# Basi di dati distribuite

1

## Paradigmi per la distribuzione dei dati

- **Architettura client-server:** separazione tra client e server della base di dati
- **Basi di dati distribuite:** la stessa applicazione utilizza diversi database server
- **Basi di dati parallele:** diversi supporti di memorizzazione e processori utilizzati in parallelo per migliorare le prestazioni
- **Basi di dati replicate:** dati che rappresentano logicamente la stessa informazione sono memorizzati fisicamente su diversi server



## Architettura client-server

- è un modello generale di interazione tra processi software, in cui i processi che interagiscono sono suddivisi in
  - **client:** che richiedono servizi, e
  - **server:** che offrono servizi
- richiede la definizione precisa di una *interfaccia di servizi*, che specifica i servizi offerti dal server

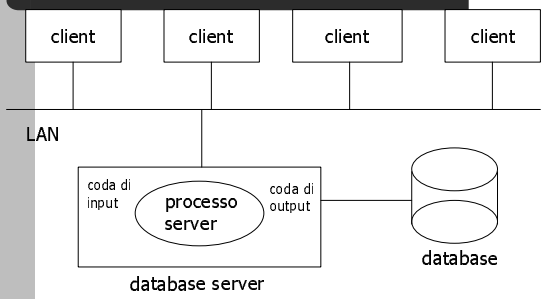


## Architettura client-server

- il processo client ha un ruolo attivo
  (genera autonomamente richieste di servizi)
- il processo server è un processo reattivo (svolge una computazione solo a seguito di una richiesta da parte di un client)
- il processo client richiede un insieme limitato di servizi sequenzialmente ad uno o più processi server
- un processo server risponde a molte richieste provenienti da molti processi client



## Architettura client-server






## Architettura client-server

- nella gestione dati, l'allocazione di processi client e server a computer differenti è ormai molto diffusa
  - le funzionalità di client e server sono ben identificate
  - portano a una separazione conveniente delle attività di progettazione e gestione
- le caratteristiche delle macchine destinate ai client e al server possono essere molto diverse
  - i computer dedicati a client devono essere adatti all'interazione con l'utente
  - i computer dedicati al server devono avere una memoria di capacità elevata (per la gestione del buffer) e un disco di capacità elevata (per la memorizzazione della base di dati)

## Architettura client-server

- SQL offre un paradigma di programmazione ideale per l'identificazione dell'interfaccia di servizi
  - le interrogazioni SQL sono formulate dal client e inviate al server
  - i risultati delle interrogazioni sono calcolati dal server e restituiti al client
  - grazie alla standardizzazione offerta da SQL è possibile sviluppare applicazioni client che coinvolgono diversi sistemi server



## Architettura client-server

- spesso il server è *multi-threaded*
  - si comporta come un singolo processo che lavora dinamicamente per conto di diverse transazioni
  - ogni unità di esecuzione del processo server per una certa transazione è detta *thread*
- i server sono processi permanentemente attivi che controllano una *coda di input* per le richieste dei client e una *coda di output* per i risultati delle interrogazioni
- è spesso presente un processo *dispatcher* che distribuisce le richieste ai server e restituisce le risposte ai client



## Architettura client-server

- **architettura two-tier:**
  - il client è sia l'interfaccia utente che il gestore dell'applicazione
  - si parla di *thick-client* in quanto il client contiene la logica dell'applicazione
- **architettura three-tier:**
  - è presente un secondo server, chiamato *application server*, responsabile della gestione della logica dell'applicazione che è comune a molti client
  - si parla di *thin-client* in quanto il client è responsabile unicamente dell'interfaccia con l'utente finale



## Basi di dati distribuite

- una base di dati distribuita è un sistema in cui almeno un client interagisce con più server per l'esecuzione di un'applicazione
- le basi di dati distribuite richiedono di riesaminare
  - come l'utente può specificare le interrogazioni
  - come la tecnologia server può essere estesa ad un contesto distribuito



## Basi di dati distribuite

- indipendenza dalla distribuzione dei dati: gli utenti non devono sapere dove i dati sono memorizzati (estende i principi di indipendenza logica e fisica dei dati)
- atomicità distribuita delle transazioni: gli utenti dovrebbero essere in grado di scrivere transazioni che accedono più siti esattamente come transazioni locali
- queste due proprietà non sono in genere supportate, per problemi di efficienza ⇒ gli utenti devono sapere dove i dati sono memorizzati



