

Implementazione delle directory

Sono tipi speciali di file che hanno il compito di incapsulare altri oggetti, il loro contenuto è sostanzialmente un elenco di file.

Il contenuto di un file è un file dell'elenco di file del file system usato.

Nel caso di FAT: tutti i metadati sono memorizzati nel record insieme a nome-file e numero del 1o blocco.

Nel caso di C-mode: si memorizzano nome/file e C-number

Per rappresentare la via di una dir. ci sono diverse strategie

Ogni strategia prevede però necessariamente una sorta di struttura di dati e una variabile quella a sua volta rappresenta il nome della file

1^a strategia

Per ogni voce ~~solitamente~~ max

l'entry del file. L'entry del file contiene:

- entry length

- attributi

- nome (cattivo)

cattivo var e punti

cattivo di terminazione

padding per allineamento/contendere
di word

file 1 entry length

file 1 entry length
file 1 attr.
a b c d
e <input checked="" type="checkbox"/>
file 2 entry length

attributi

indice lung di tutta
la entry

estettrare la
terminazione

Compte spazi; uscire
di fram. interni.

2^a strategia

pointer to file 1's name

File 1 attr

pointer to file 2's name

File 2 attr

a	b	c	d
e <input checked="" type="checkbox"/> f g			
h i l <input checked="" type="checkbox"/>			

I cattivi padding non
sono previsti
ma non sono aggiornati
in caso nuovo file

Nella parte iniziale abbiamo

pointer al nome e gli
attributi. La parte alla
fine ha un file

header. Se questo non
è problema o siamo costretti

che non è un file

La concatenazione dell'header viene rimpicciolita dal
ultimo n bytes meno due (non cattivo)

Ricerca di un file con un certo nome (it name o the phone)

Nei file system moderni i dati sono predisposti in cache Host che garantiscono ricerche efficienti. I meccanismi di ricerca sono favolti da un supporto detto Cache del file che velocizza l'accesso. Questa cache è implementata su un set di 50 spaziando la RAM.

Combinazione di file in file system

Se già utili vogliamo usare un file e mi ritrovo in un file system basato in c-mode possiamo usare gli Hard Link (Conservano file).

Gli Hard Link sono dei riferimenti agli ~~file~~ c-mode che vogliamo mettere in una directory.

Tra i metadati dell'i-mode vi è il Contatore di Hard Link utile per il file system.

Metti in fase di rimozione della dir:

- la voce nulla dir è cancellata
- il contatore viene decrementato
- Se è uguale a 0: tutti i blocchi dell'i-mode vengono liberati

Con questi meccanismi

possiamo verificare le anomalie di accounting.

Ovvio: C è il proprietario di un file, il file appartiene a B e poi C cancella la sua voce relativa all'i-mode del file.

B sta puntando a un file B chi non è proprietario e C è l'unico a puntarci (anomaly).

Cioè il file monterà come file correttamente
connesso dal proprietario l'orsa juntas da B.

Un'altra strategia prevede l'uso di soft-link,
ci sono dei vari e propri oggetti detti soft-link
che hanno come contenuto il path name dell'oggetto
riferito.

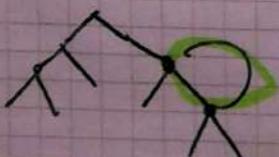
Avere un soft link dunque significa avere il
file a cui si giunta.

Perché con un soft link andrà a creare un
nuovo oggetto che ha il suo i-node & il suo path
contento il che appena dice il meccanismo
è l'unico modo per fare il riferimento (a appartenere
a oggetti al di fuori del file system nell'ambito
di soft link possono riferirsi a qualunque tipo
di oggetto).

In cosa non è possibile con gli Hard Link
dato che nello i-node del file metto il numero
dell'i-node dell'oggetto da riferire.

Dunque file-system differenti possono avere
oggetti diversi ma con i-number uguali, però.
L'Hard Link è utile solo nel contesto file system
partizione.

Gli Hard Link potenzialmente potrebbero colpire alle
directory ma la stessa sia altro file dir
verrebbe meno nel file system ed esso diventerebbe
un caso. Vedo che sarebbe problemi alla
maniera delle certe kemberly infinite.



N.B.: Differenze soft link e hard link

• Soft Link (Link simbolico)

Gerico file. Sono all'interno di solo il seth per il file. Appena si crea il necessario (altrimenti) se si cancella il file target il link diventa incommutabile.

• Hard Link

Veri e propri riferimenti agli i-node

E' un file contenente l'elenco i-node

Possiamo lavorare solo sulla partizione (i soft link no)

Non si possono usare gli Hard Link tra due

L'Hard Link crea una nuova "pista" per accedere al file

Gli Hard Link sono per definizione riferimenti allo stesso file/i-node del file

Moltiplicando un file da Hard Link lo modifica in tutte le quanti gli Hard Link, anche l'originale

Ultima cosa: Ho un file con un i-node e creo un Hard link riferito ad esso. Cancello il file delle BT originali, in teoria il file è ancora più accessibile quando l'Hard Link (il record) appena creato.

Se è un file System basato in FAT è possibile duplicare la lista com i riferimenti ai blocchi per fare sovrapposizione → problemi di open

Versione blocchi liberi

Tra i contatti del file system c'è quello di tenere traccia dello stato di allocazione dei blocchi nel disco.

