Sistemi Operativi

Ionut Zbirciog

7 December 2023

Performance del File System

C'è un gap di velocità di accesso troppo grande tra RAM e memoria volatile, negli ultimi anni questo gap è stato minimizzato con l'invenzione degli SSD. Per leggere una parola a 32 bit, la RAM ci mette circa 10ns mentre un disco magnetico ci mette circa 10ms. Pertanto, bisogna progettare un file system con diverse ottimizzazioni per migliorare le prestazioni, considerando le significative differenze nel tempo di accesso e riducendo al minimo i numeri di accesso al disco/SSD.

Ci sono due tecniche principali per ottimizzare il file system.

Uso della Cache

Utilizzata per ridurre i tempi di accesso al disco mantenendo i blocchi più usati in memoria. La cache si trova all'interno della memoria RAM.

Concetti di Caching

- **Buffer Cache**: Memorizza i blocchi del disco in RAM per ridurre gli accessi al disco.
- **Page Cache**: Memorizza le pagine del filesystem virtuale (VFS) in RAM prima di passare al driver del dispositivo. Sono struttura di intermezzo tra il virtual file system e il vero file system.

Le cache devono essere ottimizzate in modo tale che non ci siano file duplicati, perché può succedere che buffer cache e page cache spesso contengono gli stessi dati, per esempio file "mappati in memoria".

In poche parole, i file sono nella cache delle pagine e i blocchi del disco sono nella cache buffer. I sistemi operativi possono combinare buffer cache e page cache per un uso più efficiente della memoria e una riduzione degli accessi al disco.

Implementazione della Cache

Per la gestione della cache si può usare il seguente algoritmo: si controllano tutte le richieste di lettura per verificare se il blocco necessario sia nella cache. Se lo è, la richiesta di lettura è soddisfatta senza accedere al disco. Se non lo è, per prima cosa viene letto da disco e portato nella cache, quindi copiato ovunque ve ne sia bisogno. Le successive richieste per lo stesso blocco potranno essere soddisfatte dalla cache.

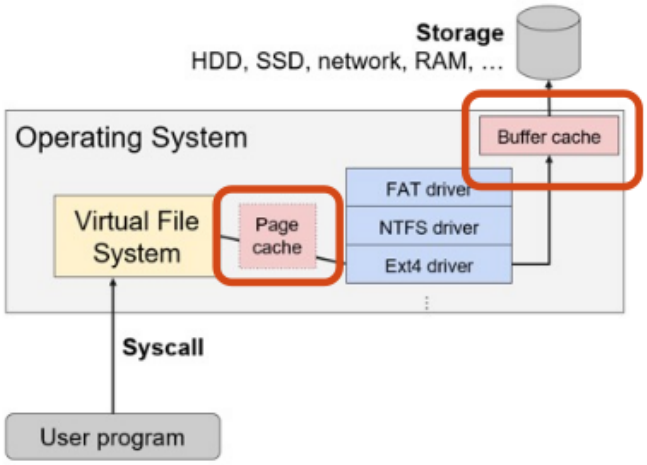

Figure 1: Struttura della cache.

Generalmente nella cache vi sono molti blocchi e serve un metodo rapido per determinare se un blocco sia o meno nella cache. Viene quindi utilizzata una hash table per velocizzare la ricerca di un blocco all'interno di un blocco.

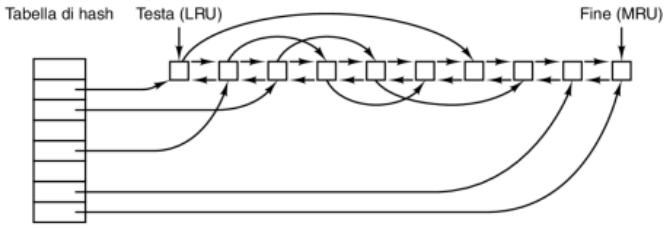

Figure 2: Struttura di una hash table nella cache.