## Basi di dati distribuite

- rispondono a esigenze applicative naturali

  le imprese sono strutturalmente distribuite, una gestione distribuita dei dati permette di distribuire il controllo e l'elaborazione dei dati dove questi sono generati e maggiormente utilizzati
- offrono maggiore flessibilità, modularità e resistenza ai guasti
  - possono essere configurate con aggiunta e modifica progressiva di componenti
  - anche se sono maggiormente soggette a guasti a causa della maggiore complessità strutturale, rispondono a guasti con una riduzione delle prestazioni piuttosto che con un failure totale

## Classificazione delle applicazioni

- rispetto al tipo di DBMS coinvolti:
  - **basi di dati distribuite omogenee**: tutti i server hanno lo stesso DBMS
  - **basi di dati distribuite eterogenee**: i server hanno DBMS diversi
- rispetto alla rete:
  - possono utilizzare una rete locale LAN (local area network)
  - possono utilizzare una rete WAN (wide area network)



## Classificazione delle applicazioni

| | LAN | WAN |
|---|---|---|
| omogenee | applicazioni gestionali e finanziarie | applicazioni finanziarie e sistemi di prenotazione |
| eterogenee | sistemi informativi interdivisionali | sistemi di prenotazione integrati, sistemi interbancari |



## Basi di dati distribuite

- in una base di dati distribuita ogni server ha la capacità di gestire le applicazioni in modo indipendente
- una base di dati distribuita dovrebbe minimizzare le interazioni e la necessità di trasmettere dati attraverso la rete
- la distribuzione dei dati dovrebbe essere pianificata in modo tale che le applicazioni possano operare in modo indipendente su un singolo server



## Frammentazione e allocazione dei dati

- data una relazione R, la sua frammentazione consiste nel determinare dei frammenti $R_i$ di R applicando a R operazioni algebriche
  - **frammentazione orizzontale**: i frammenti $R_i$ sono gruppi di tuple che hanno lo stesso schema di R
    (ottenibili come selezione su R)
    i frammenti orizzontali sono generalmente disgiunti
  - **frammentazione verticale**: ogni frammento $R_i$ ha come schema un sottoschema dello schema di R
    (ottenibile come proiezione su R)
    i frammenti verticali devono includere la chiave primaria di R



## Frammentazione e allocazione dei dati

- una frammentazione è corretta se è
  - *completa:* ogni dato in R deve essere presente in uno dei frammenti $R_i$
  - *ricostruibile:* il contenuto di R deve essere ricostruibile a partire da quello dei suoi frammenti
- la frammentazione è una tecnica di organizzazione dei dati che permette una distribuzione e un'elaborazione efficiente dei dati



## Frammentazione dei dati

- consideriamo la relazione

IMPIEGATI(Imp#, Nome, Mansione, Data_A,Stipendio, Premio_P,Dip#)

- frammentazione orizzontale:
  - IMP1 = $\sigma_{Dip\#=10}$ IMPIEGATI
  - IMP2 = $\sigma_{Dip\#=20\ OR\ Dip\#=30}$ IMPIEGATI
- la ricostruzione richiede un'unione:
  IMPIEGATI = IMP1 $\cup$ IMP2

## Frammentazione dei dati

IMPIEGATI(Imp#, Nome, Mansione, Data_A,Stipendio, Premio_P,Dip#)

- frammentazione verticale:
  - IMP1 = $\pi_{Imp\#,Nome,Mansione,Dip\#}$ IMPIEGATI
  - IMP2 = $\pi_{Imp\#,Data\_A,Stipendio,Premio\_P}$ IMPIEGATI
- la ricostruzione richiede un join naturale sul valore di chiave:

