

CAPITOLO

I/O

OBIETTIVI DEL CAPITOLO

- Descrizione della struttura fisica dei dispositivi di memorizzazione secondaria e degli effetti che ne derivano per il loro utilizzo.
- Spiegazione delle caratteristiche prestazionali della memoria secondaria.
- Analisi degli algoritmi di scheduling delle unità a disco.
- Analisi dei servizi offerti dal sistema operativo per la memorizzazione secondaria, tra cui il RAID.

Memoria secondaria

Il file system, da un punto di vista logico, si può considerare composto da tre parti: nel Capitolo 11 sarà presentata l'interfaccia per il programmatore e per l'utente del file system; nel Capitolo 12 descriveremo le strutture dati interne e gli algoritmi usati dal sistema operativo per realizzare quest'interfaccia; nel presente capitolo si comincia l'analisi del file system dal livello più basso: la struttura della memoria secondaria. Si descrivono innanzitutto la struttura fisica dei dischi e dei nastri magnetici. Vengono poi discussi gli algoritmi di scheduling delle unità a disco, che ordinano la sequenza delle operazioni di I/O al fine di massimizzarne le prestazioni. Quindi si illustrano la formattazione dei dischi e la gestione dei blocchi d'avviamento, dei blocchi danneggiati e dell'area d'avvicendamento dei processi. Si analizza infine la struttura dei sistemi RAID.

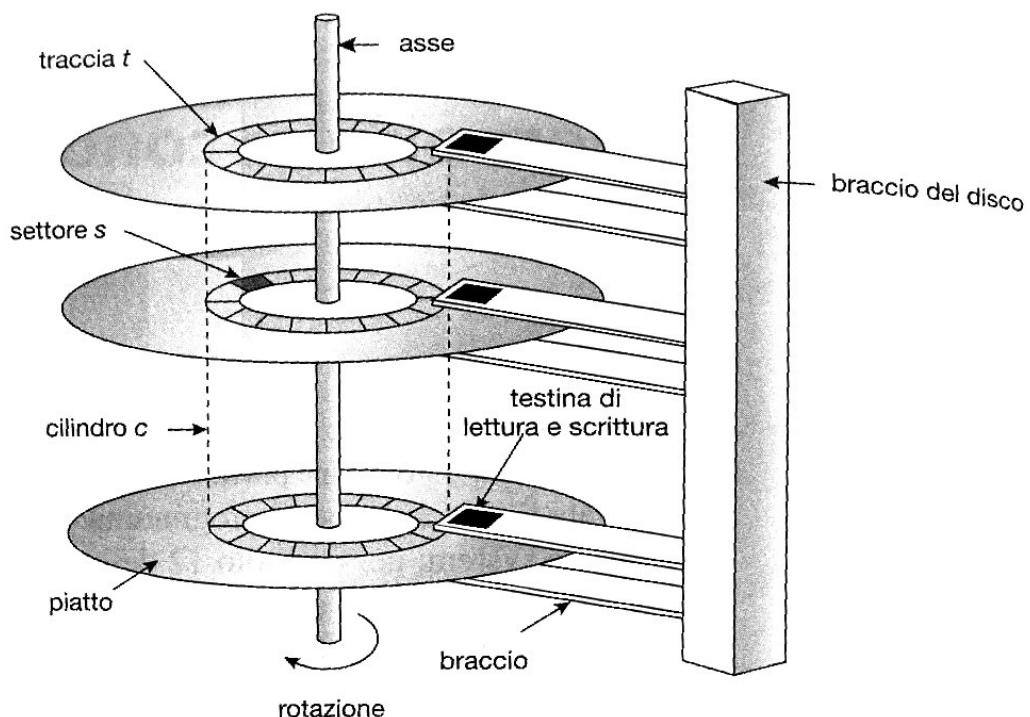


Figura 10.1 Schema di un disco a testine mobili.

10.1 Struttura dei dispositivi di memorizzazione

In questo paragrafo si presenta una rassegna generale della struttura fisica dei dispositivi di memorizzazione secondaria e terziaria.

10.1.1 Dischi magnetici

I **dischi magnetici** sono il supporto fondamentale di memoria secondaria dei moderni sistemi elaborativi. Concettualmente, i dischi (Figura 10.1) sono relativamente semplici: i **piatti** dei dischi hanno una forma piana e rotonda come quella dei CD, con un diametro che comunemente varia tra 1,8 e 3,5 pollici, e le due superfici ricoperte di materiale magnetico; le informazioni si memorizzano registrandole magneticamente sui piatti.

Una testina di lettura e scrittura è sospesa su ciascuna superficie d'ogni piatto. Le testine sono attaccate al **braccio del disco** che le muove in blocco. La superficie di un piatto è divisa logicamente in **tracce** circolari a loro volta suddivise in **settori**; l'insieme delle tracce corrispondenti a una posizione del braccio costituisce un **cilindro**. In un'unità a disco possono esservi migliaia di cilindri concentrici e ogni traccia può contenere centinaia di settori. La capacità di memorizzazione di una comune unità a disco si misura in gigabyte.

Quando un disco è in funzione, un motore lo fa ruotare ad alta velocità; la maggior parte dei dischi ruota a velocità comprese tra 60 e 250 giri al secondo. Questa velocità viene espresa in termini di giri al minuto (RPM): i comuni dischi possono lavorare alle velocità di 5400, 7200, 10000 o 15000 RPM. La velocità di un disco è caratteriz-

zata da due valori: la **velocità di trasferimento**, cioè la velocità con cui i dati fluiscono dall'unità a disco al calcolatore, e il **tempo di posizionamento**, detto anche tempo d'accesso casuale, che consiste di due componenti: il tempo necessario a spostare il braccio del disco in corrispondenza del cilindro desiderato, detto **tempo di ricerca** (*seek time*), e nel tempo necessario affinché il settore desiderato si porti, tramite la rotazione del disco, sotto la testina, detto **latenza di rotazione**. In genere i dischi possono trasferire parecchi megabyte di dati al secondo e hanno un tempo di ricerca e una latenza di rotazione di diversi millisecondi.

Poiché le testine di un disco sono sospese su un cuscino d'aria sottilissimo (dell'ordine dei micron), esiste il pericolo che la testina urti la superficie del disco; in tal caso, nonostante i piatti del disco siano ricoperti da un sottile strato protettivo, la testina può danneggiare la superficie magnetica. Tale incidente, detto **urto della testina**, di solito non può essere riparato e comporta la sostituzione dell'intera unità a disco.

