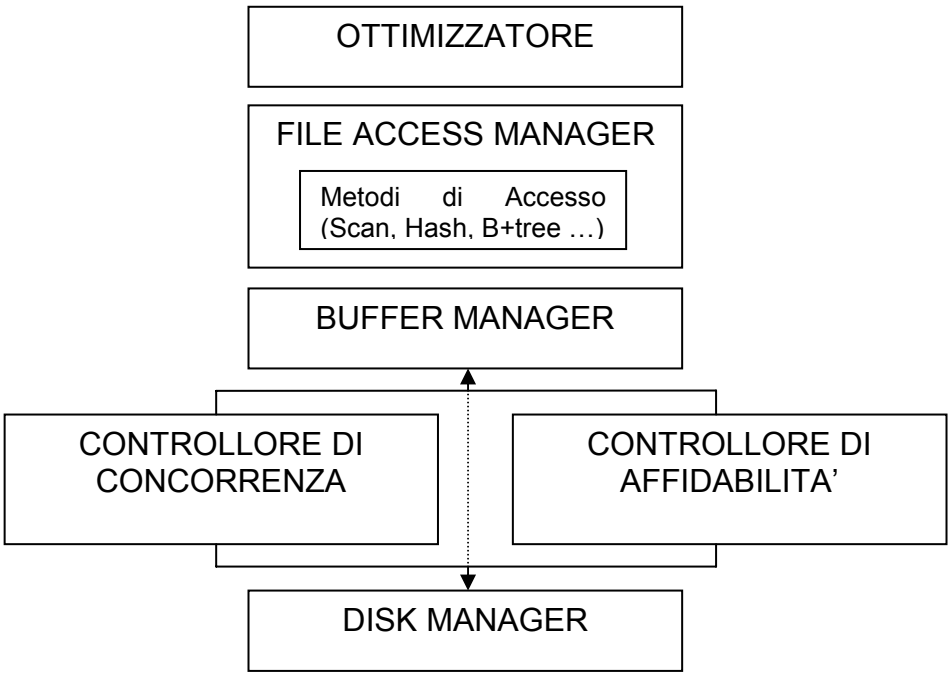
ARCHITETTURA DI UN B.D.M.S.

Parte II – Il Buffer Manager

Michele de Nittis

Generalità

Consideriamo nuovamente la struttura modulare astratta di un DBMS:



In generale la memoria centrale dei DBMS (veloce, costosa e volatile) è disponibile in quantità molto limitata rispetto alla memoria secondaria (lenta, permanente ed economica), ma, data la lentezza di quest'ultima, tutte le operazioni sui dati devono essere effettuate su quella centrale, molto veloce. Sono, pertanto, necessari degli accorgimenti per gestire il trasferimento dei dati dalla memoria di massa a quelle centrale, e viceversa, in modo che le prestazioni generali del sistema non degradino.

Il **Buffer Manager** è quel modulo che gestisce la memoria centrale ed il trasferimento dei dati da e verso la memoria di massa. Esso si avvale, tra le altre cose, dei servizi offerti dal controllo di concorrenza e dal controllo di affidabilità. Dato che la memoria di massa è disponibile in quantità maggiori rispetto alla memoria centrale, il BM deve **decidere** quali dati, tra quelli già memorizzati in quest'ultima, sono sacrificabili per fare posto ai nuovi. Tale decisione, come vedremo, è presa in base ad una politica denominata **politica di rimpiazzo** (*Replacement Policy*).

Il Buffer Pool

La parte di memoria centrale gestita dal DBMS è suddivisa in **pagine** (o *frame*), cioè in blocchi di memoria di dimensione fissa pari a quella usata per il trasferimento dei dati da e verso la memoria disco in un'unica operazione di I/O. L'insieme delle suddette pagine è denominato **Buffer Pool** e viene gestito da un particolare servizio, denominato

Buffer Manager, che mette a disposizione dei servizi di livello superiore delle primitive per richiedere il caricamento o lo scaricamento delle pagine di memoria.

Per svolgere le proprie funzioni di gestione, il BM dispone di molte variabili interne, tra le quali ve ne sono due per ogni pagina del buffer pool:

Pin Count: indica il numero corrente di transazioni che impiegano la pagina;

Dirty: è un *flag* che indica se una pagina è stata modificata da quando è stata caricata nel buffer o dall'ultima volta che è stata scaricata in memoria secondaria.