  IMPIEGATI = IMP1 |X| IMP2



## Frammentazione dei dati

ESEMPIO - IMPIEGATI

| Imp# | Nome | Mansione | Data_A | Stipendio | Premio_P | Dip# |
|---|---|---|---|---|---|---|
| 7369 | Rossi | ingegnere | 17-Dic-80 | 1600,00 | 500,00 | 20 |
| 7499 | Andrei | tecnico | 20-Feb-81 | 800,00 | ? | 30 |
| 7521 | Bianchi | tecnico | 20-Feb-81 | 800,00 | 100,00 | 30 |
| 7566 | Rosi | dirigente | 02-Apr-81 | 2975,00 | ? | 20 |
| 7654 | Martini | segretaria | 28-Set-81 | 800,00 | ? | 30 |
| 7698 | Blacchi | dirigente | 01-Mag-81 | 2850,00 | ? | 30 |
| 7782 | Neri | ingegnere | 01-Giu-81 | 2450,00 | 200,00 | 10 |
| 7788 | Scotti | segretaria | 09-Nov-81 | 800,00 | ? | 20 |
| 7839 | Dare | ingegnere | 17-Nov-81 | 2600,00 | 300,00 | 10 |
| 7844 | Turni | tecnico | 08-Set-81 | 1500,00 | ? | 30 |
| 7876 | Adarni | ingegnere | 28-Set-81 | 1100,00 | 500,00 | 20 |
| 7900 | Gianni | ingegnere | 03-Dic-81 | 1950,00 | ? | 30 |
| 7902 | Fordi | segretaria | 03-Dic-81 | 1000,00 | ? | 20 |
| 7934 | Milli | ingegnere | 23-Gen-82 | 1300,00 | 150,00 | 10 |
| 7977 | Verdi | dirigente | 10-Dic-80 | 3000,00 | ? | 10 |



## Frammentazione dei dati

IMP1

| Imp# | Nome | Mansione | Data_A | Stipendio | Premio_P | Dip# |
|---|---|---|---|---|---|---|
| 7782 | Neri | ingegnere | 01-Giu-81 | 2450,00 | 200,00 | 10 |
| 7839 | Dare | ingegnere | 17-Nov-81 | 2600,00 | 300,00 | 10 |
| 7934 | Milli | ingegnere | 23-Gen-82 | 1300,00 | 150,00 | 10 |
| 7977 | Verdi | dirigente | 10-Dic-80 | 3000,00 | ? | 10 |

IMP2

| Imp# | Nome | Mansione | Data_A | Stipendio | Premio_P | Dip# |
|---|---|---|---|---|---|---|
| 7369 | Rossi | ingegnere | 17-Dic-80 | 1600,00 | 500,00 | 20 |
| 7499 | Andrei | tecnico | 20-Feb-81 | 800,00 | ? | 30 |
| 7521 | Bianchi | tecnico | 20-Feb-81 | 800,00 | 100,00 | 30 |
| 7566 | Rosi | dirigente | 02-Apr-81 | 2975,00 | ? | 20 |
| 7654 | Martini | segretaria | 28-Set-81 | 800,00 | ? | 30 |
| 7698 | Blacchi | dirigente | 01-Mag-81 | 2850,00 | ? | 30 |
| 7788 | Scotti | segretaria | 09-Nov-81 | 800,00 | ? | 20 |
| 7844 | Turni | tecnico | 08-Set-81 | 1500,00 | ? | 30 |
| 7876 | Adarni | ingegnere | 28-Set-81 | 1100,00 | 500,00 | 20 |
| 7900 | Gianni | ingegnere | 03-Dic-81 | 1950,00 | ? | 30 |
| 7902 | Fordi | segretaria | 03-Dic-81 | 1000,00 | ? | 20 |



## Frammentazione dei dati

ESEMPIO - IMP1

| Imp# | Nome | Mansione | Dip# |
|---|---|---|---|
| 7369 | Rossi | ingegnere | 20 |
| 7499 | Andrei | tecnico | 30 |
| 7521 | Bianchi | tecnico | 30 |
| 7566 | Rosi | dirigente | 20 |
| 7654 | Martini | segretaria | 30 |
| 7698 | Blacchi | dirigente | 30 |
| 7782 | Neri | ingegnere | 10 |
| 7788 | Scotti | segretaria | 20 |
| 7839 | Dare | ingegnere | 10 |
| 7844 | Turni | tecnico | 30 |
| 7876 | Adarni | ingegnere | 20 |
| 7900 | Gianni | ingegnere | 30 |
| 7902 | Fordi | segretaria | 20 |
| 7934 | Milli | ingegnere | 10 |
| 7977 | Verdi | dirigente | 10 |