Quando un blocco deve essere caricato in una cache piena, ne deve essere rimosso qualcuno. Questa situazione è molto simile alla paginazione e sono applicabili tutti gli abituali algoritmi di sostituzione delle pagine, come FIFO, seconda chance ed LRU. Una gradevole differenza fra l'uso della paginazione e della cache sta nel fatto che i riferimenti alla cache sono relativamente poco frequenti, così è possibile mantenere tutti i blocchi nell'esatto ordine LRU con le liste concatenate.

C'è un unico problema però: LRU può portare a incosistenza in caso di crash, specialmente per blocchi critici come gli i-node. Per esempio, un blocco di i-node viene letto nella cache e modificato, ma non riscritto su disco, un crash del file system lo lascerebbe in uno stato incoerente. Se il blocco di i-node viene posto alla fine della catena LRU, può volerci parecchio tempo prima che raggiunga la testa e sia riscritto sul disco. Per risolvere questo problema si può pensare a modificare

LRU, dividendo i blocchi per categorie basate sulla loro importanza. Se un blocco è fondamentale per la coerenza del file system, questo blocco dovrebbe essere scritto immediatamente sul disco, indipendentemente dalla sua posizione nella LRU.

Oltre a queste tecniche, UNIX mette a disposizione una chiamata di sistema 'sync' che forza la scrittura di tutti blocchi modificati. Al momento dell'avvio del sistema, viene eseguito un processo in background che ad ogni 30 secondi scrive tutti i blocchi modificati su disco. Ad oggi esiste una chiamata simile anche in Windows, ma prima veniva usata un'altra tecnica detta write-through, ovvero il sistema scriveva su disco ogni blocco che veniva modificato appena veniva scritto in cache. Una conseguenza di tale differenza nella strategia di caching è che la rimozione di un disco da un sistema UNIX senza fare la sync (per questo è opportuno prima di togliere un disco, fare umount) provoca quasi certamente una perdita di dati, e spesso intacca anche il file system. Con le cache write-through il problema non si pone.

Concettualmente cache page e cache buffer sono diversi nel senso che la cache delle pagine contiene le pagine dei file per ottimizzare l'I/O, mentre la cache buffer contiene semplicemente blocchi del disco. La cache buffer, nata prima della cache delle pagine, in realtà si comporta un po' come un disco, tranne che letture e scritture avvengono nella memoria. Il motivo per cui è stata aggiunta una cache delle pagine è che sembrava una buona idea portare la cache più in alto nello stack, in modo che le richieste di file potessero essere soddisfatte senza passare per il codice del file system e tutte le sue complessità. In altre parole: i file sono nella cache delle pagine e i blocchi del disco nella cache buffer. Alcuni sistemi operativi combinano cache buffer con cache page per una gestione efficiente dei dati. Inoltre, supportano i file mappati in memoria, trattando allo stesso modo pagine e blocchi del disco, in un'unica cache.

Allocazione dei Blocchi e Read Ahead

Un'altra tecnica importante per i dischi magnetici è quella di ridurre la quantità di movimenti del braccio mettendo vicini, preferibilmente nello stesso cilindro, i blocchi ai quali è probabile si acceda in sequenza. Quando viene scritto un file di output, il file system deve allocare i blocchi uno alla volta, a richiesta. Se i blocchi liberi sono registrati in una bitmap e l'intera bitmap è nella memoria principale, è abbastanza semplice scegliere un blocco libero che sia il più vicino possibile al blocco precedente. Se invece si usa la lista dei blocchi liberi, parte della quale sta su disco, è molto più difficile allocare blocchi vicini l'uno all'altro.

Altro collo di bottiglia delle prestazioni nei sistemi che usano gli i-node, o qualcosa di analogo, è che la lettura di un file, anche piccolo, richiede due accessi al disco: uno per l'i-node e uno per il blocco.