Un disco può essere **rimovibile**: ciò permette che diversi dischi siano montati secondo le necessità. I dischi magnetici rimovibili consistono generalmente in un piatto contenuto in un involucro di materiale plastico, che serve a evitare danni che possono verificarsi quando il disco non è inserito nella propria unità. Sono dischi rimovibili anche i CD, i DVD, i Blu-ray e i dispositivi di memoria flash (che costituiscono una tipologia di drive a stato solido). Un'unità a disco è connessa a un calcolatore attraverso un insieme di fili detto **bus di I/O**; esistono diversi tipi di tale bus, tra i quali i bus ATA (*advanced technology attachment*), SATA (*serial ATA*), eSATA, USB (*universal serial bus*) e FC (*fiber channel*). Il trasferimento dei dati in un bus è eseguito da processori dedicati detti **controllori**: i **controllori di macchina** (*host controller*) sono i controllori posti all'estremità del bus lato calcolatore; i **controllori dei dischi** (*disk controller*) sono incorporati in ciascuna unità a disco. Per eseguire un'operazione di I/O il calcolatore inserisce un comando nell'adattatore, generalmente mediante porte di I/O mappate in memoria, com'è descritto nel Paragrafo 9.7.3; l'adattatore invia il comando mediante messaggi al controllore del disco, che agisce sull'hardware dell'unità a disco per portare a termine il comando. I controllori dei dischi di solito hanno una cache incorporata: il trasferimento dei dati nell'unità a disco avviene tra la cache e la superficie del disco; il trasferimento dei dati tra la cache e l'adattatore avviene alla velocità propria dei dispositivi elettronici.

10.1.2 Dischi a stato solido

A volte le vecchie tecnologie sono utilizzate in modo nuovo grazie ai cambiamenti economici e all'evolversi delle tecnologie stesse. Un esempio è la crescente importanza dei **dischi a stato solido**, o SSD. In breve, un SSD è una memoria non volatile che viene utilizzata come un disco rigido. Ci sono molte varianti di questa tecnologia, dalle DRAM dotate di batteria per permettere di mantenere lo stato in caso di caduta di tensione alle tecnologie di memoria flash come i chip SLC (*single-level cell*) e MLC (*multilevel cell*).

Gli SSD hanno le stesse caratteristiche dei dischi rigidi tradizionali, ma possono essere più affidabili, perché non hanno parti in movimento, e più veloci, perché non hanno tempo di ricerca o di latenza. Inoltre consumano meno energia. Tuttavia il costo al megabyte è superiore rispetto ai dischi rigidi tradizionali, hanno meno capacità rispetto ai dischi rigidi più grandi e possono avere una durata di vita più breve, quindi il loro uso resta un po' limitato. Un utilizzo degli SSD è negli storage array, per mantenere i metadati del file-system che richiedono prestazioni elevate. Vengono utilizzati anche in alcuni computer portatili, per renderli più piccoli, più veloci e più efficienti.

Poiché gli SSD possono essere molto più veloci rispetto alle unità disco magnetiche, le interfacce bus standard possono costituire un notevole limite per il throughput. Alcuni SSD sono progettati per essere collegati direttamente al bus di sistema (PCI, per esempio). Gli SSD stanno cambiando altri aspetti tradizionali nel progetto dei computer. Alcuni sistemi li usano in sostituzione delle unità disco tradizionali, mentre altri li usano come un nuovo livello di cache, spostando i dati tra dischi magnetici, SSD e memoria per ottimizzare le prestazioni.

Nel resto di questo capitolo alcuni paragrafi hanno a che fare con gli SSD, mentre altri non li riguardano. Per esempio, poiché gli SSD non hanno una testina, la maggior parte degli algoritmi di scheduling non si applica a questi dispositivi. I paragrafi che trattano di throughput e formattazione riguardano invece anche gli SSD.

10.1.3 Nastri magnetici

I **nastri magnetici** sono stati i primi supporti di memorizzazione secondaria. Pur avendo la capacità di memorizzare in modo permanente un'enorme quantità di dati, queste unità sono caratterizzate da un tempo d'accesso molto elevato rispetto a quello della memoria centrale e dei dischi magnetici. Inoltre il tempo d'accesso casuale dei nastri magnetici (essendo fisicamente ad accesso sequenziale) è un migliaio di volte maggiore di quello dei dischi magnetici, e ciò li rende inadatti come supporto di memoria secondaria. Gli usi principali dei nastri sono la creazione di copie di backup dei dati, la registrazione di dati poco usati e il trasferimento di informazioni tra diversi sistemi elaborativi.

Il nastro è avvolto in bobine e scorre su una testina di lettura e scrittura. Il posizionamento sul settore richiesto può richiedere alcuni minuti, anche se, una volta raggiunta la posizione desiderata, l'unità a nastro può leggere o scrivere informazioni a una velocità paragonabile a quella di un'unità a disco. La capacità varia secondo il particolare tipo di unità a nastro. I nastri di oggi hanno una capacità di diversi terabyte. Alcune unità sono dotate di funzionalità di compressione dei dati che permettono di più che raddoppiare la capacità effettiva. Le unità a nastro e i loro driver sono solitamente classificate per larghezza: misure tipiche sono 4, 8 o 19 millimetri, e 1/4 o 1/2 pollice. Alcune unità prendono il nome dalla tecnologia su cui si basano, come nel caso di LTO-5 e SDLT.

VELOCITÀ DI TRASFERIMENTO DEL DISCO

Come per molti altri aspetti dell'informatica, le prestazioni pubblicate relative ai dischi non coincidono con le cifre reali: sono sempre inferiori delle **velocità di trasferimento effettive**, per esempio. La velocità di trasferimento può essere considerata la velocità con cui la testina legge i bit dal supporto magnetico, ma questa va distinta dalla velocità con cui i blocchi sono consegnati al sistema operativo.

10.2 Struttura dei dischi

I moderni dischi magnetici sono indirizzati come grandi array monodimensionali di **blocchi logici**, dove un blocco logico è la minima unità di trasferimento. La dimensione di un blocco logico è di solito di 512 byte, sebbene alcuni dischi si possano **formattare a basso livello** allo scopo di ottenere una diversa dimensione dei blocchi logici, per esempio 1024 byte; per altre informazioni su quest'opzione, si veda il Paragrafo 10.5.1. L'array monodimensionale di blocchi logici è mappato in modo sequenziale sui settori del disco: il settore 0 è il primo settore della prima traccia sul cilindro più esterno; la corrispondenza prosegue ordinatamente lungo la prima traccia, quindi lungo le rimanenti tracce del primo cilindro, e così via di cilindro in cilindro dall'esterno verso l'interno.