## Frammentazione dei dati

ESEMPIO - IMP2

| Imp# | Data_A | Stipendio | Premio_P |
|---|---|---|---|
| 7369 | 17-Dic-80 | 1600,00 | 500,00 |
| 7499 | 20-Feb-81 | 800,00 | ? |
| 7521 | 20-Feb-81 | 800,00 | 100,00 |
| 7566 | 02-Apr-81 | 2975,00 | ? |
| 7654 | 28-Set-81 | 800,00 | ? |
| 7698 | 01-Mag-81 | 2850,00 | ? |
| 7782 | 01-Giu-81 | 2450,00 | 200,00 |
| 7788 | 09-Nov-81 | 800,00 | ? |
| 7839 | 17-Nov-81 | 2600,00 | 300,00 |
| 7844 | 08-Set-81 | 1500,00 | ? |
| 7876 | 28-Set-81 | 1100,00 | 500,00 |
| 7900 | 03-Dic-81 | 1950,00 | ? |
| 7902 | 03-Dic-81 | 1000,00 | ? |
| 7934 | 23-Gen-82 | 1300,00 | 150,00 |
| 7977 | 10-Dic-80 | 3000,00 | ? |



## Frammentazione e allocazione dei dati

- ogni frammento $R_i$ corrisponde ad un diverso file fisico ed è allocato su un diverso server
- quindi la relazione è un'entità virtuale (come una vista) mentre i frammenti sono effettivamente memorizzati
- lo schema di allocazione descrive il mapping da relazioni o frammenti al server su cui sono memorizzati
- tale mapping può essere:
  - **non ridondante**: quando ogni frammento o relazione è memorizzato su un singolo server
  - **ridondante**: quando almeno un frammento o relazione è memorizzato su più di un server (in questo caso si parla anche di *replicazione*)

## Tecnologia delle basi di dati distribuite

- la distribuzione dei dati non influenza la proprietà di consistenza e persistenza delle transazioni
- gli altri sottosistemi devono essere rivisti in ambito distribuito:
  - ottimizzazione delle interrogazioni
  - controllo della concorrenza
  - gestione del ripristino



## Ottimizzazione di interrogazioni

- è necessaria quando un DBMS riceve una richiesta distribuita
- il DBMS interrogato è responsabile dell'ottimizzazione globale
  - decide come suddividere l'interrogazione in sottointerrogazioni, ciascuna rivolta ad uno specifico DBMS
  - costruisce un piano di esecuzione distribuito, che consiste nell'esecuzione coordinata di vari programmi in vari DBMS e nello scambio di dati tra di essi
- il costo di esecuzione di un'interrogazione distribuita include il costo di trasmissione dei dati sulla rete



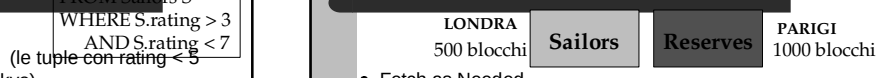
## Ottimizzazione di interrogazioni

```
SELECT AVG(S.age)
FROM Sailors S
WHERE S.rating > 3
      AND S.rating < 7
```

- Esempio:
- frammentazione orizzontale: (le tuple con rating < 5 a Shanghai, quelle >= 5 aTokyo)
  - bisogna calcolare SUM(age), COUNT(age) in entrambi i siti
  - se la clausola WHERE contenesse solo S.rating>6, avremmo potuto risolvere l'interrogazione in un unico sito
- frammentazione verticale: (*sid* e *rating* a Shanghai, *sname* e *age* a Tokyo, *tid* in entrambe)
  - bisogna ricostruire la relazione con un join su *tid*, e poi valutare la query
- replicazione: copie di Sailors in entrambi i siti
  - si sceglie il sito basandosi sui costi locali e i costi di trasmissione dati



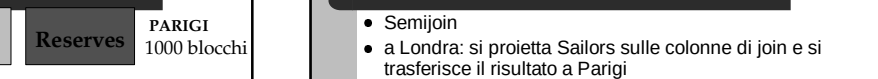
## Ottimizzazione di interrogazioni Join distribuiti




