Tecnologia di un Database Server (centralizzato)

Gestione del buffer

Angelo Montanari

Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università di Udine

Introduzione

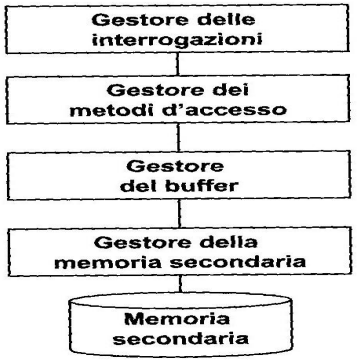
Il **buffer** è una (vasta) zona di memoria centrale preallocata al DBMS e condivisa fra le varie transazioni.

Linea di tendenza: dato l'abbattimento dei costi delle memorie, vengono allocati ai DBMS buffer di dimensione sempre maggiore.

In alcuni casi, l'intera base di dati può essere copiata e gestita in memoria centrale (in verità, l'aumento delle dimensioni dei buffer è stato accompagnato da un aumento delle dimensioni delle basi di dati).

Componenti di un DBMS - 1

I **componenti** di un DBMS coinvolti nei processi di gestione delle interrogazioni e di accesso alla memoria secondaria sono riassunti nella seguente figura:



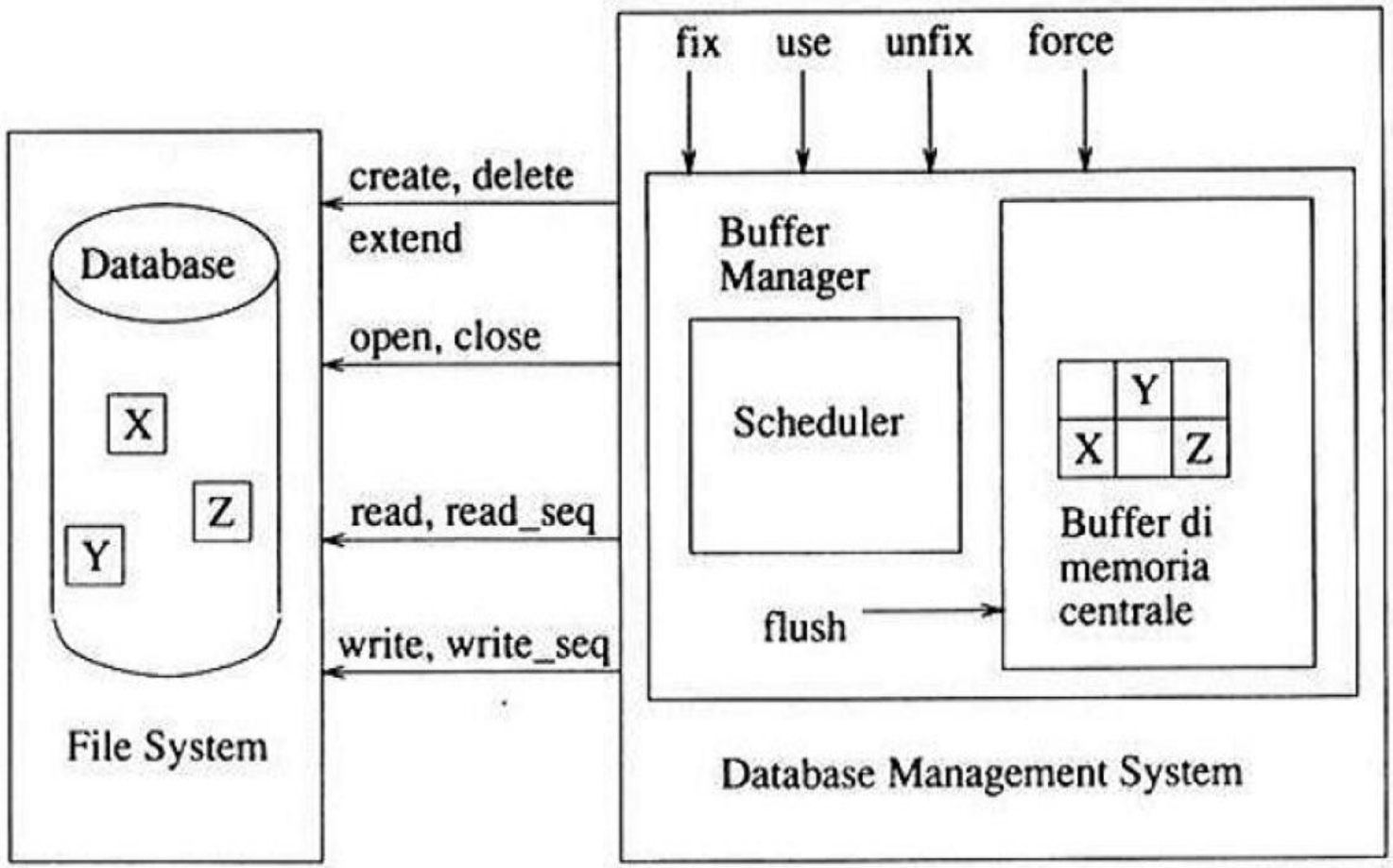
Il **gestore delle interrogazioni** esegue un'ottimizzazione delle interrogazioni: traduce le interrogazioni in un formato interno e le trasforma in modo da consentirne un'esecuzione efficiente.