Sfruttando questa corrispondenza sarebbe possibile – almeno in teoria – trasformare il numero di un blocco logico in un indirizzo fisico di vecchio tipo, consistente in un numero di cilindro, un numero di traccia concernente quel cilindro, e un numero di settore relativo a quella traccia. In pratica, però, vi sono due motivi che rendono difficile quest'operazione: in primo luogo, la maggior parte dei dischi contiene settori difettosi, ma la corrispondenza nasconde questo fatto sostituendo ai settori malfunzionanti settori sparsi in altre parti del disco; in secondo luogo, il numero di settori per traccia in certe unità a disco non è costante.

Nei supporti che impiegano la **velocità lineare costante** (*constant linear velocity*, CLV) la densità di bit per traccia è uniforme. Più è lontana dal centro del disco, tanto maggiore è la lunghezza della traccia, tanto maggiore è il numero di settori che essa può contenere. Spostandosi da aree esterne verso aree più interne il numero di settori per traccia diminuisce. Le tracce nell'area più esterna contengono in genere il 40 per cento in più dei settori contenuti nelle tracce dell'area più interna. L'unità aumenta la sua velocità di rotazione man mano che le testine si spostano dalle tracce esterne verso le tracce più interne per mantenere costante la quantità di dati che scorrono sotto le testine. Questo metodo si usa nelle unità per CD-ROM e DVD. In alternativa la velocità di rotazione dei dischi può rimanere costante, e la densità di bit decresce dalle tracce interne alle tracce più esterne per mantenere costante la quantità di dati che scorre sotto le testine. Questo metodo si usa nelle unità a disco magnetico ed è noto come **velocità angolare costante** (*constant angular velocity*, CAV).

Il numero di settori per traccia cresce con l'evoluzione della tecnologia dei dischi, e l'area più esterna di un disco di solito contiene diverse centinaia di settori per trac-

cia. Anche il numero di cilindri è andato aumentando; le grandi unità a disco contengono decine di migliaia di cilindri.

10.3 Connessione dei dischi

I calcolatori accedono alla memoria secondaria in due modi: nei sistemi di piccole dimensioni il modo più comune è tramite le porte di I/O (**memoria secondaria connessa alla macchina**); oppure ciò avviene tramite un host remoto in un file system distribuito (**memoria secondaria connessa alla rete**).

10.3.1 Memoria secondaria connessa alla macchina

Alla memoria secondaria connessa alla macchina (*host-attached storage*) si accede dalle porte locali di I/O. Queste porte sono disponibili in diverse tecnologie; i comuni PC impiegano un'architettura per il bus di I/O chiamata IDE o ATA. Quest'architettura consente di avere non più di due unità per ciascun bus di I/O. Un protocollo più moderno che prevede un cablaggio più semplice è il SATA.

Le stazioni di lavoro di fascia alta e i server impiegano architetture più raffinate come FC (*fibre channel*), un'architettura seriale ad alta velocità che può funzionare sia su fibra ottica sia su un cavo con 4 conduttori di rame. FC ha due varianti. La prima è una grande struttura di commutazione con uno spazio d'indirizzi a 24 bit. Per il futuro ci si aspetta che questo metodo prevalga, ed è la base per le **storage-area network** (*reti di memoria secondaria*), trattate nel Paragrafo 10.3.3. Grazie al vasto spazio d'indirizzi e alla natura commutata della comunicazione, si possono connettere più macchine e dispositivi di memorizzazione alla struttura a commutazione, permettendo una notevole flessibilità nella comunicazione di I/O. La seconda variante si chiama FC-AL (*arbitrated loop*) e può indirizzare fino a 126 dispositivi (unità e controllori).

C'è un gran numero di dispositivi utilizzabili come memoria secondaria connessa alla macchina, tra questi le unità a disco, le unità RAID, le unità a CD, DVD e a nastri magnetici. I comandi di I/O che avviano trasferimenti di dati a un dispositivo di memoria connessa alla macchina sono letture e scritture di blocchi logici di dati, dirette a un'unità di memorizzazione specificamente identificate (per esempio, tramite bus IDE e unità logica del dispositivo).

10.3.2 Memoria secondaria connessa alla rete

Un dispositivo di memoria secondaria connessa alla rete (*network-attached storage*, NAS) è un sistema di memoria specializzato al quale si accede in modo remoto per mezzo di una rete di trasmissione di dati (Figura 10.2). I client accedono alla memoria connessa alla rete tramite un'interfaccia RPC, come l'NFS nel caso dei sistemi UNIX o CIFS nel caso di sistemi Windows. Le chiamate di procedura remota (RPC) sono realizzate per mezzo dei protocolli TCP o UDP sopra una rete IP (di solito la stessa rete locale che porta tutto il traffico dati ai client). Può quindi risultare più semplice per-

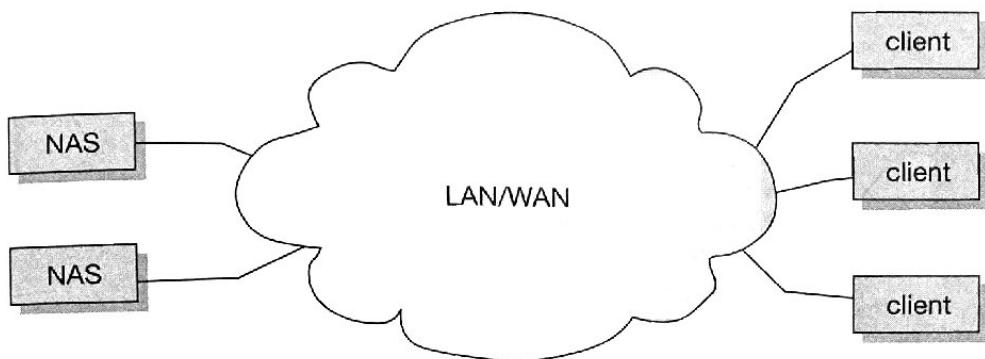


Figura 10.2 Memoria secondaria connessa alla rete.

sare a NAS come a un ulteriore protocollo di accesso alla memoria secondaria. L'unità di memoria è normalmente realizzata come una batteria RAID con programmi di controllo che implementano l'interfaccia per le RPC.