- Fetch as Needed
  - Iterazione orientata ai blocchi, Sailors come outer:
  - **Costo:** 500 D + 500 * 1000 (D+S) [no caching]
  - **D** è il costo di leggere/scrivere un blocco; **S** è il costo di trasferire un blocco
  - se la query non fosse richiesta a Londra ci sarebbe il costo addizionale di inviare il risultato al sito della query
  - in alternativa il join può essere effettuato sempre a Londra, e sempre trasferendo a Londra ogni tupla richiesta quando necessario, utilizzando INL (indice) invece che BNL
  - in entrambi i casi il costo di trasferimento delle tuple domina il costo totale anche con una rete veloce



## Ottimizzazione di interrogazioni Join distribuiti




- Ship to One Site:
  - ad esempio si trasferisce tutta la relazione Reserves a Londra
  - costo: 1000 (2D +S) + 1500 D (utilizzando merge-join costo = 500+1000)
  - alternativamente si trasferisce tutta la relazione Sailors a Parigi
  - costo: 500 (2D +S) + 1500 D (utilizzando merge-join costo = 500+1000)
  - se la dimensione del risultato è grande, può essere meglio trasferire entrambe le relazioni al sito in cui è stata richiesta la query ed effettuare lì il join



## Ottimizzazione di interrogazioni - Join distribuiti

- Semijoin
- a Londra: si proietta Sailors sulle colonne di join e si trasferisce il risultato a Parigi
- a Parigi: si effettua il join della proiezione di Sailors con Reserves (in questo caso solo *sid*)
  - il risultato è chiamato una **riduzione** di Reserves rispetto a Sailors
- si trasferisce la riduzione di Reserves a Londra
- a Londra: si effettua il join di Sailors con la riduzione di Reserves
- tradeoff tra il costo aggiuntivo di calcolare e trasferire la proiezione e il costo di trasferire l'intera relazione Reserves
- specialmente utile se c'è una selezione su Sailors e il risultato serve a Londra

## Ottimizzazione di interrogazioni - Join distribuiti

- Bloomjoin
- a Londra: calcola un bit-vector di dimensione k:
  - si effettua l'hash dei valori della colonna nel range 0 ... k-1
  - se qualche tupla finisce nella posizione i, assegna il bit i a 1 (i=0…k-1)
  - si trasferisce il bit vector a Parigi
- a Parigi: si applica la stessa funzione hash a ogni tupla di Reserves e si elimina le tuple che finiscono in una posizione del Sailors bit-vector contenente 0
  - il risultato è chiamato la **riduzione** di Reserves rispetto a Sailors
- si trasferisce la riduzione di Reserves a Londra
- a Londra: si effettua il join di Sailors con la riduzione di Reserves
- il bit-vector è più conveniente da trasferire e quasi altrettanto efficace



## Ottimizzazione di interrogazioni

- approccio basato sui costi: si considerano tutti i piani e si sceglie il più economico (come nel caso centralizzato)
- differenze:
  - bisogna considerare i costi di comunicazione
  - bisogna rispettare l'autonomia locale dei siti
  - nuovi metodi di join distribuiti (semijoin, bloomjoin)
- il sito a cui è rivolta l'interrogazione costruisce un piano globale, che comprende dei piani locali suggeriti ad ogni sito

  se un sito può migliorare il piano locale che gli viene suggerito è libero di farlo



## Controllo della concorrenza

- la serializzabilità locale non garantisce la serializzabilità
- esempio (il primo pedice denota la transazione, il secondo pedice denota il sito)

  S1: $r_{11}(x)$ $w_{11}(x)$ $r_{21}(x)$ $w_{21}(x)$
  S2: $r_{22}(y)$ $w_{22}(y)$ $r_{12}(y)$ $w_{12}(y)$

- la serializzabilità globale di un insieme di transazioni distribuite sui nodi di una base di dati distribuita richiede l'esistenza di un unico schedule seriale S equivalente a tutti gli schedule locali $S_i$ prodotti ad ogni nodo



## Controllo della concorrenza

- se ogni scheduler di una base di dati distribuita usa il metodo di locking a due fasi in ogni nodo e completa l'azione di commit quando tutte le sotto-transazioni hanno acquisito tutto le risorse, gli schedule risultanti sono globalmente serializzabili
- se ogni transazione distribuita acquisisce un singolo timestamp e usa tale timestamp in tutte le richieste a tutti gli scheduler che usano un protocollo di controllo della concorrenza basato su timestamp, gli schedule risultanti sono globalmente serializzabili, basati sull'ordine imposto dai timestamp
- metodo di Lamport per assegnare timestamp in ambito distribuito
- metodi di risoluzione di deadlock in ambito distribuito