Le Primitive del Buffer Manager

Le primitive messe a disposizione dal BM sono:

fix: usata per richiedere l'accesso ad una pagina di memoria secondaria e caricarla nel buffer, se non già disponibile. Restituisce alla transazione che l'ha invocata un riferimento alla pagina richiesta;

use: usata da una transazione per indicare al BM l'accesso ad una pagina precedentemente caricata in memoria, confermandone lo stato di **pagina valida o attiva**;

ufix: usata da una transazione per indicare al BM che ha terminato di impiegare una data pagina (non è detto, tuttavia, che la pagina passi nello stato di **pagina non attiva**);

force: usata da una transazione per richiedere (o forzare) il trasferimento di una pagina in memoria di massa. La transazione richiedente rimane sospesa fino al termine dell'operazione. Questo tipo di trasferimento è detto **sincrono**.

Flush: è una primitiva ad uso interno del BM ed è impiegata per il trasferimento in memoria di massa di una o più pagine del buffer, indipendentemente dalle transazioni attive e dalle richieste eventualmente formulate. Questo tipo di trasferimento viene detto **asincrono**.

Vediamo come opera il BM quando esegue la direttiva **fix**:

Cerca la pagina richiesta nel Buffer e se la trova ne restituisce il riferimento, altrimenti cerca una pagina libera o non attiva (**Pin Count=0**) nel buffer pool (*vittima*), ne trasferisce eventualmente il contenuto in modo asincrono (*flush()*) in memoria secondaria se modificata (**Dirty=true**), carica al suo posto la pagina richiesta, prelevandola dalla memoria secondaria, e ne restituisce il riferimento. La scelta della pagina libera da riempire, in caso ve ne fossero diverse, viene fatta in base alla **politica di sostituzione.** Nel caso non fosse disponibile una pagina libera, il BM può decidere, in base alla **politica di gestione** adottata, di sacrificarne una scegliendola tra quelle attive oppure di sospendere o terminare la richiesta. Una volta caricata la pagina, il BM incrementa la variabile **Pin_Count** (pinning).

Quando esegue la direttiva **Use** su una pagina, il BM incrementa la variabile **Pin_Count** (pinning) e, se la transazione richiedente compie una modifica, pone il flag Dirty al valore 'true'. Viceversa quando il BM esegue la direttiva **unfix** su una pagina, decrementa la relativa variabile Pin_Count (unpinning).

Politica di gestione del Buffer Manager

Il Buffer Manager può gestire il *buffer pool* secondo due coppie alternative di politiche:

1. **Steal**: in caso d'indisponibilità di una pagina libera, il BM sacrifica una pagina attiva, ripristinandone lo stato precedente all'inizio le transazioni ancora attive che vi hanno avuto accesso e che dovranno abortire;
2. **No Steal**: è la politica duale alla precedente: esclude la possibilità '**steal**';

3. **Force**: tutte le pagine caricate in memoria centrale da una transazione devono essere trascritte in memoria secondaria dopo il commit (in modo sincrono)**;**
4. **No Force**: la trascrizione delle pagine in memoria di massa viene lasciata ai meccanismi di gestione asincrona del BM, *consentendo che le scritture avvengano al di fuori dei confini transazionali* [ACBT 326].

La coppia di politiche '**no force'/'no steal'** è quella maggiormente impiegata nei DBMS commerciali poiché la 'no steal' è di più semplice realizzazione e 'no force' è più efficiente.

La politica di sostituzione (*replacement policy))*

In base a questa politica il BM sceglie la pagina del Buffer Pool da sostituire tra quelle disponibili (vittima). Le politiche più adottate in ambito commerciale sono:

Last Recent Used (LRU): i riferimenti alle pagine disponibili (pin_count=0) vengono inseriti in una coda (L.I.F.O.) e prelevati, all'occorrenza, dalla testa. Con questa politica vengono sostituite le pagine disponibili da più tempo.

Clock: Supponendo che le pagine del buffer pool siano N, questa politica seleziona la pagina referenziata da una variabile intera, denominata *current*, il cui compito è quello di **indicare la prossima pagina (frame) disponibile per la sostituzione**. A sostituzione avvenuta, la variabile *current*, che può assumere valori nell'intervallo (1, N), viene incrementata mod(N) fino al valore corrispondente alla prossima pagina disponibile. Le pagine disponibili sono quelle che hanno le variabili *pin_count* =0 e *referenced*=0. La variabile booleana *referenced* ha lo scopo di ritardare la scelta di una pagina di un ciclo della variabile *current*.