- a) Posizionamento tradizionale degli I-node: I-node vicino all'inizio del disco, risultando in ricerche più lunghe.
- b) Centrare gli I-node: Posizionare gli I-node nel mezzo del disco per dimezzare il tempo di ricerca. Dividere il disco in gruppi di cilindri con I-node, blocchi e lista dei blocchi liberi per ciascun gruppo.

Ovviamente, i movimenti del braccio del disco e il tempo di rotazione sono importanti solamente se il disco è magnetico; non sono rilevanti per gli SSD, che non hanno parti mobili. Per questi dischi, costruiti con la tecnologia delle schede flash, l'accesso casuale (in lettura) è rapido quanto quello sequenziale.

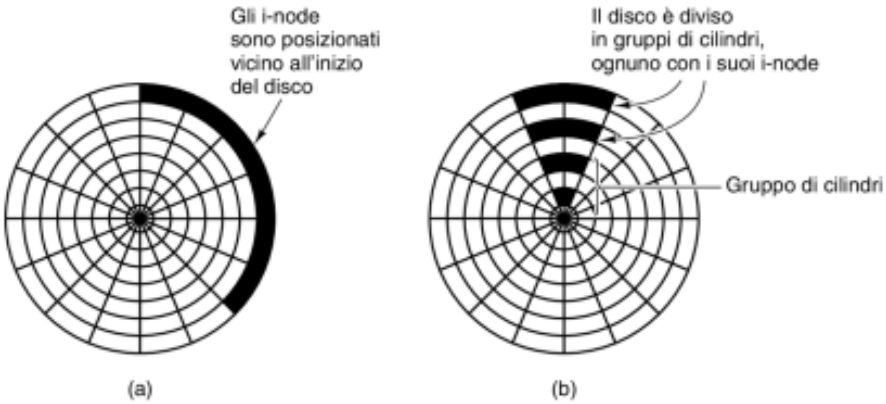

Figure 3: Posizionamento degli I-node.

Deframmentazione dei dischi

Con il passare del tempo, i file vengono creati ed eliminati e di norma il disco diventa molto frammentato, con file e buchi sparsi ovunque. Di conseguenza, quando viene creato un nuovo file i blocchi utilizzati potrebbero essere sparpagliati per tutto il disco, ottenendo basse prestazioni.

Le prestazioni possono essere ripristinate spostando i file in modo che siano contigui e mettendo tutto (o la maggior parte) dello spazio libero in una o più grandi zone continue su disco. Windows ha un programma, defrag, che fa esattamente questo. Gli utenti di Windows dovrebbero eseguirlo con regolarità, ma non sugli SSD.

I file system di Linux (specialmente ext3, ext4, btrfs) gestiscono meglio la frammentazione, riducendo la necessità di deframmentazione manuale. Ext4 introduce la preallocazione dei blocchi. Quando si scrive un file, Ext4 prealloca un gruppo di blocchi contigui, piuttosto che uno alla volta, riducendo la frammentazione per i file in espansione.

Comprensione e Deduplicazione

Alcuni file system come NTFS, Btrfs possono comprimere dati automaticamente e scrivere sul disco i dati compressi per risparmiare spazio. Oltre a eliminare la ridondanza entro un singolo file, alcuni file system eliminano anche la ridondanza in tutti i file. Nei sistemi che memorizzano dati di molti utenti, ad esempio in un ambiente cloud o server, è facile trovare file che contengono gli stessi dati quando più utenti salvano gli stessi documenti, file binari o video. Anziché salvare più volte gli stessi dati, alcuni file system implementano la deduplicazione per eliminare i duplicati.

La deduplicazione può essere eseguita inline o post-process. Con la deduplicazione inline, il file system calcola uno hash per ogni chunk ("pezzo" di memoria) che sta per scrivere e lo confronta con gli hash dei chunk esistenti. Se il chunk è già presente, eviterà di scrivere i dati e aggiungerà invece un riferimento (hard link) al chunk esistente. Naturalmente i calcoli aggiuntivi richiedono tempo e rallentano la scrittura. La deduplicazione post-process, invece, scrive sempre i dati ed esegue l'hashing e i confronti in background, senza rallentare le operazioni di elaborazione dei file.