## Guasti nei sistemi distribuiti

- **guasti dei nodi** possono verificarsi in ogni nodo del sistema
- **perdite di messaggi** lasciano l'esecuzione di un protocollo in uno stato di incertezza
  - ogni messaggio del protocollo (msg) è seguito da un messaggio di conferma (ack)
  - la perdita di uno dei due messaggi lascia il ricevente incerto circa il ricevimento del messaggio
- **partizionamento della rete** è un guasto del collegamento di comunicazione della rete che la suddivide in due sottoreti senza possibilità di comunicazione
  - una transazione può essere simultaneamente attiva in più di una sotto-rete



## Protocollo di commit a due fasi

- i protocolli di commit permettono ad una transazione di raggiungere la decisione corretta relativamente al commit o all'abort in tutti i nodi che partecipano ad una transazione
- il protocollo di commit a due fasi può essere pensato come un matrimonio, in quanto la decisione delle due parti è ricevuta e registrata da una terza parte, che la ratifica
  - i server (che rappresentano i partecipanti al matrimonio) sono detti gestori delle risorse (RM)
  - il coordinatore (celebrante) è un processo, detto transaction manager (TM)

## Commit a due fasi

- il protocollo consiste in un rapido scambio di messaggi tra TM e RM e in scritture di record nei loro log
- il TM può utilizzare
  - meccanismi di broadcast, per trasmettere lo stesso messaggio a molti nodi, raccogliendo poi le risposte che arrivano dai vari nodi
  - comunicazione seriale con uno dei RM per volta
- record del TM
  - il record **prepare** contiene l'identità di tutti i processi RM (id di nodo e processo)
  - il record **global commit** o **global abort** descrive la decisione globale (quando il TM scrive tale record nel suo log raggiunge la decisione finale)
  - il record **complete** viene scritto alla fine del protocollo di commit a due fasi



## Commit a due fasi

- record del RM
  - il record **ready** indica la disponibilità irrevocabile a partecipare al protocollo
  - può essere scritto solo quando il RM è in uno stato affidabile, cioè possiede tutti i lock sulle risorse che devono essere scritte
    - il record deve contenere anche l'id del TM
    - si hanno inoltre i record classici del caso centralizzato
- in ogni momento un RM può autonomamente abortire una sotto-transazione, disfandone gli effetti, senza partecipare al protocollo di commit a due fasi



## Commit a due fasi - fase 1

- il TM scrive il record **prepare** nel suo log e manda un messaggio **prepare** a tutti i RM e setta un timeout che indica il massimo tempo allocato per il completamento della prima fase
- i RM ripristinabili scrivono sul loro log un record **ready** e mandano al TM un messaggio **ready**, che indica la volontà di partecipare al commit
- i RM non ripristinabili mandano un messaggio **not-ready** e terminano il protocollo
- il TM raccoglie i messaggi di risposta da parte dei RM:
  - se riceve risposte positive da tutti i RM, scrive un record **global commit** sul suo log
  - se riceve una o più risposte negative, o se il time-out scade senza che il TM abbia ricevuto tutte le risposte, scrive un record **global abort** sul suo log

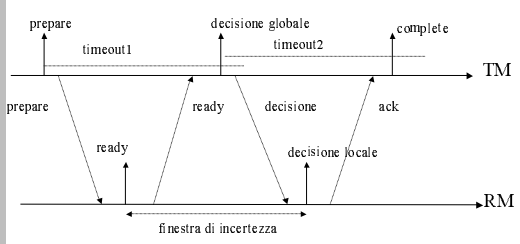


## Commit a due fasi - fase 2

- il TM trasmette la sua decisione globale ai RM e setta un secondo timeout
- i RM che sono pronti a ricevere il messaggio di decisione, scrivono i record **commit** o **abort** nel loro log e mandano una conferma (**ack**) al TM
- implementano poi il commit o l'abort scrivendo le pagine nella base di dati come nel caso centralizzato
- il TM raccoglie tutti i messaggi di **ack** dai RM coinvolti nella seconda fase
- se il timeout scade setta un altro timeout e ripete la trasmissione a tutti i RM da cui non ha ricevuto un **ack**
- quando riceve tutti gli **ack**, il TM scrive il record **complete** sul suo log