La memoria secondaria connessa alla rete fornisce un modo semplice per condividere spazio di storage per tutti i calcolatori di una LAN, con la stessa facilità di gestione dei nomi e degli accessi caratteristica della memoria secondaria connessa alla macchina. Tuttavia, un sistema di questo genere tende a essere meno efficiente e ad avere prestazioni inferiori rispetto ad alcuni sistemi con connessione diretta alla macchina.

iSCSI è il più recente protocollo per la memoria connessa alla rete. Essenzialmente sfrutta il protocollo IP della rete per il trasporto del protocollo SCSI. Ne consegue la possibilità di usare cavi di rete invece che cavi SCSI per connettere le diverse macchine alla memoria secondaria. Uno dei vantaggi di questa tecnica è che le macchine sono in grado di trattare la memoria secondaria come se fosse direttamente collegata, sebbene possa essere collocata a distanza.

10.3.3 Storage-area network

Uno svantaggio dei sistemi di memoria secondaria connessa alla rete è che le operazioni di I/O sulla memoria secondaria impegnano banda della rete e quindi aumentano la latenza della comunicazione nella rete. Questo problema può essere particolarmente grave per sistemi client-server di grandi dimensioni: l'ordinaria comunicazione tra i server e i client compete per la banda con la comunicazione tra i server e i dispositivi di memorizzazione.

Una storage-area network (SAN) è una rete privata (che impiega protocolli specifici per la memorizzazione anziché protocolli di rete) tra i server e le unità di memoria secondaria (Figura 10.3). La potenza di una SAN sta nella sua flessibilità: si possono connettere alla stessa SAN molte macchine e molti storage array; la memoria può essere allocata alle macchine dinamicamente. Uno switch SAN nega o concede alle macchine l'accesso alla memoria secondaria. Per fare un esempio, è possibile configurarlo di modo che allochi ulteriore memoria secondaria alle macchine che stanno esaurendo lo spazio sui propri dischi. Questi switch rendono anche possibile la condivisione del-

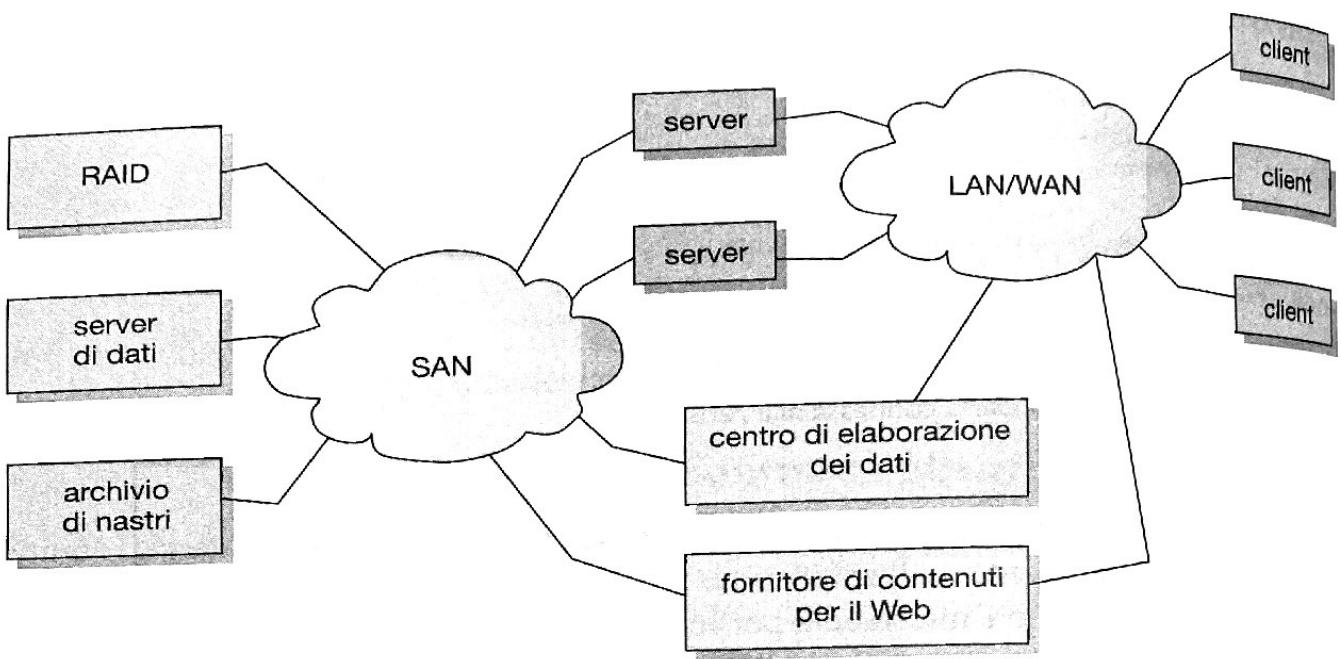


Figura 10.3 Storage area network.

la memoria di massa da parte di cluster di server, e il collegamento diretto di più macchine alla memoria di massa. Gli switch SAN hanno spesso porte in maggior numero, oltre che più costose, dei normali storage array.

La soluzione più comune per il collegamento tramite SAN è il FC, anche se la diffusione di iSCSI è in crescita, grazie alla sua semplicità. Una possibile alternativa è InfiniBand, un'architettura di bus specifica per SAN, che fornisce supporto hardware e software per reti d'interconnessione ad alta velocità tra server e unità di memoria secondaria.

10.4 Scheduling del disco

Una delle responsabilità del sistema operativo è quella di fare un uso efficiente dell'hardware. Nel caso delle unità a disco, far fronte a questa responsabilità significa garantire tempi d'accesso contenuti e ampiezze di banda elevate. Nel caso dei dischi magnetici il tempo d'accesso si può scindere in due componenti principali, come menzionato nel Paragrafo 10.1.1: il **tempo di ricerca** (*seek time*), cioè il tempo necessario affinché il braccio dell'unità a disco sposti le testine fino al cilindro contenente il settore desiderato, e la **latenza di rotazione** (*rotational latency*), e cioè il tempo aggiuntivo necessario perché il disco ruoti finché il settore desiderato si trovi sotto la testina. L'**ampiezza di banda** (*bandwidth*) del disco è il numero totale di byte trasferiti diviso il tempo totale intercorso fra la prima richiesta e il completamento dell'ultimo trasferimento. Gestendo l'ordine delle richieste di I/O relative al disco si possono migliorare sia il tempo d'accesso sia l'ampiezza di banda.