Affidabilità del File System

Minacce alla Affidabilità del File System

- **Guasti del Disco:** blocchi danneggiati o settori illeggibili che possono corrompere dati, o errori su intero disco, rendendo il disco inutilizzabile.
- **Interruzioni di Energia:** causano incongruenze nei dati o nei metadati sul disco.
- **Bug del Software:** errori di programmazione portano alla scrittura di dati errati/corrotti.
- **Errori Umani:** ad esempio, "rm *.o" vs "rm * .o"; il primo cancella tutti i file con estensione '.o', mentre il secondo cancella tutti i file (*) e poi, se ci sono ancora tutti i file che finiscono con '.o'.
- **Perdita o Furto del Computer:** rischi di accesso ai dati da parte di persone non autorizzate in caso di furto o smarrimento del dispositivo.
- **Malware/Ransomware:** virus o altri software dannosi che possono infettare, criptare o distruggere dati.

Backup

I backup su disco sono generalmente effettuati per affrontare uno dei due potenziali problemi:

1. **Recupero da un Disastro:** crash del disco, incendio, catastrofe naturale.
2. **Recupero dalla Stupidità:** eliminazione accidentale di file; ad esempio, in Windows esiste il cestino, una cartella in cui vanno i file 'eliminati', in modo poi da poterli recuperare.

Ci sono cinque punti fondamentali di cui tener conto.

1. Un backup impiega molto tempo e occupa molto spazio, per questo è importante farlo in modo efficiente e comodo. Di quali file bisogna fare il backup? Di tutto il file system o solo di alcune directory?
2. È uno spreco rifare il backup di file che non sono cambiati dall'esecuzione dell'ultimo backup, e questo ci porta al concetto di backup incrementale. La forma incrementale più semplice consiste nel fare periodicamente un backup completo e eseguire un backup giornaliero solo dei file modificati dopo l'ultimo backup completo.
3. Poiché generalmente sono memorizzate grandi quantità di dati, potrebbe essere utile comprimere i dati prima di scriverli sul supporto di backup.
4. È difficile eseguire il backup di un file system attivo. Sono stati definiti algoritmi che creano delle istantanee (snapshot) dello stato del file system.
5. I backup devono essere sempre tenuti al di fuori del luogo dove risiedono i computer, introducendo ulteriori rischi per la sicurezza.

Strategie di Backup su Disco

- **Backup Fisico:** Copia sequenziale di tutti i blocchi del disco, semplice e veloce, ma ha difficoltà nel saltare le directory specifiche o nel fare backup incrementali. Non è possibile ripristinare file individuali senza un intero ripristino del sistema.
- **Backup Logico:** Seleziona e copia solo i file e le directory specifici, ideale per backup incrementali o completi. Permette il ripristino o il trasferimento dell'intero file system su un nuovo computer.

Algoritmo di Backup Logico

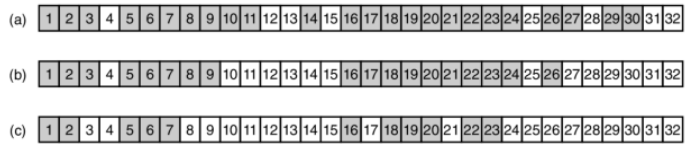

Figure 4: Algoritmo di backup logico.

L'algoritmo consiste di 4 fasi:

- **Fase 1. Rilevamento delle modifiche:** Inizia dalla directory radice, esamina tutte le voci e contrassegna nella bitmap gli I-node di file e directory modificati.
- **Fase 2. Pulizia della Bitmap:** Deseleziona le directory che non contengono né file modificati né sottodirectory modificate.
- **Fase 3. Backup delle Directory:** Backup delle directory contrassegnate, con i loro attributi.
- **Fase 4. Backup dei File:** Backup dei file contrassegnati, sempre con i relativi attributi.