Bit map: invilene di bit, ognuno rappresentante un blocco (ci dà l'info "libero" / "non libero")

La struttura è chiamata free blocks (dove si indicano le caselle vuote).

La struttura è usata per trovare blocchi liberi da usare per i nostri file.

Essa è memorizzata in disco e può contenere solo la posizione della nostra tabella nel RAM oppure pagine intere.

Lista concatenata: rappresentiamo una lista di blocchi liberi. Viene usato un proprio tipo di struttura che rappresenta l'informazione voluta usando i blocchi non liberi.

Per i blocchi liberi contigui ci usano poi dei contatori.

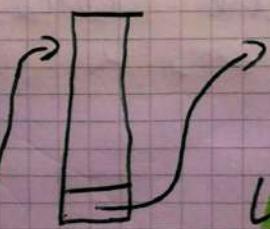
Nominalmente rendono vantaggiose il liste, vengono usate le BITMAP.

Le liste rappresentano SOLO i blocchi liberi (i numeri).

Struttura lista: ho 1KB di dimensioni di blocco e numeri a 32 bit.

Ogni nodo allora ha 256 voci ($(1024 \cdot 8)/32$) delle cui 255 per contenere numeri di blocchi (dati) in disco e una per puntare prossimo nodo.

10
20
30
...
8
...



Se il File system è importante molto la lista ordinamente in accrescere.

La lista vera è implementata tramite blocchi liberi.

La gestione delle liste è già racchiusa
L'elenco dei numeri non è ordinato (all'inizio si)
ma poi l'ordine a fini di allocare l'indirizzo
viene perso

Oggetto N fa un'allocazione o l'indirizzo si lavora
sulla lista delle liste

La ricerca di contiguità nella lista non è buona,
è inefficiente

Il controllo di blocchi liberi è protetto dalle memorizzazioni
ma non tanto i blocchi liberi non informano
sulle regole di blocchi liberi contigui.

Controlli di consistenza

Effettuati dal file system o da utility fette
a mano, invocati o periodicamente o dopo un crash
di sistema. In seguito ai Crash la RAM perde il
suo contenuto ma il file system non è robusto
e le operazioni sui blocchi devono riconoscere

Queste operazioni non sono atomiche e il SO
al pianto non è in grado di cogne dove
è avvenuta l'interruzione.

Dopo il pianto dunque è possibile che il file
system non sia coerente

Ese: potrebbero essere blocchi registrati come occupati
ma in realtà liberi facendo l'operazione è
stata interrotta e mette.

Per questi motivi vengono attivate le utility
di controllo di consistenza che cercano inconcordanze
all'interno del sistema e provano a correggerle

Contiene un meccanismo di controllo consistenza

Contiene 2 vitori: uno che indica il numero di volte che il blocco è usato e uno che indica il numero di volte che il blocco è citato come libero

Dopo aver creato i vitori (memoria file/c-mode e struttura dati che tiene traccia dei blocchi liberi)

dovrebbe contenere tutti 0 oppure 0 e 1. In realtà non garantisce viene a mandare

[0, 0] 1^o: bit nella tabella dei blocchi "in uso"
2^o: bit nella tabella dei blocchi liberi

• [0, 0]: blocchi usati che liberi? Fai la risposta: se segui il blocco alla fine liberi (il bit 2° sarà 1)

• [0, 1]: il blocco compare liber 2 volte (vi è una voce nella lista).

Basta ricordare la lista

• [1, 0]: stesso blocco nello stesso file/c-mode
Se cancelliamo da uno dei 2 si presenta [1, 1] (caso d'eccezione), se cancelliamo da entrambi si presenta [0, 2]

La soluzione è: allontanare un blocco libero, copiargli il contenuto del blocco in questione (quello che c'è da 2 volte) in uno dei due c-mode

(~~e sincronizzare quelli~~)

La struttura del file system viene resa consistente

(i due c-mode puntano a blocchi diversi ma con uguale contenuto)

Lo Journalling di controllo dei blocchi immutabili
non può applicare allo Journalling di controllo dei cambiamenti
degli i-nodo nelle biteset.

Journaling

Molti file system moderni impiegano queste tecniche
che consiste nell'effettuare una meta operazione
(che è un insieme di micro-operazioni) e di
prevedere una transazione fatti persistente sul JOURNAL
in cui vi si tiene una descrizione delle
operazioni che sta per effettuare.

Non vengono scritte tutte le info ma solo quello
che tocca i metadati per motivi di spazio.

Dopo che è stata eseguita l'operazione i log nel
Journal sono cancellati e si fa ritorno alla
prima meta operazione.

Idea: registrare le operazioni salvate sul log (che è
salvato sul Disk) in caso di crash.

Il Journaling ferde i metadati salvati e offre
un reboot veloce.

Il meccanismo è applicabile solo alle operazioni
salvate sul log sono interattivi ovvero rigettibili
senza fare damage.

Salvare permanentemente le operazioni sul log può
creare gravi over Head.

Cache del Disco (buffer cache)

La cache del disco è implementata via SW e
utilizza la RAM per "graze".

Ha le stesse 5 operazioni di I/O del
disco quindi possibile.

Idee: memorizzare blocchi già precedentemente usati

Ni sistemi UNIX qualiasi blocco di RAM libero è
usato nella cache del disco, bensìche la cache non
ha memoria fino.