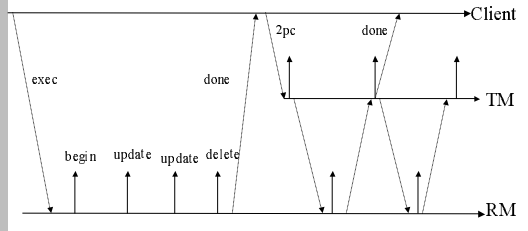


## Commit a due fasi






## Commit a due fasi



## Commit a due fasi

- un RM in stato **ready** perde la sua autonomia e aspetta la decisione del TM
- un guasto del TM lascia il RM in uno stato di incertezza
- le risorse acquisite utilizzando lock sono bloccate
- la finestra compresa tra la scrittura sul log del RM del record **ready** e la scrittura del record **commit** o **abort** è chiamata *finestra di incertezza*
- il protocollo deve mantenere tale intervallo al minimo
- il TM o il RM eseguono protocolli di ripristino dopo eventuali guasti
- lo stato finale ripristinato dipende dalla decisione globale del TM



## Ripristino dei partecipanti

- dipende dall'ultimo record scritto nel log
  - se è un'azione o un record **abort**, le azioni vengono disfatte (*undo*)
  - se è un record **commit**, le azioni sono rieseguite (*redo*)
- in entrambi i casi il guasto si è verificato prima dell'inizio del protocollo di commit
- se è un record **ready**, il guasto si è verificato durante il 2PC, e il partecipante è in dubbio circa l'esito della transazione



## Ripristino dei partecipanti

- durante il protocollo di ripristino, gli identificatori delle transazioni in dubbio sono raccolti in un insieme (ready set)
- per ognuna di esse l'esito finale della transazione deve essere richiesto al TM
- questo può succedere sia come risultato di una richiesta diretta (ripristino remoto) da parte del RM o come una ripetizione della seconda fase del protocollo



## Ripristino del coordinatore

- quando l'ultimo record nel log è un record **prepare**, il fallimento del TM può aver lasciato qualche RM in una situazione di blocco
- ci sono due opzioni di ripristino possibili:
  - scrivere **global abort** sul log, e procedere con la seconda fase del protocollo
  - ripetere la prima fase, cercando di arrivare a un commit globale
- quando l'ultimo record nel log è una decisione globale, alcuni RM possono essere stati correttamente informati della decisione ed altri possono essere stati lasciati in uno stato bloccato, il TM deve quindi ripetere la seconda fase



## Perdita di messaggi e partizionamento della rete

- la perdita di un messaggio **prepare** non è distinguibile dal TM dalla perdita di un messaggio **ready**
- in entrambi i casi il timeout della prima fase scade e viene presa la decisione globale di abort
- allo stesso modo, la perdita di un messaggio di decisione non è distinguibile dalla perdita di un messaggio di **ack**
- in entrambi i casi il timeout della seconda fase scade e la seconda fase viene ripetuta
- un partizionamento della rete non causa problemi ulteriori, perchè la transazione verrà completata con successo solo se il TM e tutti i RM appartengono alla stessa partizione



## Ottimizzazioni del 2PC

- il protocollo visto è abbastanza costoso, in particolare assume che tutte le scritture sul log siano sincrone in modo da garantirne la persistenza
- esistono varianti del protocollo che evitano la scrittura sincrona di alcuni record nel log
- protocolli di abort presunto (adottato dalla maggioranza dei DBMS commerciali) e di commit presunto
- nel protocollo di abort presunto quando un TM riceve una richiesta di ripristino remoto da parte di un partecipante in dubbio sulla cui transazione il TM non ha informazioni, viene restituita la decisione di abort

## Ottimizzazioni del 2PC

- non sono più necessarie le scritture sincrone dei record di **prepare** e **global abort**, perchè nel caso fossero perse il comportamento di default sarebbe lo stesso
- il record complete inoltre non è più critico per l'algoritmo, e in alcuni sistemi è omesso
- i record da scrivere in maniera sincrona quindi sono **ready**, **global commit** e **commit**
- un'altra ottimizzazione è relativa ai partecipanti read-only
  - rispondono con un messaggio **read-only** al messaggio **prepare** e sospendono l'esecuzione del protocollo
  - vengono ignorati nella seconda fase del protocollo