Per ripristinare un file system con un backup, si crea un nuovo file system vuoto, poi viene ripristinato il backup completo, seguito dai backup incrementali.

- Dato che la lista dei blocchi liberi non è un file, non è oggetto di backup e perciò deve essere ricostruita da zero alla fine del ripristino di tutti i backup.
- Se un file è collegato a una o più directory tramite link, è importante che quel file sia ripristinato solo una volta e che tutte le directory che dovrebbero puntare a esso lo facciano.
- Un'ulteriore questione è il fatto che i file UNIX possono contenere dei buchi. È consentito aprire un file, scriverci pochi byte, quindi spostarsi a un offset distante e scrivervi altri byte. I blocchi nel mezzo non sono parte del file, non dovrebbero essere salvati nel backup né ripristinati.
- File speciali come "pipe" e altri file simili non andrebbero mai salvati nel backup, indipendentemente dalla directory in cui si possano trovare.

Coerenza

La coerenza del file system è cruciale per mantenere l'integrità dei dati. Problemi di incoerenza possono sorgere a seguito di crash durante la scrittura dei blocchi. In UNIX si utilizza `fsck`, e in Windows `sfc` per verificare la coerenza all'avvio dopo un crash.

Controllo dei blocchi

Costruzione di due tabelle con contatori per ogni blocco (per file e blocchi liberi). Analisi di tutti gli I-node e verifica della presenza dei blocchi in file e nella lista dei blocchi liberi. Dopo questo controllo si possono avere 3 riscontri possibili:

1. Blocchi mancanti: non presenti in nessuna delle due tabelle.
2. Blocchi duplicati nella lista dei blocchi liberi.
3. Blocchi di dati presenti in più file.

Per correggerli si possono fare due azioni:

1. Aggiunta dei blocchi mancanti ai blocchi liberi.
2. Assegnazione di blocchi duplicati a file diversi.

File System con Journaling

Registra anticipatamente le operazioni da eseguire in un log, per garantire la coerenza in caso di crash. Ampiamente usato in NTFS, ext4, macOS. Un journal in un file system è come un registro che tiene traccia delle modifiche che verranno apportate al file system prima che esse avvengano effettivamente.

Funzionamento:

- **Fase di Registrazione:** prima di eseguire qualsiasi modifica, il file system scrive un record nel journal, descrivendo l'operazione che verrà eseguita.
- **Fase di Esecuzione:** dopo aver registrato l'operazione, il file system procede con la modifica effettiva dei dati sul disco.
- **Fase di Conferma:** una volta completata l'operazione, il file system aggiorna il journal per indicare che l'azione è stata completata con successo.

Se si verifica un crash del sistema prima che un modifica sia completata, al riavvio successivo il file system constata il journal. Se trova operazioni registrate ma non confermate procede a completarle, in questo modo il file system non verrà mai lasciato in uno stato incoerente.

Vantaggi:

- **Integrità dei Dati:** il journaling riduce la possibilità di corruzione del file system in caso di crash inaspettato, assicurando che tutte le operazioni siano completate o nessuna.
- **Recupero Rapido:** Riduce significativamente il tempo di recupero dopo un crash.

Eliminazione sicura e cifratura del disco

La cancellazione standard non rimuove fisicamente i dati dal disco, lasciandoli vulnerabili agli attacchi. L'eliminazione sicura richiede la distruzione fisica o la sovrascrittura approfondita dei dati. Non basta sovrascrivere con zero a causa dei residui magnetici che possono essere recuperati con tecniche avanzate. È consigliato inserire sequenze di zeri e numeri casuali, ripetendo l'operazione almeno 3-7 volte, sconsigliato su SSD.

La soluzione più efficace per proteggere i dati è cifrare l'intero disco con algoritmi robusti come l'AES.

File System Virtuali

I sistemi operativi moderni gestiscono diversi file system simultaneamente. Windows utilizza lettere come C:, D:, per gestire file system differenti. I sistemi UNIX tentano di integrare più file system in una singola struttura gerarchica.