La ricerca della pagina da sostituire, dunque, è ciclica (da cui il nome *Clock*) e si può sintetizzare nei seguenti passi:

1. se la pagina indicata da *current* è disponibile (pin_count=0 e *referenced*=0), il BM esegue la sostituzione, setta la variabile *referenced* =1 ed incrementa mod(N) la variabile *current*;
2. se la pagina indicata da *current* non è disponibile (*pin_count*=0, *referenced*=1), il BM pone la variabile *referenced*=0, in modo da renderla disponibile al prossimo ciclo, ed incrementa mod(N) la variabile *current*;
3. se la pagina indicata da *current* non è disponibile (*pin_count*>0, *referenced*=1), il BM incrementa mod(N) la variabile *current*;

La politica Clock, come quella L.R.U. di cui è considerata un'evoluzione, consente al BM di scegliere per la sostituzione le pagine disponibili da più tempo.

La politica Clock soffre di un problema, peraltro molto raro, denominato ***flooding sequenziale*** che si verifica allorché tutte le transazioni attive richiedano, in modo non concorrente, la scansione sequenziale di un *file* di K pagine, con K>N. In queste condizioni il BM compierà ripetutamente il caricamento di tutte le K pagine del file, come mostra il seguente esempio.

Esempio:

Supposto N=10 e K=11, rappresentiamo le scansioni effettuate nel tempo dalle transazioni attive, a partire dall'istante t_0, nella seguente figura, dove le pagine del file sono evidenziate.

File	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**K**	**L**	**tempo**
Lettura T_1	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**K**		$T=t_0$
Lettura T_1	**L**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**K**	(A)	$T=t_0+\tau$
Lettura T_2	**L**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	(K)	$T=t_0+2\tau$
Lettura T_2	**L**	**A**	**K**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	(B)	$T=t_0+3\tau$
Lettura T_3	**L**	**A**	**K**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	(I)	$T=t_0+4\tau$
…	…	…	…	…	…	…	…	…	…	…	**Vittima**	

Le caselle, la cui lettera è sottolineata, rappresentano le pagine del buffer pool referenziate dalla variabile *current* all'istante considerato.

Come si vede dalla figura, il BM è costretto, per ogni nuova transazione, a caricare l'ultima pagina vittima, sostituita durante la scansione della transazione precedente, e parte delle altre pagine del file compiendo, così, nel tempo ripetute scansioni.

Altre politiche di sostituzione sono:

- FIFO;
- MRU (*Most Recently Used)*
- Random.

Buffer Manager e File System

Molti DBMS commerciali impiegano alcune funzionalità I/O di basso livello disponibili nei recenti sistemi operativi. Tra queste funzionalità troviamo:

- **create():** crea un nuovo file;
- **delete():** elimina un file;
- **extend():** aumenta le dimensioni i blocchi di un file;
- **open():** apre un file;
- **close():** chiude un file;
- **read():** legge un blocco di un file;
- **read_seq():** legge una sequenza di blocchi di un file;
- **write():** scrive un blocco in un file;
- **write_seq():** scrive una sequenza di blocchi in un file.

Per quanto riguarda l'organizzazione interna dei file, generalmente il BM di un DBMS commerciale ne adotta una propria per migliorare e garantire efficacia, efficienza e transazionalità. Infatti, sebbene anche i sistemi operativi debbano risolvere il problema dell'esigua disponibilità della memoria centrale, i DBMS possono sfruttare la capacità dipoter predire l'ordine con cui le pagine saranno richieste e rilasciate, le cosiddette page *reference patterns*, per:

- una migliore scelta delle pagine da sostituire;
- precaricare in memoria centrale le pagine che si prevede saranno richieste dai livelli superiori (tecnica del *pre-fetching*);
- prescaricare in memoria secondaria le pagine che si prevede saranno rilasciate dai livelli superiori (tecnica del *pre-flushing*);

I DBMS, inoltre, devono garantire il *crash-recovery* (vedi parte III), per cui molte pagine devono essere scritte nella memoria disco prima di altre per poter rispettare certi protocolli, quali il W.A.L. (vedi parte III).

Dunque, concludendo, il BM richiede più controllo sulle operazioni di lettura e scrittura delle pagine di quanto un comune S.O. può garantire.