Tutti i frame in questione sono inseriti in una
lista appositamente indicata, così si può applicare
come algoritmo di gestione l'LRU

Il costo della lettura su lista non sarà mai
presente quanto un I/O

Per prestazioni già elevate si usa una tabella Hash
che mappa i nodi della lista (non si usa una
lista circolare)

Dunque ho una struttura ibrida che permette
accesso veloce e LRU

La cache funziona anche in scrittura, infatti
ritorna (in chi può) lo stesso nel disco
del blocco.

Infatti ti turbano 10 scritture in DAS per poi
effettuare tutto di nuovo già in la provocando
materiali miglioramenti di prestazioni (il costo
di tifprimento continua della testina)

Il blocco sparsa (non intende da memorizzare) venrebbe scritto in disco solo quando scritto dall'LRU.

Questo già dove problemi in caso di cache, il blocco non sarà verrebbe ad essere memorizzato con i logici in RAM (anzi però)

L'integrità è più importante dell'efficienza per questo l'LRU usato è un po' modificato.

Se c'è bisogno di una sincronizzazione ti guarda i metadati ma non viene ritardata ma effettuata subito.

Se la memorizzazione non ti guarda metadati vi è un po' di ritardo ma che è controllato.

Viene meno un limite al ritardo

Optimizzazione Cache

- **Read-ahead**: Il SO quando carica un file in cache un blocco in carica anche i blocchi contigui.

Porta enormi vantaggi se il file è memorizzato in maniera sequenziale.

Questo è particolarmente importante, se avranno blocchi solisti in varie zone non contigue. Il meccanismo potrebbe in cache blocchi che non entrano nulla.

- **Free-behind**: L'idea è che in un processo sta leggendo un blocco. Difficilmente i blocchi precedenti saranno usati e quindi si liberano (abbiamo già spazio in RAM).

Queste due tecniche sono state usate insieme.

Antae toccherà più pezzi (nella elaborazione linea)

Il SO gestisce anche come riunire i dati sul disco
nel modo più efficiente possibile.

Ese: gli i-mode avendo pre-allocati sono messi in
una posizione ben precisa quindi potremo disporre
in modo contiguo nella traccia più esterna del
disco. A ogni i-mode corrisponderanno poi
dai blocchi che contengono il file.

Sappiamo che prima è letto l'i-mode e poi il
file, mettere distanze non ha senso

* Per rendere più efficiente la lettura di file conviene
mettere gli i-mode in tracce diverse e disporre
il file ~~in modo regolare sulla traccia dell'i-mode~~
~~corrispondente al file allo stesso cilindro~~

In questo modo il braccio meccanico con la testina
è嘛 il meno possibile nella lettura di un file

Dunque la distanza linea tra blocchi e i-mode corrisponde
al tempo per I/O per riportare fisicamente la testina

Un esempio sarebbe mettere gli i-mode invece
che sulla traccia più esterna sulla traccia intermedia
(il tempo medio di i-mode \rightarrow blocchi è dimezzato)

Nella realtà: * gli i-mode sono suddivisi
in gruppi, il gruppo è rappresentato dal cilindro
In un cilindro ci mette degli i-mode e blocchi
corrispondenti così le probabilità per il riavvicinamento
della testina migliori.

Se il file in un certo cilindro ha dei blocchi il
SO cerca di uscire blocchi dello stesso cilindro,
se non sono disponibili prende blocchi di altri
cilindri e si scommette

Idealmente il file è quindi sommerso tutto nello stesso cilindro.

Se non si può usare lo stesso cilindro (necessario)
uso blocchi di cilindri controllati

Abbiamo creato una sorta di new contiguous

per strategia

contiguous



(l'assegnamento prende in considerazione gli ammortatori nella traccia centrale)

Degradazione: process che monitora blocchi per ottenere la situazione ideale quando ci viene incontro.

Esercizi esami (file system)

1) i-node con metablo, 10 blocchi dati, 3 ad offset
(1 blocks indirizzo, 2 e 3)

Blocos di 5KB
numero di blocos: 32 bit \rightarrow Voci ? $\frac{5 \cdot 1024 \cdot 8}{32} = 1024$

In quale blocco viene ; 0 byte di offset XXX ?

Sì fa XXX (in bit !?) \rightarrow 5,36 (ad esempio)

attraversone per ecceso

5° bloco

NB: il 1° bloco è al 10
alla estensione.

Dunque se per esempio tale divisione si mette 12 e nel 2° bloco alla 1a estensione c'è 565

La risposta è 55.
 Dunque nella tabella prima: punto 10
 nella tabella seconda: 11 - 1035 (ogni)
 La 1^a voce della tabella terza è 10350.
 (in realtà la tabella terza giunta a
 1025 tabella con numeri di blocchi)
 La 1^a di queste va da 1035 a 2058, e
 neanche da 2060 a 3085

2) FAT

index	next
1	5
2	3
3	15
4	5
5	10
6	12
7	1
8	2
9	3
10	-1

Controlla facile

name	10	size
file1.txt	?	?
file2.txt	?	?
file3.txt	?	?

$$7 \rightarrow 1 \rightarrow 5 \rightarrow 5 \rightarrow 10$$

Dim del File? (blocks = 5 KB)