Il VFS è una struttura che permette di integrare vari file system in una struttura unificata. Si basa su un livello di codice comune che interagisce con i file system reali sottostanti. L'interfaccia superiore interagisce con le chiamate di sistema POSIX di processi utente, mentre l'interfaccia inferiore è composta da decine di funzioni che il VFS può inviare ai file system sottostanti.

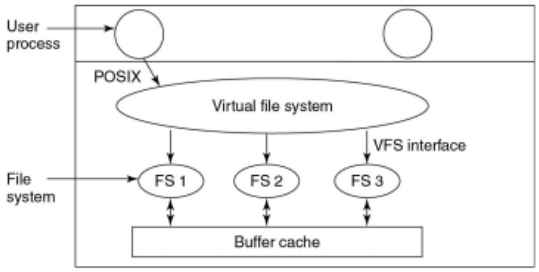

Figure 5: Struttura del File System Virtuale (VFS)

- **Superblock nel VFS:** rappresenta il descrittore di alto livello di un file system specifico nel VFS. Contiene informazioni cruciali sul file system ed è usato per identificare e interagire con il file system sottostante.
- **V-node nel VFS:** astrazione di un file individuale all'interno del VFS, rappresentando un nodo nel file system virtuale. Contiene metadati come permessi, proprietà, dimensione del file. Il VFS sfrutta i v-node per fornire.
- **Directory nel VFS:** struttura che gestisce l'organizzazione e il mapping dei file e delle sottodirectory all'interno del VFS. Permette al VFS di mappare i nomi dei file ai loro v-node corrispondenti, indipendentemente dal file system in cui si trovano.

Al momento della registrazione di un nuovo file system all'interno del VFS, il file system mette a disposizione un vettore di funzioni per il VFS. Inoltre, al montaggio, il file system mette a

disposizione tutte le informazioni necessarie, ad esempio il superblock. La gestione delle richieste di I/O nei processi utente avviene tramite i v-node e tabelle dei descrittori dei file. Aggiungere un nuovo file system diventa relativamente semplice, i progettisti devono soltanto fornire le funzioni che rispettino l'interfaccia VFS.

RAID

RAID, acronimo di Redundant Array Of Inexpensive (Independent) Disks, è un meccanismo per migliorare le prestazioni e l'affidabilità della memoria non volatile.

- RAID Level 0: +storage Descrizione dei dati in strip su dischi multipli, migliora le prestazioni con richieste grandi. Nessuna ridondanza.
- RAID Level 1: +redundancy Duplicazione dei dischi per tolleranza agli errori. Prestazioni di lettura migliorate, scrittura simile a un'unità singola.
- RAID Level 2: +storage Level 2 basato su parole o byte con codice di Hamming.
- RAID Level 3: +redundancy, +storage Level 3 usa un singolo bit di parità per parola, richiede sincronizzazione delle unità.
- RAID Level 4: +redundancy, +storage Utilizzo di strip con un'unità extra per la parità.
- RAID Level 5: +redundancy, +storage Distribuisce bit di parità in modo uniforme su tutte le unità.
- RAID Level 6: +redundancy, +storage Simile a Level 5 ma con un blocco di parità aggiuntivo. Maggiore affidabilità e tolleranza degli errori.
- RAID 0 + 1: duplicazione ridondante secondo lo schema di RAID 1 di dischi associati in schema RAID 0

Esempio RAID 5

RAID 5 con 3 dischi: Disco A, Disco B, Disco C. Ogni disco contiene una strip di dati e una strip di parità viene calcolata usando l'operazione XOR bit a bit dei dati da Disco A a Disco B, e poi memorizza sul Disco C.

- Disco A: 1011
- Disco B: 1100
- `XOR bit a bit tra A e B`
- Disco C: 0111

In caso di guasto di un disco, ad esempio il disco B, possiamo ricostruire i suoi dati facendo lo XOR tra il disco A ancora funzionante e il disco C.