}$ sui nodi n_k del DBMS distribuito su cui opera
- I singoli DBMS locali assicurano la *serializzabilità locale*, cioè relativamente alle schedule che ciascuno di essi genera
- Questo purtroppo non è sufficiente a garantire l'isolamento della transazione globale
- Occorre introdurre un nuovo criterio di serializzabilità a livello globale, e protocolli che lo garantiscano

Protocolli di commit

- Per garantire l'atomicità occorre che tutti i nodi che partecipano ad una transazione raggiungano la stessa decisione riguardo al suo esito (*commit* o *abort*)
- La decisione viene presa tramite scambio di messaggi secondo un *protocollo di commit*, che deve anche essere resistente a vari tipi di guasti:
 - caduta di un nodo
 - perdita di un messaggio: oppure del rispettivo *ack* (*acknowledgement*) inviato dal destinatario
 - partizionamento della rete: due o più insiemi di sottotransazioni restano attive in sottoreti temporaneamente sconnesse

Two phase commit (2PC)

- Prevede uno scambio di messaggi tra:
 - Un processo coordinatore detto *Transaction Manager (TM)* che gira sul nodo a cui la transazione globale è stata sottomessa
 - Un insieme di processi detti *Resource Manager (RM)*, uno per ogni nodo su cui la transazione abbia una sottotransazione.
- Sia il TM che gli RM oltre a scambiarsi messaggi scrivono opportuni record nei loro log, per consentire una fase di recovery in seguito a guasti
- È implementato da tutti i DBMS commerciali

Interoperabilità

- Problema cruciale nella realizzazione di *basi di dati distribuite eterogenee*
- Purtroppo nessuno standard si è ancora definitivamente affermato ed esistono varie proposte in contrapposizione
- In particolare esaminiamo:
 - ODBC, uno standard per garantire accessi remoti senza commit a due fasi
 - X-OPEN, uno standard relativo al protocollo di commit

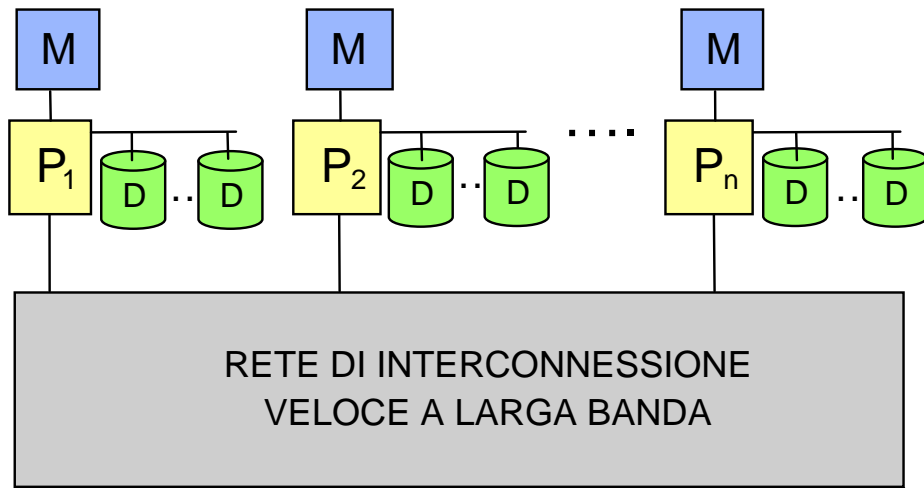
Open DataBase Connectivity (ODBC)

- Proposto inizialmente da Microsoft nel '91, è un'interfaccia applicativa supportata dalla maggior parte dei sistemi relazionali
- Permette l'accesso a dati remoti tramite un sottoinsieme 'minimo' dell'SQL definito dall' *SQL Access Group*
- Le applicazioni comunicano con il server tramite una libreria denominata *driver*, che rende l'accesso trasparente rispetto a:
 - il DBMS utilizzato
 - il sistema operativo
 - il protocollo di rete
- Ciascun produttore deve fornire un driver per ogni coppia sistema operativo-protocollo di rete.

Accesso remoto tramite ODBC

- Concorrono quattro componenti:
 - L'*applicazione* richiama funzioni SQL per eseguire interrogazione SQL e per acquisirne i risultati, attraverso il driver che maschera il protocollo di rete, il server DBMS e il sistema operativo del nodo server
 - Il *driver manager* è responsabile di caricare i driver a richiesta dell'applicazione (fornito da Microsoft)
 - I *driver* eseguono le funzioni ODBC traducendo opportunamente le interrogazioni in SQL, e bufferizzando i risultati
 - La *data source* cioè il sistema remoto, che esegue le funzioni trasmesse dal *client*

Architetture parallele



Architetture parallele (2)

- Piattaforme parallele permettono di realizzare DB server per carico molto elevato
- Usate sempre di più in alternativa ai *mainframe*
- Basate su piattaforme di tipo *shared-nothing*
- DBMS paralleli con architettura specializzata per sfruttare il parallelismo
- Partizionamento dei dati su più nodi per aumentare il livello di elaborazione parallela
- I maggiori DBMS relazionali sono oggi disponibili anche in versione parallela
- Problemi notevoli di configurazione e di tuning