Connesso tra $(5 \times 5\text{KB}) + 1 = 26\text{KB}$

Impronta, blocks di file è 1025

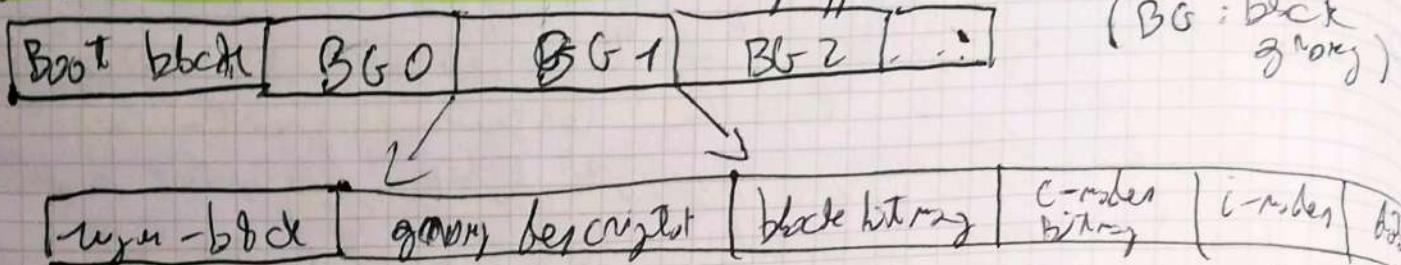
L'output 10100 dunque pippo.txt?

$$\frac{10100}{5096} = 2,3 \approx 3$$

Nel 3^o blocks di pippo
 Dunque 3 - 10 blocks
 E' stato creato il 3

File system di Linux

Bordo in i-mode e in gruppi di blocchi.

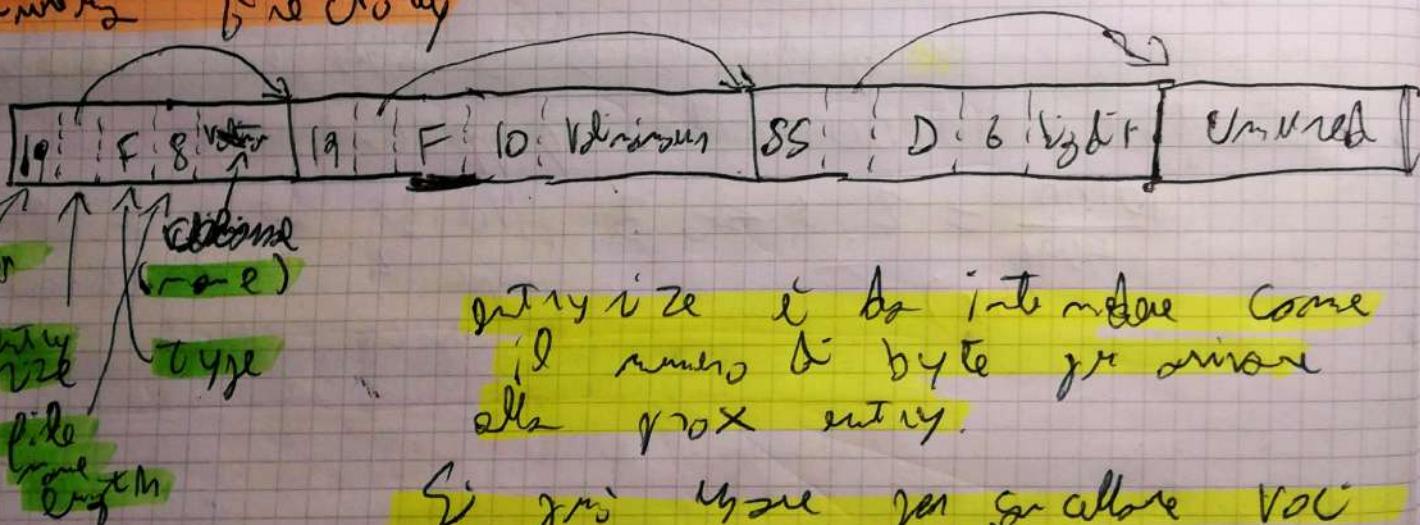


Nel singolo gruppo ha una sua struttura

- Super-block ha anche info sul gruppo stesso
- Unico degli i-nodes presenti in i-nodes gruppi, le gestisce con bitmap
- i block di dati li gestisce pure con bitmap

Bordo ha un file F in genere i suoi blocchi si cercano nei primi nello stesso BG per efficienza
Se F non è presente integralmente nello stesso BU, var blocc. si cercano nei BU di intersc.

Structure Directory



entry size è da intendersi come il numero di byte per arrivare alla prossima entry.

E' possibile usare per scaricare VOC (rendendo Unread).

Queste strutture sono chiamate framme interne (buchi interni) ma possono essere facilmente ricomposte queste direttive (-f /nurber anche la cache del Nics)

I primi quattro punti sono ovviamente gravi e questi dovrebbero
essere evitati (fatto 255 char)

Con ext 3 si dirizza a usare il journaling
(nella interno alla partizione si per la partizione
principale)

Con ext 4:

• introdotto extent per avere i-mode e long
variable (la voce contiene degli extent che
contengono i seg di block)

Così creano già contiguità.