Ogni volta che deve compiere operazioni di I/O con un'unità a disco, un processo effettua una chiamata di sistema.

La richiesta contiene diverse informazioni:

- se l'operazione è di input o di output;
- l'indirizzo nel disco per il trasferimento;
- l'indirizzo di memoria per il trasferimento;
- il numero di settori da trasferire.

Se l'unità a disco desiderata e il controllore sono disponibili, la richiesta si può immediatamente soddisfare; altrimenti le nuove richieste si aggiungono alla coda di richieste in evase relativa a quell'unità. La coda relativa a un'unità a disco in un sistema con multiprogrammazione può spesso essere piuttosto lunga, quindi, al completamento di una richiesta, il sistema operativo sceglie quale fra le richieste in evase conviene servire prima. Come il sistema operativo affronta tale scelta? Per mezzo di uno dei vari algoritmi di scheduling che presentiamo nel seguito.

10.4.1 Scheduling in ordine d'arrivo – FCFS

La forma più semplice di scheduling è, naturalmente, l'algoritmo di servizio secondo l'ordine d'arrivo (*first come, first served*, FCFS). Si tratta di un algoritmo intrinsecamente equo, ma che in generale non garantisce la massima velocità del servizio. Si consideri, per esempio, una coda di richieste per l'unità a disco che riguardino blocchi sui seguenti cilindri (nell'ordine):

98, 183, 37, 122, 14, 124, 65, 67.

Se si trova inizialmente al cilindro 53, la testina dell'unità a disco dovrà prima spostarsi al cilindro 98, poi al 183, 37, 122, 14, 124, 65 e infine al 67, per un movimento totale della testina, misurato in numero di cilindri visitati, di 640 cilindri. La sequenza è rappresentata nella Figura 10.4.

Le defezioni di quest'algoritmo sono evidenziate dal grande salto da 122 a 14 e poi di nuovo a 124: se le richieste per i cilindri 37 e 14 si potessero soddisfare in sequenza, la distanza totale percorsa diminuirebbe notevolmente e le prestazioni migliorerebbero di conseguenza.

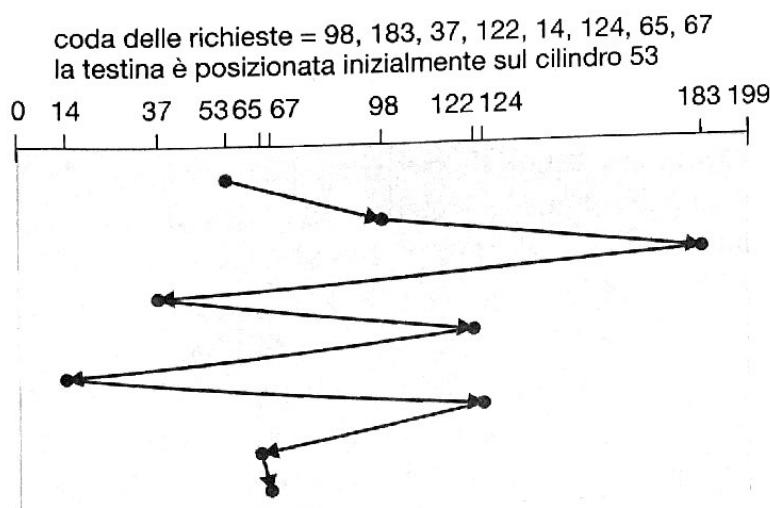


Figura 10.4 Scheduling FCFS.

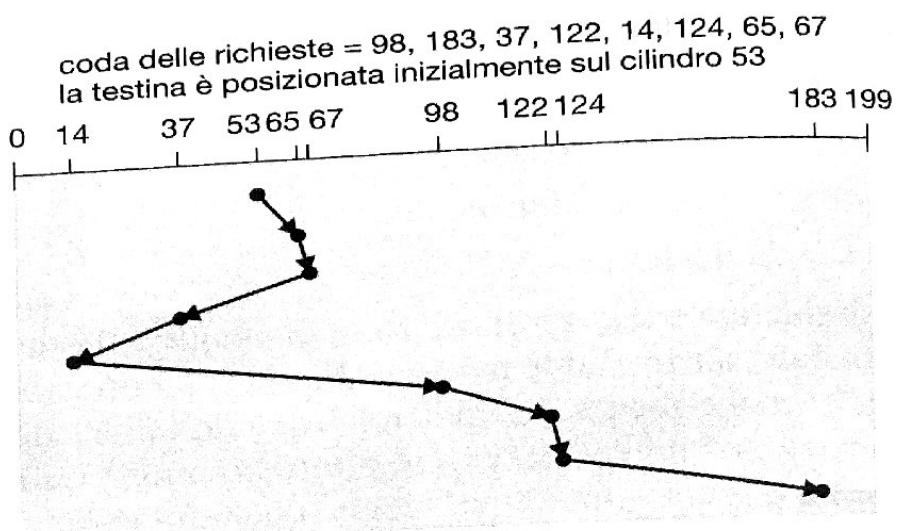


Figura 10.5 Scheduling SSTF.

10.4.2 Scheduling – SSTF

Sembra ragionevole servire tutte le richieste vicine alla posizione corrente della testina prima di spostarla in un'area lontana per soddisfarne altre: questa considerazione è alla base dell'**algoritmo di servizio secondo il più breve tempo di ricerca** (*shortest seek time first*, SSTF). SSTF sceglie la richiesta che dà il minimo tempo di ricerca rispetto alla posizione corrente della testina. In altre parole, l'algoritmo sceglie la richiesta relativa ai cilindri più vicini alla posizione della testina.

Se si considera nuovamente la sequenza di richieste dell'esempio sopra, il cilindro più vicino alla posizione iniziale della testina (cioè 53) è il 65, la successiva richiesta più vicina è quella relativa al cilindro 67; da questo cilindro, la richiesta relativa al cilindro 37 è più vicina di quella relativa al cilindro 98, ed è quindi servita per terza. Continuando allo stesso modo, sarà soddisfatta la richiesta relativa al cilindro 14, poi 98, 122, 124 e infine 183 (Figura 10.5). Questo metodo di scheduling implica un movimento totale della testina di soli 236 cilindri, poco più di un terzo di quello ottenuto con lo scheduling FCFS di questa coda di richieste: esso porta quindi sostanziali miglioramenti d'efficienza.

