

Gestore del ripristino / 13-05

Durability - effetti delle transazioni committed devono essere persistenti
Atomicity - gli effetti delle transazioni aborted non devono lasciare tracce

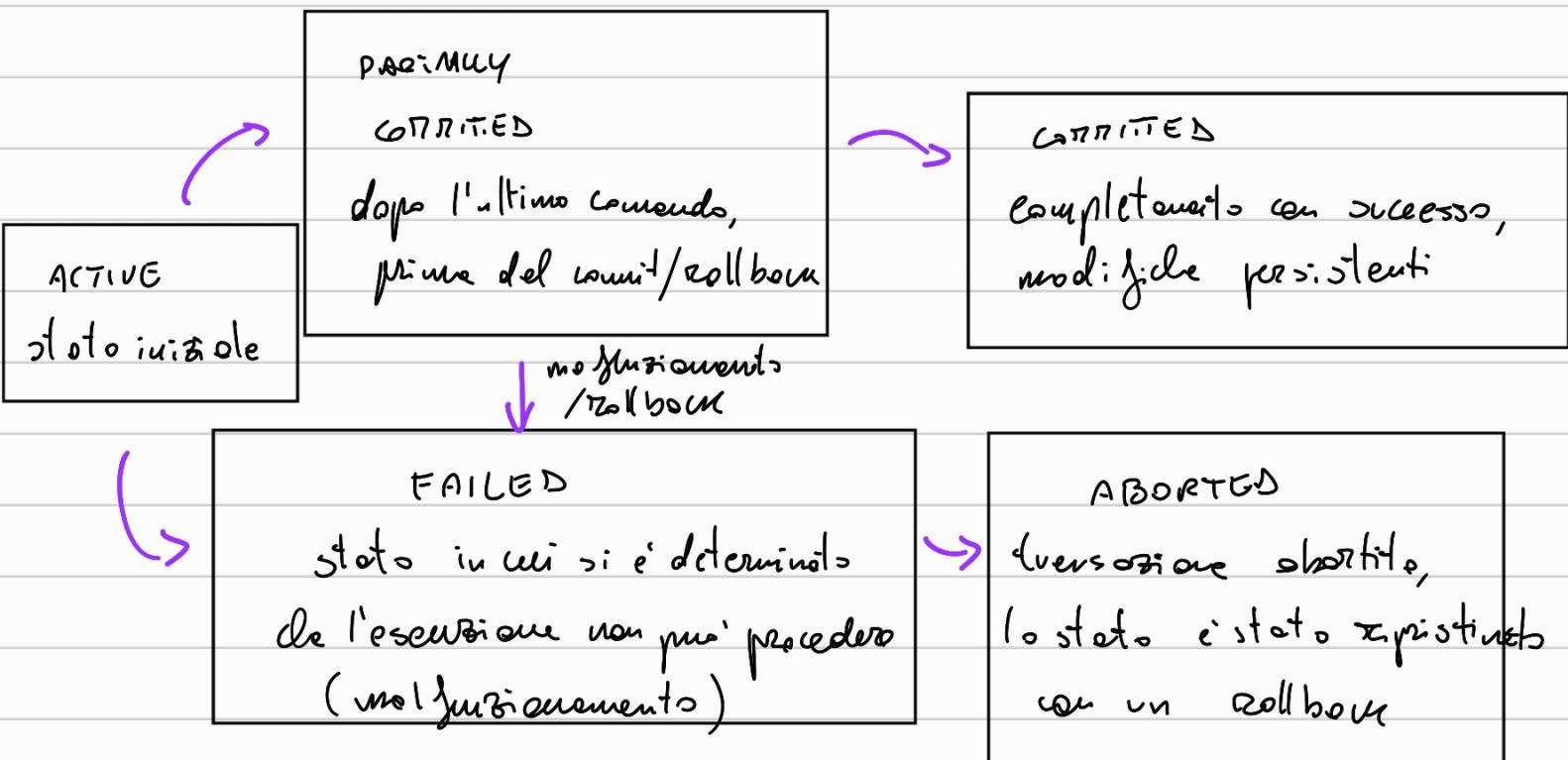
/ \
ROLLBACK GHOSTS

Se ne occupa il gestore del ripristino

Moluzionamenti

- transaction failure: abortisce per sue scelte, fa rollback
- system failure: questo risulta che interrompe le transazioni, impatta la memoria volatile, non i dischi
- media failure: problemi sui dischi (memorie non volatili)