Anche nelle versioni precedenti n. 4 sono tecniche
~~per~~ per creare già contiguità possibile in
partizione le stripe o
a un file serve un
lo contigui così i max
in modo contiguo)

• allocazione multi-block: il processo può richiedere
l'allocazione di tutti blocki, quindi non
extent

• allocazione (metto time out per evitare tick enorri)
ritornata: lavora la allocazione di extent
Quando il processo richiede 5 blocki (ad esempio)
il SO li prende in carico il contenuto e
rimanda l'allocazione dei 5 blocki.
Può essere ~~limitato~~ genioso (rimozione di
allocazioni e scrittura)

Pericolo sono i holes: può accadere
richieste (es: 5 file da 5 blocki) per
tutti insieme come un unico extent (16 blocki)
Perché rischia? se c'è crash durante questo
procedimento

• Un altro con che oggi oggi è **ReFS** (Refrigerator online)

BTRFS: "B-tree FS" o "butter FS"

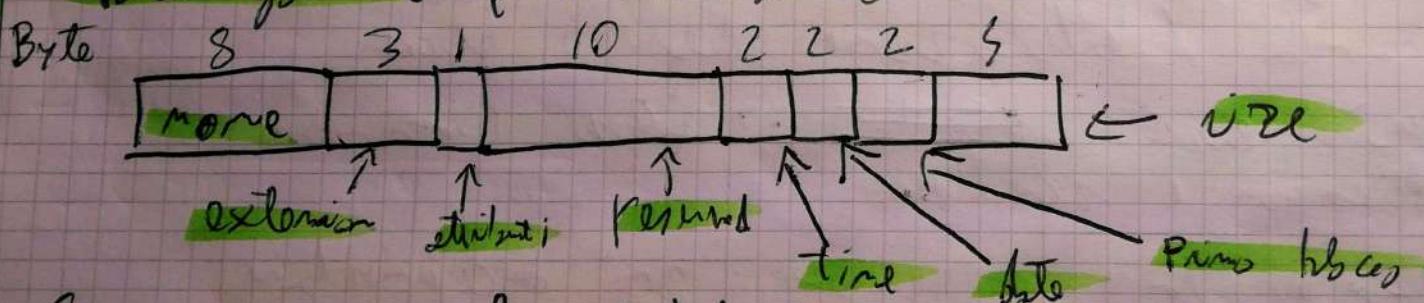
Un altro tipo di file system usato in Linux

Ajrage caratteristi che si prevedono FS Linux

- chiamazione di ogni file (e i parametri sono spesso copy-on-write nei file)
- sottovolumi
- maghiot di sottovolumi (con copy on write)
- non ricevere dati in maghiot
- API basata in B-tree
- Defragmentazione, aumento e riduzione online di volumi
- check sum per dati e metadati (per rilevare errori)
- compressione trasparente file
- meccanismo tipo RAID nel volume

File System MS-DOS (FAT) (non ridevo)

Voci per il file nella FAT:



ti dice gli numeri disponibili ai blocchi.

FAT-12 (12 bit) FAT-16 FAT-32

altrettanto: exFAT (64)

File system NTFS (non tradotto)

- Nome con Windows NT: è avanzato e complesso
- Volume: una o più partizioni o di cui
- cluster
- Master file table composta da record
- file: collezione di attributi vari
 - metadati o flusso (di dati)
 - persistenti o non residenti
 - file può richiedere più record
- API basata su B+Tree (ricerche efficienti)
- block liberi gestiti con bit map
- hard-link e soft-link e montaggio altri FS
- journalizing
- compressione e crittografia file
- copy shadow di volumi (modo copy-on-write)

Scheduling del Dico

- minimizzare throughput
- minimizzare tempo medio di accesso

Il SO raggiunge questo obiettivo facendo degli assegnamenti.

Si vedono gruppi di richieste condivise sul Dico.
Le richieste si mettono nello stato delle richieste pendenti che è nel controller del Dico.

Il SO abba varie politiche di selezione delle richieste per prendere la prossima da eseguire.

La sua già superficie da fare dovrebbe soddisfare le richieste nell'ordine in cui arrivano ma se il SO sceglie in modo opportuno potrebbe raggiungere gli obiettivi (o almeno in modo approssimativo).

Selezionare in modo opportuno sarà come conseguire:

- ridurre tempo di piavimento (seek time)
- ridurre latenza di lettura

Ottimizzando il seek-time (il seek-time è la parte già aperta)

Le richieste in ordine di arrivo e # di cilindri

98 183 37 122 15 125 65 67

per iniziale tenere: cilindro 53

Cilindri: rappresentazione delle tracce

Tracce rosse: non opposte, formano
logamente un cilindro

Dunque le due richieste hanno # cilindri uguale il seek time è 0

E potrebbe immagazzinare che organizziamo



le richieste in gruppi, dove le richieste di un gruppo formano lo stesso # cilindro.
Colgono seek-time per la lista di grida.

53. 98 183 37 122 15 125 65 67
inizio

FCFS: first come first served

seek time totale = 690 trace jucorre

53 - 98 + 198 - 183 + 183 - 37 + ... 165 - 67

Nb. FCFS è semplice, ma non inefficiente

SSTF: shortest seek time first (consiglio per es: ottimizzazione delle richieste)

Nella fl. diversi nella coda, si sposta nel giro vicino, salvo la ricerca e riconcilia.

Dal 53 valo a 65, poi 67 e così via...

53 → 65 → 67 → 37 → 15 → 98 → 122 → 125 → 183

seek time totale: 236 trace jucorre.

NB: i kerne re le richieste sono nate a tempo 0, se arrivano a tempo t sono di corrispondere in molti atti.

Il ritardo non è ego, crea l'arrivo di richieste (ovvero richieste inviate da clienti che sono già partiti).