Componenti di un DBMS - 2

Le operazioni richieste vengono formulate in termini di operazioni elementari, quali scansioni, ordinamenti e accessi diretti ai dati, che fanno riferimento all'organizzazione fisica del file. Tali operazioni vengono eseguite dal gestore dei metodi di accesso. Il **gestore dei metodi di accesso** trasforma le richieste in operazioni di lettura e scrittura di dati in memoria secondaria. Tali richieste vengono mediate dal gestore del buffer.

Gestore del buffer (buffer manager): ha la responsabilità di mantenere temporaneamente porzioni della base di dati in memoria centrale, per aumentare l'efficienza del DBMS e garantire nel contempo l'affidabilità. Il gestore del buffer invia al **gestore della memoria secondaria** le effettive richieste di lettura e scrittura fisica.

Architettura del gestore del buffer



Compiti del gestore del buffer

I compiti del gestore del buffer sono:

- **caricamento/scaricamento** delle pagine dalla memoria secondaria alla memoria principale/dalla memoria principale alla memoria secondaria;
- **accesso** alle pagine presenti nel buffer.

Per il controllo della concorrenza, il gestore del buffer interagisce con lo scheduler (usualmente il lock manager).

Organizzazione del buffer

Il **buffer** è organizzato in pagine che hanno dimensione pari a uno o più blocchi di memoria secondaria.

Il blocco è l'unità di riferimento per le operazioni di ingresso/uscita effettuate dal sistema operativo per leggere/scrivere blocchi dai dispositivi di memoria secondaria; le dimensioni del blocco possono variare da alcuni Kbyte ad alcune decine di Kbyte.

Per semplicità assumiamo che le pagine del buffer abbiano dimensione pari a quella del blocco.

Gestione delle letture e delle scritture

La **tempistica** delle letture/scritture sulla base di dati da parte del gestore del buffer non necessariamente coincide con quella delle corrispondenti richieste:

- in caso di **lettura**, se la pagina/blocco è già presente nel buffer, non è necessario effettuare la lettura fisica (i tempi di accesso alle pagine/blocchi di memoria secondaria e alle pagine/blocchi di memoria principale differiscono di vari ordini di grandezza);
- in caso di **scrittura**, il gestore del buffer può decidere di ritardare la scrittura fisica, qualora consentito dalle proprietà di affidabilità del sistema.

In entrambi i casi, l'obiettivo è quello di ridurre i tempi di risposta alle richieste di un'applicazione.

Politiche di gestione del buffer

Le **politiche di gestione** del buffer sono analoghe a quelle di gestione della memoria centrale da parte del sistema operativo e rispondono al medesimo principio di località dei dati:

i dati acceduti di recente hanno maggiore probabilità di essere nuovamente acceduti nel futuro prossimo

In aggiunta, anche in tale ambito sembra valere una variante del noto principio di Pareto (l'80% degli effetti ẁ prodotto dal 20% delle cause): il 20% dei dati è acceduto dall'80% delle applicazioni. Conseguenza: i buffer contengono la maggior parte delle pagine accedute dalle applicazioni.

Direttorio e variabili di stato

Per la gestione del buffer, il buffer manager mantiene le seguenti informazioni/strutture dati:

- un **direttorio** che descrive il contenuto corrente del buffer, indicando per ciascuna pagina caricata il file fisico e il numero di blocco corrispondenti;
- per ogni pagina del buffer, due **variabili di stato** (un contatore, che specifica quanti sono i programmi/transazioni che utilizzano la pagina, e un bit di stato, che specifica se la pagina è stata o meno modificata – le modifiche devono essere prima o poi riportate in memoria secondaria).

Osservazione: al crescere della dimensione del buffer cresce l'importanza di gestire in modo efficiente il direttorio.

Primitive per la gestione del buffer

Il buffer manager mette a disposizione delle transazioni un insieme di primitive per il caricamento e lo scaricamento delle pagine:

1. La primitiva **fix**. Viene utilizzata per richiedere l'accesso ad una pagina e restituisce al chiamante il riferimento alla pagina del buffer, in modo da consentire l'effettivo accesso ai dati.
2. La primitiva **setDirty**. Indica al buffer manager che una pagina ѐstata modificata. L'effetto è la modifica del relativo bit di stato.
3. La primitiva **unfix**. Indica al buffer manager che il chiamante ha terminato di utilizzare la pagina. Ha l'effetto di decrementare il contatore di utilizzo della pagina.
4. La primitiva **force**. Trascrive in memoria secondaria, in modo sincrono, una pagina del buffer (e aggiorna il corrispondente bit di stato). Tale operazione è richiesta dal gestore dell'affidabilità quando risulta necessario garantire che alcuni dati non vengano persi.

