Semantica della consistenza sul file system

sabato 17 giugno 2023 16:27

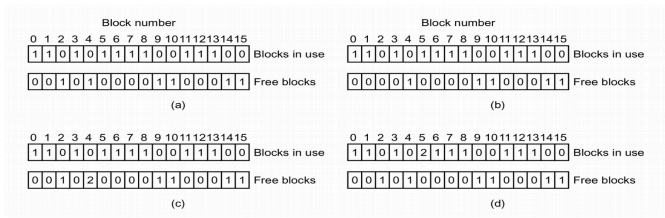
Abbiamo il disco rigido D fisico in cui i dati sono presenti, quindi i blocchi che contengono i record dei file, però in RAM abbiamo il buffer cache.
Un aggiornamento di un file F avviene quando noi portiamo il blocco che contiene i dati dal disco rigido al buffer cache, e aggiorniamo il buffer cache. Ci manteniamo i dati aggiornati in buffer cache. Quando poi dobbiamo aggiornare e rendere permanente il contenuto del file dopo un power off, dal buffer cache il blocco viene riportato nel disco rigido.
Se all'istante T0 io porto il blocco in buffer cache e all'istante T1 migro il blocco aggiornando il file su disco, tra T0 e T1 può succedere qualsiasi problema, ad esempio un problema sull'alimentazione. Quindi i dati che sono in buffer cache rimangono lì in ram e non sono correttamente portati sul dispositivo di memoria di massa perché ancora non è stata chiamata l'operazione che riporta il blocco nel disco rigido dato che abbiamo avuto un errore.
Ciò che abbiamo scritto all'interno di un file in realtà è perso. Quindi sul dispositivo di memoria di massa D questa informazione aggiornata non c'è mai andata. Quindi ci troviamo un file system che è inconsistente.

Ogni scrittura di dati sul file system è sicuramente visibile ad applicazioni che eseguono la lettura sugli stessi dati, e queste applicazioni leggono da buffer cache, quindi è palese. E chi scrive, scrive proprio in buffer cache. Abbiamo un'inconsistenza dei dati che sono effettivamente su dispositivo di memoria di massa e dei dati che sono in buffer cache, se non c'è stato aggiornamento dei dati su dispositivo di memoria di massa.
Questa è una semantica Unix supportata anche da WINDOWS.

Abbiamo la semantica di sessione che dice che dopo che un file è stato aggiornato in realtà l'aggiornamento è visibile soltanto dopo che il file è stato anche chiuso - questa è una semantica che è stata inventata in alcune varianti dei file system unix (ANDREW IN PARTICOLARE).

Però ci sono problemi di inconsistenza rispetto a quello di cui parlavamo prima. C'è necessità di ricostruire il file system quando noi andiamo a ri-lavorare su questo file system che non era stato sincronizzato correttamente su dispositivo di memoria di massa rispetto agli aggiornamenti che avevamo effettuato all'interno della RAM.

Ricostruzione dei file system



a) Situazione consistente

b) *Il blocco 2 non è in nessun file né nella lista libera*: aggiungilo alla lista libera

c) *Il blocco 4 compare due volte nella lista libera*: togli un'occorrenza

d) *Il blocco 5 compare in due file*: duplica il blocco e sostituiscilo in uno dei file (rimedio parziale!!!)

La situazione A è una situazione interessante. Se noi con delle BIT MAP rappresentiamo quali sono i blocchi usati e quali sono i blocchi free e con dei counter che ci dicano quante istanze di questi blocchi sono free/in uso, quello che abbiamo è che un blocco ha sempre un 1-0 o 0-1. Se è libero nella free blocks metto 1 e nel blocco in uso metto 0, e viceversa.

Però potremmo avere scenari sulla situazione c in cui abbiamo che il blocco numero 4 non è in uso ma è livero due volte, perché magari i metadati di questo file system sono stati aggiornati correttamente rispetto agli aggiornamenti che erano stati fatti invece in RAM.

Queste sono situazioni che sono facilmente risolvibili, ma sono rimedi parziali quando cerchiamo di riparare il contenuto di un file system che non era stato correttamente sincronizzato rispetto alle operazioni che erano state fatte in RAM.

Per effettuare queste operazioni di riparazione e di controllo della consistenza abbiamo sicuramente delle applicazioni che possono essere utilizzate sia in UNIX sia in WINDOWS:
-FSCK in UNIX ci permette di controllare lo stato di un file system su un dispositivo di memoria di massa.
-SCANDISK in WINDOWS la stessa cosa.

- vengono analizzate le strutture del file system (MFT, I-nodes) e si ricava per ogni file system:
  - *Blocchi in uso*: a quanti e quali file appartengono (si spera uno eccetto che in scenari di deduplicazione)
  - *Blocchi liberi*: quante volte compaiono nella lista libera (si spera una o nessuna)

SINCRONIZZAZIONE DEL FILE SYSTEM.

Su linux possiamo chiamare FSYNC E FDATASYNC che prendono entrambe un file descriptor, ed è una richiesta al kernel di sincronizzare i dati che sono sul dispositivo di memoria di massa rispetto agli aggiornamenti di questi dati che sono stati effettuati all'interno della RAM, all'interno del buffer cache o per esempio all'interno della cache degli I-NODES. Andiamo ad aggiornare un file F con una write in buffer cache, non c'è una sincronizzazione col contenuto di quel file aggiornato sul dispositivo di memoria di massa, questa cosa verrà fatta dal software del virtual file system quando chiamiamo una di quelle due API, in questo modo chiediamo in maniera esplicita al VFS di prendere i dati dal buffer cache e di flasharli su dispositivo di memoria di massa. Quindi dopo aver fatto questo, se il sistema va in shutdown in maniera scorretta siamo tranquilli che l'aggiornamento lo ritroviamo all'interno di quel file anche dopo lo start up.
Queste API tornano il controllo solamente quando i dati sono stati flashati sul dispositivo.

Con fdatasync() noi siamo interessati a riportare su dispositivo di memoria di massa (flashare) soltanto i dati aggiornati di un file, non i metadati, non l'i-node. Qui aggiorniamo il file ma non riaggiorniamo per sincronizzare l'i-node, questo perché l'i-node è da una parte e i dati sono da un'altra parte. Quindi se io voglio scrivere un i-node e sincronizzarlo nel disco devo fare un'operazione apposita per riportare il blocco relativo al vettore i-nodes dove c'è l'informazione aggiornata DA RAM a DISCO, e poi ovviamente i dati dei blocchi.
Facendo fdatasync() noi sincronizziamo solo i dati.
Mentre invece fsync() sincronizza anche l'i-node. Quindi tutto.

Su WINDOWS abbiamo il duale, dato un handle (maniglia) abbiamo la possibilità di chiamare FlushFileBuffers() quindi richiedere tutti gli aggiornamenti di uno specifico handle che ci porta su una sessione di uno specifico file, devono essere flashati su dispositivo di memoria di massa.