NB $\frac{0}{T} \dots \frac{199}{T}$ trace estre

Scheda di funzione SCAN

(dopo l'ultimo
della funzione)

La testina ha un solo di funzione

$$0 \leftarrow \dots \rightarrow 199$$

Si continua funzione della testina non dopo aver
tracciato ~~o~~ un estremo

Dunque all'inizio effettua una posizione iniziale e
un passo della testina.

(53, \leftarrow) subito a 37, poi a 15, dopo va
ancora verso ~~15~~ e raggiunge 0.
Raggiunto 0 il passo si trasforma in dx.

Dunque la sequenza del sovrapposito è:

53 37 15 0 65 67 98 122 125 183
 \uparrow
inizio

Fare una rimozione uniforme

Garantisce la stessa manovra per ogni traccia, ma
è più lente di modo

Dunque il worst case per una ricchezza t di avere
una latenza di max equivalente al passo della
lateralità del disco (se il binario è lungo 200
tracce una ricchezza segnala max 400 tracce percorso)

Variante C-SCAN : rimozione circolare

Considerare le posizioni 0 e 199 come collegate
 $199 \rightarrow 0$

Se arriva a 199 posiziona la testina al cilindro
0 (ignorando i ricchiamenti intermedii)

Con le liste di giri si ottiene questo ordine

53 65 67 98 122 125 183 199 0 15 37

Il C-S CAN rende gli stadi più imballo con
ritardori in cui arrivano a tempo ridotti formazione
regalo

Grazie a un tempo medio di attesa più uniforme

Il C-S CAN ~~raggi~~ non ha c'è allo stesso, se ci
sono canali è meglio il SCAN monodirezionale

Ottimizzazioni SCAN

SCAN → LOOK

C-S CAN → C-LOOK

Ottimizzazione consiste nell'estate di orbite agli
estremi del disco se non è necessario

Nello SCAN si gira invece di fare $15 \rightarrow 0$ e
~~poi~~ $0 \rightarrow 65$, farsi invece $15 \rightarrow 65$

(ovvero si ricorda con # gli spazi e soprattutto
come si fanno le curve) si riceverà per il dx)

Nel C-SCAN si gira invece di fare $183 \rightarrow 199$,
 $199 \rightarrow 0$ e $0 \rightarrow 15$ farsi direttamente

$183 \rightarrow 15 \rightarrow 37$

(ovvero 183 e 15 è soprattutto come gli estremi)

Altri metodi:
• C-LOOK per allo canali
• LOOK o SSTF per bassi canali

Esercizio

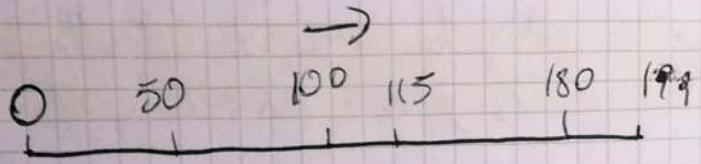
200 tracce (0...199) - velocità media = 1 m/s

a t=0 siamo alla traccia 100

Inoltre 50 115 180, a t=70 arriva traccia
per 150

a t=130 per 90

Calcolare tempo media confezione
utilizzando politica look
(verso iniziale \rightarrow)



$t=0$ coda: 30, 115, 180

$t=70$ 150
 $t=130$ 90

$t=15$ coda 50, 180

Poi sollecitazioni 180 al $t=80$

$t=80$ coda 50, 150 (~~150 non è stata sollecitata perché (150 è arrivata nel frattempo)~~)

Almeno uno interno (fino a 150)

$t=110$ (80 + (180 - 150)) coda 50

Vado ora verso 50

$t=210$ (110 + (50 - 50)) coda: 90 (e' arrivata nel frattempo)

Vado verso 90 (ho arrivato lì)

$t=250$ (210 + (90 - 50))

Seek totale = 250 ms

Sistemi RAID (oltre 20 anni usati per protezione dati in più processori per funzionalità)

Abbocca la ragionevole protezione già oltre, basati sullo 1° livello il parallelo

E se non ce n'è uno già di cui interamente (almeno 2)

Attribuzione: creare un unico volume logico che in realtà è implementato su N dischi fisici

L'attribuzione consiste dunque nell'usare già molti dischi per rappresentare in modo logico un unico volume logico

N.B.: mettendo in corso una rotazione da fine agli estremi

RAID: Redundant Array of Inexpensive Disks
o sono più in grado di creare un ottimo

Poi in realtà la cosa non era del tutto vero e quindi è cominciato l'acronimo: Redundant Array of Independent Disks

Infatti i dischi potevano lavorare in modo parallelo e indipendente

Questo modello permette di utilizzare i dati relativi ad una unità logica (file) su due dischi.

Ci dice questa tecnica striping

La suddivisione è trasparente all'utente

Il modello si può implementare via HW (e il SO non ha né scorge rispetto) o via SW (lavora applicato alla CPU)

Tecnica striping: tecnica di base per creare parallelismo

L'idea è partizionare il file in pezzi/linee
che vengono scritti in parallelo sul disco

Ogni stripe è memorizzato su diversi dischi. Se ho

5 stripes e 5 dischi: 1^o stripe su 1^o disco,
2^o stripe su 2^o disco, 3^o stripe su 3^o disco,
4^o stripe su 4^o disco e 5^o stripe su 5^o disco.