Esecuzione della primitiva fix - 1

L'**esecuzione** della fix è effettuata nel modo seguente:

(i) Si cerca la pagina fra quelle già presenti nel buffer (pagine caricate in precedenza). Se trovata, l'operazione termina immediatamente con successo. Per il principio di località, ciò avviene abbastanza spesso.

(ii) Se non è presente, si cerca nel buffer una pagina libera (valore del contatore uguale a 0). Se il bit di stato segnala che è stata modificata (e non ancora salvata), viene aggiornata in memoria secondaria (operazione asincrona di **flush**). Vengono, quindi, operate le necessarie conversioni di indirizzi per identificare la pagina da caricare nel buffer ed viene effettuata l'operazione di lettura

Esecuzione della primitiva fix - 2

(iii) Se non esistono pagine libere, due politiche alternative sono possibili:

- Politica **steal**. Consente di sottrarre una pagina ad un'altra transazione. La pagina selezionata, detta vittima, viene scaricata in memoria di massa (tramite un'operazione asincrona di flush). Vengono, quindi, operate le necessarie conversioni di indirizzi e viene effettuata l'operazione di lettura.
- Politica **no-steal**. Non consente di sottrarre pagine alle transazioni attive. La transazione viene pertanto sospesa ed entra in una coda di transazioni gestita dal buffer manager. Quando si libera una pagine, il buffer manager procede come al punto (ii).

Esecuzione della primitiva fix - 3

(iv) In tutti i casi considerati, quando si effettua un accesso ad una pagina, viene incrementato il contatore relativo agli utilizzatori della pagina.

Politiche force/no-force.

La scrittura in memoria secondaria può avvenire in modo sincrono, all'interno della transazione che la richiede, attraverso la primitiva force (politica **force**) oppure in modo asincrono (indipendente dall'applicazione) sulla base delle decisioni del buffer manager che tengono conto della necessità di recuperare spazio e dei criteri di ottimizzazione (politica **no-force**).

Pre-fetching e pre-flushing

Esiste la possibilità di anticipare i tempi di caricamento e di scaricamento delle pagine.

Pre-fetching. Caricamento delle pagine anticipato rispetto alle richieste delle transazioni, in quei casi in cui sono note a priori le modalità di accesso alle pagine della base di dati da parte di una transazione.

Pre-flushing. Scaricamento anticipato delle pagine rispetto al momento in cui vengono scelte come vittime: scrittura anticipata di pagine rese libere dall'esecuzione di un'operazione di unfix, che sono state modificate nel corso del loro utilizzo (bit di stato con valore *dirty*). L'esecuzione del pre-flushing rende più efficienti le successive operazioni di fix.

Osservazione: una pagina utilizzata da molte applicazioni può restare a lungo nel buffer, subendo varie modifiche, e venire trascritta in memoria secondaria con una sola operazione di scrittura.

DBMS e file system

L'interazione tra il DBMS (buffer manager) e il sistema operativo (file system) **non è banale**.

Il DBMS usa alcune (poche) **funzionalità** del **file system**, creando, però, una propria **astrazione** dei file, che consentono di garantire **efficienza** (tramite l'uso del buffer e una gestione di basso livello delle strutture fisiche) e **transazionalità** (attraverso il gestore dell'affidabilità, che assicura le proprietà di atomicità e persistenza).

Prospettiva futura: far migrare tali funzionalità dei DBMS al di fuori di essi, ad esempio, incorporandole all'interno del sistema operativo.

Le primitive del file system usate dai DBMS - 1

I DBMS usano alcune (poche) funzionalità del sistema operativo per creare/eliminare file e per leggere/scrivere singoli blocchi o sequenze di blocci contigui. La struttura dei file, sia all'interno dei singoli blocchi sia nell'organizzazione dei dati a partire dai blocchi, è, invece, gestita direttamente dal DBMS.

Creazione (`create`) ed **eliminazione** (`delete`) di un file. All'atto della creazione viene assegnato al file un numero iniziale minimo di blocchi, che può essere esteso dinamicamente (`extend`).

Apertura (`open`) e **chiusura** (`close`) di un file. Caricano informazioni che descrivono in file in opportune strutture della memoria centrale. L'apertura di un file di norma associa un identificatore numerico (`fileid`) al nome del file (`filename`).

Le primitive del file system usate dai DBMS - 2

Accesso diretto ad un **blocco** di un file: primitiva `read(fileid,block,buffer)`. I primi due parametri individuano il blocco da leggere, il terzo la pagina del buffer ove trascriverlo.

Accesso sequenziale ad un **numero fisso di blocchi** di un file: primitiva `read_seq(fileid,f-block,count,f-buffer)`. Il primo parametro identifica il file, il secondo il primo blocco della sequenza, il terzo il numero di blocchi, l'ultimo la prima pagina del buffer ove copiare la sequenza.

Le corrispondenti primitive per la scrittura sono `write(fileid,block,buffer)` e `write_seq(fileid,f-block,count,f-buffer)`, con l'ovvio significato.