Lo scheduling SSTF è essenzialmente una forma di scheduling per brevità (*shortest job first*, SJF), e al pari di questo, può condurre a situazioni d'attesa indefinita (*starvation*) di alcune richieste. Si ricordi infatti che nuove richieste possono giungere in qualunque momento: si supponga di avere due richieste in coda, una per il cilindro 14 e l'altra per il 186, e che mentre si sta servendo la richiesta relativa al cilindro 14, arrivi una nuova richiesta per un cilindro vicino al 14. Questa sarà la prossima richiesta soddisfatta, mentre la richiesta per il cilindro 186 dovrà attendere. La stessa situazione potrebbe ripetersi, perché un'altra richiesta relativa a una posizione in prossimità del cilindro 14 potrà giungere mentre si sta servendo la precedente richiesta: in teoria, un flusso continuo di richieste riferite a posizioni vicine fra loro potrebbe causare l'attesa indefinita della richiesta relativa al cilindro 186. Quest'ipotesi diviene sempre più probabile man mano che la coda di richieste si allunga.

Sebbene l'algoritmo SSTF costituisca un miglioramento notevole rispetto al FCFS, esso non è ottimale: si può fare di meglio con uno spostamento dal cilindro 53 al 37, anche se questa non è la minima distanza possibile, e poi al 14, prima di invertire la marcia per servire 65, 67, 98, 122, 124 e 183. L'adozione di questa strategia riduce il movimento totale a 208 cilindri.

10.4.3 Scheduling – SCAN

Secondo l'**algoritmo SCAN** il braccio dell'unità a disco parte da un estremo del disco e si sposta verso l'altro estremo, servendo le richieste mentre attraversa i cilindri, finché non completa il tragitto: a questo punto, il braccio inverte la marcia, e la procedura continua. Le testine attraversano continuamente il disco nelle due direzioni. L'algoritmo SCAN è a volte chiamato **algoritmo dell'ascensore**, perché il braccio dell'unità a disco si comporta proprio come un ascensore che serva prima tutte le richieste in salita e poi tutte quelle in discesa.

Si consideri ancora l'esempio precedente. Prima di poter applicare lo scheduling SCAN alle richieste per i cilindri 98, 183, 37, 122, 14, 124, 65, e 67, oltre la posizione corrente (53), occorre conoscere la direzione del movimento delle testine. Se lo spostamento è nella direzione del cilindro 0, l'unità a disco servirà prima la richiesta 37 e poi la 14; una volta giunto al cilindro 0, il braccio invertirà il movimento verso l'altro estremo del disco, servendo le richieste 65, 67, 98, 122, 124 e 183 (Figura 10.6). Se arriva una nuova richiesta riferita a uno dei cilindri posti davanti alla testina essa sarà quasi immediatamente soddisfatta; ma se la richiesta è riferita a uno dei cilindri appena sorpassati, essa dovrà attendere fino a che la testina non giunga alla fine del disco, inverta la direzione del moto, e torni indietro.

Assumendo una distribuzione uniforme delle richieste per i cilindri da visitare, si consideri la densità di richieste quando il braccio giunge a un estremo e inverte la direzione del moto: in quel momento, relativamente poche richieste sono riferite a ci-

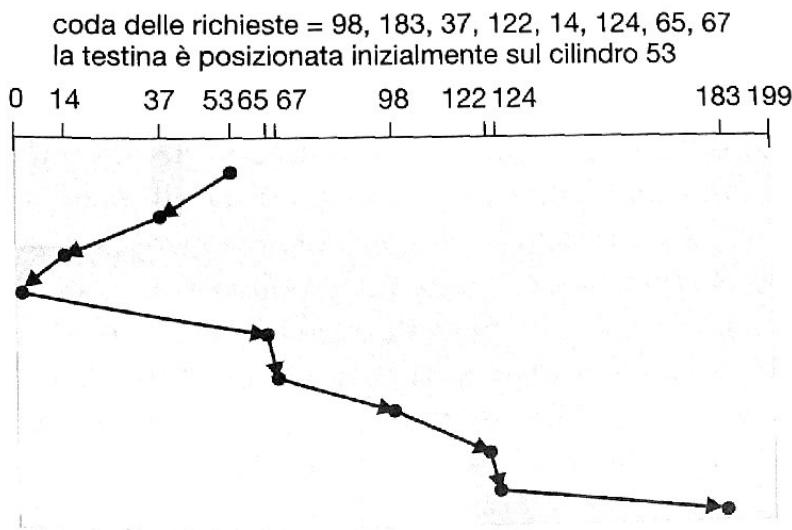


Figura 10.6 Scheduling SCAN.

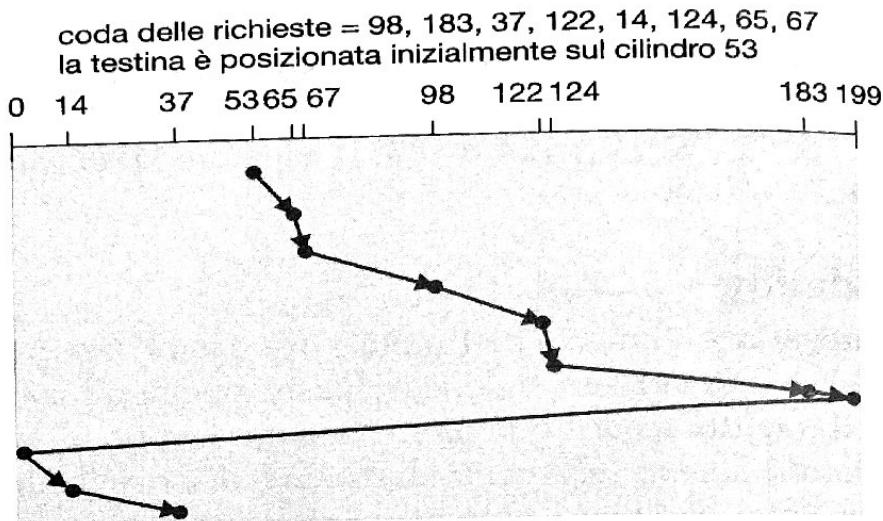


Figura 10.7 Scheduling C-SCAN.

lindri posti appena davanti alla testina, perché i cilindri in questione sono stati recentemente visitati. La massima densità di richieste si riferisce all’altro estremo del disco, e queste richieste sono anche quelle che hanno atteso più a lungo: sembra ragionevole spostarsi subito lì. Questa è l’idea dell’algoritmo seguente.