Modello di esecuzione delle transazioni



Dopo il rollback:

- la transazione puo' essere ri-eseguita
in caso di sys failure o media failure
- eliminata
in caso di errori nelle logica stessa della transazione
(no risutte), ha transaction failure

Note: alcuni dati ormai scritti, se ri-eseguo la transazione, non
possono essere ripristinati, poche' persistenti sul disco
(DB modificato prima di seguire da la transazione obbligata
successo)

Per ovviare:

- in caso di transaction o sys failure, le ottinte di ripristino
sono dette "riprresa e caldo"

Tutte le operazioni eseguite durante la transazione vengono
registerate in un file di log memorizzato su memoria
stabile (non viene influenzato dai guasti trans e sys failure)

↳ e' una storia, non "ostile" in sé'

- tecnicamente mai coinvolta in failures
- viene implementata con approssimazioni, il log viene
solvolto su più memorie con prob. di fallimenti
indipendenti
↳ non volatili

Il file di log e' un file (oppure posso aver più file) con scritte
semprezioli (append), contiene un record con info minimali
di modifica a ogni blocco/pagina utilizzata dalla transazione

Il record contiene:

- numero progressivo (LSN)
- id della transazione (T)
- id della pagina modificata (indirizzo) (PID)
- dato una pagina P , il contenuto precedente di P alla modifica ($before(P)$)
- " dopo la modifica di: P ($after(P)$)
- numero progressivo precedente (linked list) relativo allo stesso T nel log (prev LSN)

non salva il tipo di operazione

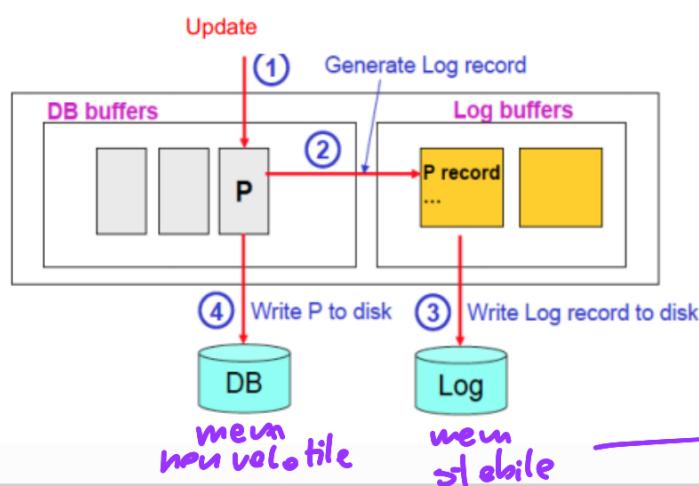
(update etc...)

+ il log contiene anche le info che specificano un BEGIN o COMMIT / ROLLBACK

Protocollo WAL - Write Ahead Logging

Garantisce l'isolabilità se applicato: prima di scrivere su un disco una pagina P modificata, il corrispondente record di log deve essere già stato aggiornato.

La persistenza è garantita quando presso a un solo commit, solo dopo che tutti i record di log siano scritti su memoria stabile



mem
volatile

(estrazione)

Il log tiene traccia di tutte le operazioni di modifica, indipendentemente dal loro tipo