Applicazione usata da Robert Baldwin per il salvataggio degli stripe.

Se poi voglio leggere l'intero file ci metterebbero un quanto del tempo rispetto a un solo disco.

Esempio: transfer rate di 1 MB/s

Pur leggendo un file di 5 MB con il singolo disco ci impiegherei 5 secondi, così se faccio RAID 5 records

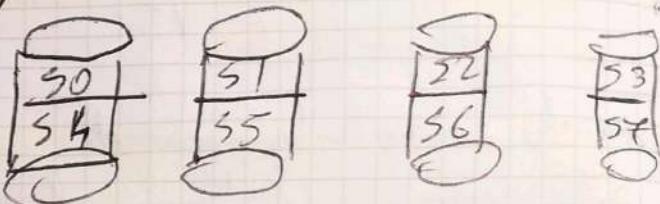
~~1~~ 1 5 dischi lavorano insieme in parallelo.

Il lavoro fatto sul file viene scritto sul Volume logico (insieme di byte)

Ese: Volume logico di 5 TB \rightarrow 5 volumi fisici da 1 TB

RAID 0 (striping in N disk)

- striping con Round Robin (non è ragionevole nel file ma sul volume logico)



Volume logico					
S0	S1	S2	S3	S4	S5...

Le richieste al S0 di leggere info sul Volume

logico vengono rispondute al leggere di tutte sul Volume fisico

• Caso ottimo: stripe richiesti alternati sui diversi dischi (speed up $\times 5$) (es: 8 richieste \rightarrow 2 stripes in ogni disco)

• Caso peggiore: tutti gli stripe richiesti stessi sullo stesso disco (speed up $\times 1$, stesso δ tempo di risposta)

• Caso medio: è già vicino al Caso peggiore
(Dato che $\frac{1}{5}$ delle richieste)

Il RAID 0 non aggiunge nessuna robustezza al sistema dei quattro hard disk perché il volume logico ha la stessa probabilità di fallimento.

Molti preferiscono l'uso di due zone = 10 dischi utili, ma ne ha 8. Le 2 degli 8 si trovano entro i 4 dischi gli altri 2.

Proteggere dai guasti? Aggiungere robustezza

RAID 1 (mirroring)

- al massimo ragione anche degli errori (mirroring)
- si crea la copia speculare del volume logico

S0	S1	S2	S3	S4	S5	S6	S7	S8
S5	S5	S6	S7	S3	S3	S6	S7	S4

Ho 8 dischi fissi frangere un volume logico in 8 nuclei fissi duplicati (il volume logico (max 5 TB))

- Questa tecnica di mirroring è più utile anche senza striping
- migrazione fault tolerance (non è perfetta)
- Ho un alto overhead di storage (mettere una striscia su un solo volume mettendo un simbolo diverso)

Le gestioni sul cosa di ricaricare i vari stripes sono costantemente

- Cosa migliore: ~~file~~ ^{triste} isolati here sui 8 dischi (e quindi anche sulla copia)
- Cosa peggiore: ~~file~~ ^{triste} concentrati su un singolo disco (quindi anche sulla sua copia)

$$\text{bytes up} = \times 2$$

- Cosa medio: bytes up tra 2 e 8

Per le ~~file~~ ^{lettura} vale appunto il binomio null' over Head di storage

- ~~Cosa pessima~~: ~~file~~ ^{triste} $\times 3$ (a causa dell'isolamento)

NB: RAID n'jmi fare anche con 2 dischi

RAID: probabilmente no un troppi dischi

Raid 2 Mixing a Circuito di bit con ECC

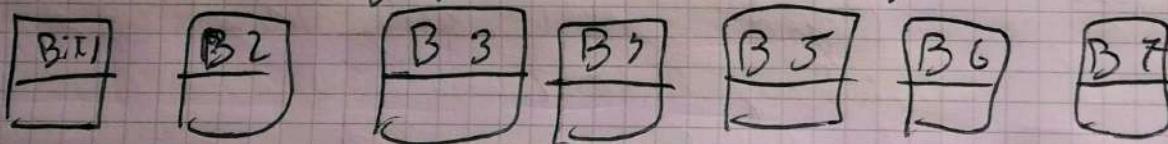
Si usc la tecnica del bit mixing e si applicano le tecniche di generazione e rilevazione di errori.

Il RAID 2 lavora ai byte significa un codice di correzione per carri

- Volume logico di 7 dischi fisici
- Parola di 5 bit \rightarrow 7 bit per Hamming (3 bit ribaditi)
- I 7 bit sono di dimensioni di 7 blocchi
- Da un punto di vista logico ci sono circa 5TB
ma: esistono 7 invece di 8.
- Ha un'ottima fault tolleranza e ha buone prestazioni per leggere e scrivere righe

Problema: serve sincronizzare perfettamente i blocchi
cioè se anche solo 1 disco fisico fa un fallo, la lettura dell'intero volume
è fallimentata.

Potrei? puo memorizzare i bit, per leggerne ^{una parola} byte
mi serve usare tutti i 7 dischi
mentre prima le parole le mettevamo
tutte raggruppate sulle righe



Per leggere la parola B1 B2 B3 B4 usiamo tutte e
7 le righe (3 ribadimenti)

Questo problema viene risolto facendo mixing a livello
di bit insieme che a livello di byte / blochi.
(maggioranza delle ~~caso~~ si bit cells word)

(In raid 0 e 1 negli stage invece maggiore i
block)

RAID 3

storing a byte of bit in a word
bit di parità
Ese: 5 bytes fino + 1 da bit di parità



parità

Se i gradi di blocchi sono > 1 e quindi oltre a queste informazioni serve il bit di parità per bloccare il valore di B5 nonostante il byte sia tutto. Dunque fatti si apre tutto questo raid 2 ma con 2 dischi in meno.