10.4.4 Scheduling – C-SCAN

L’algoritmo **SCAN circolare** (*circular SCAN*, **C-SCAN**) è una variante dello scheduling SCAN concepita per garantire un tempo d’attesa meno variabile. Anche l’algoritmo C-SCAN, come lo SCAN, sposta la testina da un estremo all’altro del disco, servendo le richieste lungo il percorso; tuttavia, quando la testina giunge all’altro estremo del disco, ritorna immediatamente all’inizio del disco stesso, senza servire richieste durante il viaggio di ritorno (Figura 10.7). L’algoritmo di scheduling C-SCAN, essenzialmente, tratta il disco come una lista circolare, cioè come se il primo e l’ultimo cilindro fossero adiacenti.

10.4.5 Scheduling LOOK

Secondo la descrizione appena data, sia l’algoritmo SCAN sia il C-SCAN spostano il braccio dell’unità attraverso tutta l’ampiezza del disco; in pratica, nessuno dei due algoritmi è implementato in questo modo: più comunemente, il braccio si sposta solo finché ci sono altre richieste da servire in quella direzione, dopo di che cambia immediatamente direzione, senza giungere all’estremo del disco. Queste versioni dello SCAN e del C-SCAN sono dette **LOOK** e **C-LOOK**, perché “guardano” (in inglese, *look*) se ci sono altre richieste da soddisfare lungo la direzione attuale prima di continuare a spostare la testina in quella direzione (Figura 10.8).

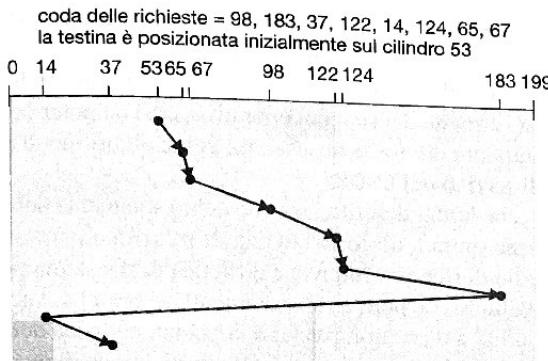


Figura 10.8 Scheduling C-LOOK.

10.4.6 Scelta di un algoritmo di scheduling

Dati tanti diversi algoritmi di scheduling, come scegliere il migliore? Un algoritmo molto comune e naturalmente attraente è l'SSTF poiché aumenta le prestazioni rispetto all'FCFS; lo SCAN e il C-SCAN danno migliori prestazioni in sistemi che sfruttano molto le unità a disco, perché conducono con minor probabilità a situazioni d'attesa infinita. Per una data lista di richieste si può definire un ordine ottimo di servizio, ma la computazione richiesta può non essere giustificata dal miglioramento in prestazioni rispetto agli algoritmi SSTF o SCAN.

Per qualunque algoritmo di scheduling, le prestazioni dipendono comunque in larga misura dal numero e dal tipo di richieste. Per esempio, si supponga che la coda sia costituita in genere di una sola richiesta inevasa: tutti gli algoritmi danno allora luogo allo stesso comportamento, perché hanno una sola scelta possibile di spostamento della testina. In questo caso, tutti gli algoritmi si comportano come l'FCFS.

Le richieste di I/O per l'unità a disco possono essere notevolmente influenzate dal metodo adottato per l'allocazione dei file. Un programma che legga un file allocato in modo contiguo genererà molte richieste raggruppate, con un conseguente limitato spostamento della testina. Un file con allocazione concatenata o indicizzata, d'altro canto, potrebbe includere blocchi sparsi per tutto il disco, e richiedere quindi un maggiore movimento della testina.

Anche la posizione delle directory e dei blocchi indice è importante: poiché ogni file deve essere aperto per essere usato, e visto che l'apertura di un file richiede una ricerca attraverso la struttura delle directory, vi saranno frequenti accessi alle directory. Si supponga che un elemento di directory risieda nel primo cilindro e che i dati del file relativo si trovino nell'ultimo cilindro; la testina dovrà allora percorrere l'intera ampiezza del disco nel caso di accesso al file in questione. Se l'elemento della directory fosse nel cilindro di mezzo, la testina dovrebbe spostarsi al più della metà dell'ampiezza. Anche l'uso della memoria centrale come cache delle directory e dei

blocchi indice può contribuire a ridurre i movimenti del braccio dell'unità a disco, in particolare quando si tratta di operazioni di lettura.

A causa di queste complicazioni, l'algoritmo di scheduling del disco dovrebbe costituire un modulo a sé stante del sistema operativo, così da poter essere sostituito da un altro algoritmo qualora ciò fosse necessario; come algoritmo di partenza è ragionevole la scelta dell'SSTF o del LOOK.

Gli algoritmi di scheduling descritti tengono conto solamente del tempo di ricerca, mentre nelle moderne unità a disco la latenza di rotazione può essere lunga quasi quanto il tempo medio di ricerca. Tuttavia è difficile per il sistema operativo adottare una strategia di scheduling che porti a miglioramenti dei tempi di latenza di rotazione, perché le moderne unità a disco non rivelano la posizione fisica dei blocchi logici. I produttori di unità a disco hanno collaborato alla limitazione di questo problema incorporando algoritmi di scheduling all'interno dei controllori hardware delle unità a disco: se il sistema operativo invia un gruppo di richieste al controllore, esso può organizzarle in una coda e poi applicare algoritmi di scheduling che riducano sia i tempi di ricerca sia la latenza di rotazione.