Stato corrente del DB = risultati di trans. committed

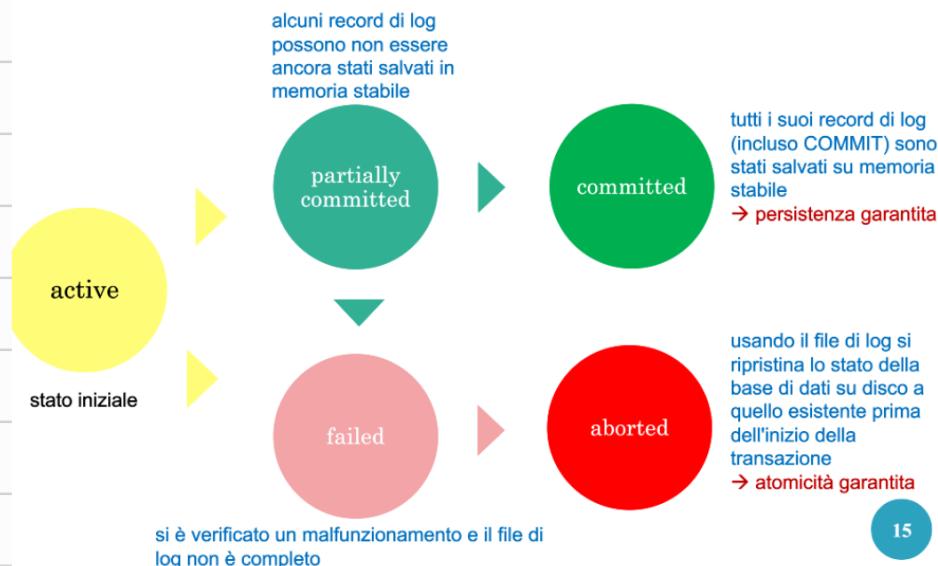
11

qui ho solo i risultati ← stato corrente del DB sul disco
già resi persistenti,
oltre non avranno
ancora fatto commit
(o magari devono ancora
fare rollback)

Il rispetto del WAL è garantito dal buffer manager.

Il momento in cui effettuare le modifiche (punto ④) determina
tempi strettamente differenti. Quando eseguire ④ quando si è nello stato
committed?

PROTOCOLLO WAL E MODELLO ASTRATTO DI ESECUZIONE



15

Quando aggiornare lo stato del DB su disco
finendo e' stato modificato nel buffer?

Protocollo (2)- (6)
log non influenzano
le politiche

Si seguono 2 politiche:

- politica **SICAL**: oppure possibile aggiornare il disco dopo averlo modificato nel buffer
(non ancora commit)

→ se scrivo su disco e poi T abortisce, devo nuovamente ricevere ripristinando i dati su disco
→ fa uno innesco di questi

- politica **NO STEAL**: ritarda la modifica su disco finché
(non ancora commit) la transazione non termina o quando più tardi

→ **peggiore** la gestione del buffer: rischio di esaurire lo spazio in memoria centrale (che comporta perdita di scrittura su disco necessarie)

Quando eseguire ① una volta giunto nello stato committed?

2 politiche:

- politica **FORCE**: prima di scrivere nel log il record GPRIT, scrivo: Record di tutte le pagine modificate

poi si scrive il log commit

entro in committed le modifiche sono persistenti

→ molti I/O inutili: se una pagina è scritta 1000 volte da diverse transazioni ho molti accessi a disco
→ non fanno se ha un questo in corrispondenza

• politice **No Force** : il concorso di force, ^{sudisco} occurs dopo il commit ou log

→ migliora gli accessi a disco, ma prende entro nello stato committed due modi fido perche non essere ancora scritte

→ le REDO di T, ri-esegue le op. se sono in committed e ha un guasto

IN SINTESI

- Se T al momento del guasto **è nello stato committed**

- Force → niente da fare
- No Force → **REDO**

- Se T al momento del guasto **non è nello stato committed**

- Steal → **UNDO**
- No Steal → niente da fare

posso avere solo sys failure
P (la logica delle trans. el commit è finale)

sono nello stato di complemento:

scrivute già eseguite, non devo ripeterle

(transaction failure si comporta = sys failure)
in uno stato intermedio

sono in uno stato intermedio

→ scrivute non eseguite, non devo ripristinare

→ Force meno usata x costi

→ Steal più utilizzata per ottenere l'uso del buffer

Si usa STEAL - NO FORCE, costoso se ha tante transazioni, sono lunghe (ha redo e undo)

REDO e UNDO devono essere **idempotenti**;

se ha un guasto durante REDO o UNDO che quindi vanno rifatte, deve eseguire REDO quando REDO era in esecuzione

e uno grande (o zero uno)

Per migliorare le prestazioni si usano dei **checkpoint**

- il sistema periodicamente fa la scrittura delle pagine di log in memoria stabile e delle pagine di dati nel disco
- se ho sys failure prendo l'ultimo commit, non devo fare niente di tutto ma solo dei record successivi ad esso (migliora il recupero, non l'uso)
riduce l'overhead delle politiche no-force
(disponibile dopo)

Caso di media failure - errore nel disco

Riposo a freddo: si utilizza log + **dump**

dump: backup completo del DB in memoria stabile

Il suo operazione si registra nel log

In caso di guasto

- si accede, o ritrova nel log, al dump per ripristinare il DB
- si effettua poi una ripresa a caldo "per rifare le operazioni della versione del dump sino al momento prima del guasto" guardando il log