Anche qui siamo che non a livello di bit ci sono i problemi di sincronizzazione tra bit

Dunque: striping + livello di bit \rightarrow inefficiente
Nulla mutua RAID 2 e 3 non è usato

RAID 5 striping + livello di blocchi con XOR
sull'ultimo disco

- solo stripe a blocchi
- ci sono blocchi extra de bit XOR degli stripe

$$\text{Quindi } P0-3 = S_0 \otimes S_1 \otimes S_2 \otimes S_3$$



volume logico

- L'ultimo stripe è la ridondanza per la fault tolerance (ne abbiamo 3 e non 2, S3 è riconstruibile)
- Non è bastato un bit quindi non meccanica di incisiva

Prestazioni lettura: uguali a RAID 0

Caso niente: x_3
caso regione: x_1

Prestazioni scrittura: gran perdita inefficienza, moltiplicare con stripe poiché dobbiamo riconstruire il blocco di ridondanza quando tutti gli XOR

Ottimizzazione: $P0-3$ è già calcolato come $P0-3 \otimes S_3 \otimes S_3$
ne è S_3 che calcola (in S_3)

Dunque l'aggiornamento di P_{0-3} è in genere fatto da un solo ribaditore non chiede tutti gli altri

Saranno i vecchi blocchi di ribadimento (P_{0-3})

$$\text{vecchio blocco di ribadimento } (S_3) \quad P_{0-3} = P_{0-3} \otimes S_3 \otimes S_3$$

$$\text{a nuovo blocco dati } (\overline{S_3})$$

Dunque con questa tecnica l'aggiornamento non è così inefficiente

NB: l'ultimo blocco (quello di ribadimento) è il più grande / usato blocco che è sincronizzato ogni 200ms

In pratica l'ultimo blocco si usa una

Risolviamo questo problema con **RAIDS**

- Utilizzando diversi blocchi con informazioni ribadimenti distribuite

Funziona come RAID 5 ma il file di contenzione di info ribadimenti è distribuito a tutti i dischi

S_0	S_1	S_2	S_3	P_{0-3}
S_4	S_5	S_6	P_3-F	S_7
S_8	S_9	P_8-II	S_{10}	S_{11}
S_{12}	P_{12-B}	S_{13}	S_{14}	S_{15}
P_{16-I}	S_{16}	S_{17}	S_{18}	S_{19}

In termini di feiabilità a quanti è identico a RAID 5

I blocchi sono in media sparsi sul tempo
modo quindi non ci sono blocchi che si usano
in modo più veloce

Permette una maggiore anche delle letture rispetto
a RAID 5 uno speed up $\times 5$

Per la scrittura i fatti le stesse considerazioni di
RAID 5 e bisogna non già di tanto inefficiente

Memoire Flash e file system

Dispositivi basati su flash (NAND o NOR) formidabili
in modo diverso dai dischi elettronici.

- Un blocco può dunque essere riscritto bene senza cancellare.
- Il numero di cancellazione (e quindi scrittura) di un blocco è l'indirizzo (per una questione proprio finita, subendo oltre il blocco in quota)
- Il singolo blocco è composto da pagine; così le operazioni di lettura / scrittura si riferiscono a pagina. PERO se devo cancellare / scrivere una pagina dovrei cancellare l'intero blocco, dunque si evita la cancellazione (finché non).

A ₁	A ₂	A ₃

Dopo modificare A₂ → Ā₂.

Se poi vado a sostituire A₂ con Ā₂ dovrei cancellare tutto il blocco e rimettere anche A₁ e A₃.

Dunque il controller trova una nuova cellula vuota e ci mette Ā₂ e nega la vecchia A₂ come "non valida".

Il SO riconosce se ne accorge, lo (ri)protegge e fa il controller.

Se il blocco è pieno e molte pagine sono "non valide"?

E se cancella il blocco e poco prima lo ha riscritto in un altro blocco (sia solo le valide). In questo caso ho reagito alla pagina pagando una cancellazione.

Problema: cancella file → SO lo sa ma il controller no, quindi il Garbage collector continua a considerare valida le pagine del multiblock file.

Garbage collector: parte che cerca blocco alla fine

Nelle pagine viste ecc.

Sostanzialmente il controller non memorizza informazioni sul SO quindi cancellazione di file continua a fare garbage collection su file.

Infatti oggi si usa la TRIM che funziona molto bene perché comunica al controller i blocchi cancellati.

Così ottimo miglioramento nel tempo del file

- Garbage collection → blocco Nove (non usano pagine relative a file cancellati)
- Garbage collection → permette di rigenerare pagine con TRIM

Dunque il SO deve adeguarsi a questo tipo di file (mentre il meccanismo di TRIM)

Di solito non ci sono file system dedicati a queste memorie

- Flash Friendly File system è nato in tentativo di preferire l'abegamento da parte del SO

NB: file system visto precedentemente conserva senza lavorare il fatto che ogni blocco abbia un limite fino a cancellazioni impossibili.