Se l'unico elemento di cui tener conto fossero le prestazioni dell'I/O, il sistema operativo scaricherebbe volentieri la responsabilità dello scheduling sull'hardware del disco. In pratica, però, il sistema operativo può dover considerare altri vincoli relativi all'ordine in cui si devono servire le richieste: per esempio, la richiesta di una pagina di memoria virtuale potrebbe avere maggiore priorità rispetto all'I/O delle applicazioni, e le scritture divengono più urgenti delle letture quando la cache sta per esaurire le pagine disponibili. Inoltre, può essere auspicabile mantenere l'ordine naturale delle richieste di scrittura al fine di rendere il file system robusto rispetto ai crash di sistema: si consideri che cosa accadrebbe se il sistema operativoassegnasse un blocco di disco a un file, e un'applicazione scrivesse dati in quel blocco prima che il sistema operativo abbia la possibilità di aggiornare i metadati del file-system sul disco. Per conciliare queste esigenze, il sistema operativo può scegliere di accollarsi la responsabilità dello scheduling del disco e, per alcuni tipi di I/O, fornire le richieste al controllore una alla volta.

SSD E SCHEDULING

Gli algoritmi di scheduling del disco discussi in questo paragrafo hanno come obiettivo principale la riduzione della quantità di movimento della testina del disco nei dischi magnetici. Gli SSD, che non contengono testine in movimento, usano solitamente una semplice politica FCFS. Ad esempio, lo scheduler Noop di Linux utilizza una politica FCFS modificata per fondere richieste adiacenti. Il comportamento osservato degli SSD indica che il tempo richiesto per le letture è uniforme, ma che, a causa delle proprietà della memoria flash, il tempo richiesto per le scritture non è uniforme. Alcuni scheduler per i dischi SSD sfruttano questa proprietà e fondono soltanto le richieste di scrittura adiacenti, mentre le richieste di lettura sono servite in ordine FCFS.

10.5 Gestione dell'unità a disco

Il sistema operativo è anche responsabile di molti altri aspetti della gestione delle unità a disco. In questo paragrafo si discutono l'inizializzazione del disco, l'avviamento del sistema da disco, e la gestione dei blocchi difettosi.

10.5.1 Formattazione del disco

Un disco magnetico nuovo è *tabula rasa*: è semplicemente un piatto ricoperto di materiale magnetico; prima che possa memorizzare dati, deve essere diviso in settori che possano essere letti o scritti dal controllore. Questo processo si chiama formattazione di basso livello, o **formattazione fisica**. La **formattazione di basso livello** riempie il disco con una speciale struttura dati per ogni settore, tipicamente consistente di un'intestazione, un'area per i dati (di solito di 512 byte), e una coda. L'intestazione e la coda contengono informazioni usate dal controllore del disco, per esempio il numero del settore e un **codice per la correzione degli errori** (*error-correcting code*, ECC). Quando il controllore scrive dati in un settore nel corso di un'ordinaria operazione di I/O, aggiorna il valore dell'ECC secondo il contenuto dell'area di dati del settore. Quando il controllore legge dati da quel settore, calcola anche l'ECC e lo confronta con il suo valore memorizzato: se risulta una differenza, l'area dei dati del settore non è integra, e il settore del disco potrebbe essere difettoso (si veda il Paragrafo 10.5.3). L'ECC è un codice per la *correzione* degli errori: se solo alcuni bit di dati sono stati alterati, esso contiene sufficienti informazioni affinché il controllore possa identificare i bit in questione e ricalcolare il loro corretto valore. In questo caso il controllore riporta un **errore recuperabile** o **soft error**. Il controllore esegue automaticamente l'elaborazione dell'ECC ogni volta che accede a un settore del disco.

La formattazione fisica dei dischi è eseguita nella maggior parte dei casi dal costruttore come parte del processo produttivo; ciò permette al costruttore di provare il disco, e di inizializzare la corrispondenza fra blocchi logici e settori correttamente funzionanti del disco. In molte unità a disco, quando si richiede al controllore di formattare fisicamente il disco, si può anche specificare il numero di byte delle aree di dati comprese fra l'intestazione e la coda di un settore. La scelta è di solito ristretta a poche opzioni, come 256, 512 o 1024 byte. La formattazione in settori più grandi implica la presenza di meno settori su ogni traccia, ma anche meno intestazioni e code, e quindi maggior spazio per i dati utente. Alcuni sistemi operativi gestiscono solo settori di 512 byte.

Prima di usare un disco come contenitore di file, il sistema operativo deve ancora registrare le proprie strutture dati nel disco, cosa che fa in due passi. Il primo consiste nel suddividere il disco in uno o più gruppi di cilindri, detti **partizioni**. Il sistema operativo può trattare ogni partizione come se fosse un'unità a disco a sé stante: per esempio, una partizione può contenere una copia del codice eseguibile del sistema operativo, mentre un'altra contiene i file degli utenti. Il passo successivo è la **formattazione logica**, cioè la creazione di un file system: il sistema operativo registra nel disco le strutture dati iniziali relative al file system. Le strutture dati in questione pos-

sono includere descrizioni dello spazio libero e dello spazio allocato e una directory iniziale vuota.

Per una maggior efficienza, la maggior parte dei file system accorpa i blocchi in gruppi, detti **cluster**. L'I/O del disco procede per blocchi, ma l'I/O del file system procede invece per cluster, di modo che l'I/O abbia caratteristiche di accesso più sequenziale che casuale.

Alcuni sistemi operativi danno l'opportunità a certi programmi speciali di impiegare una partizione del disco come un grande array sequenziale di blocchi logici, non contenente alcuna struttura dati relativa al file system. Questo array è detto a volte **disco di basso livello** (*raw disk*); l'I/O relativo si chiama **I/O di basso livello** (*raw I/O*). Alcuni sistemi per la gestione di basi di dati, per esempio, preferiscono questo tipo di I/O perché permette di controllare l'esatta posizione nel disco di ogni record. Il raw I/O bypassa tutti i servizi del file system: gestione delle *buffer cache*, locking dei file, prefetching, l'allocazione dello spazio, la gestione dei nomi dei file e le directory. È possibile rendere certe applicazioni più efficienti permettendogli di implementare servizi di memorizzazione specializzati che usino una partizione di basso livello, ma la maggior parte delle applicazioni funziona meglio quando usufruisce degli ordinari servizi del file system.

10.5.2 Blocco d'avviamento

Affinché un calcolatore possa entrare in funzione, per esempio quando viene acceso o riavviato, è necessario che esegua un programma iniziale; di solito, questo programma d'avviamento iniziale (*bootstrap*) è piuttosto semplice. Esso inizializza il sistema in tutti i suoi aspetti, dai registri della CPU ai controllori dei dispositivi e al contenuto della memoria centrale, quindi avvia il sistema operativo. Per far ciò, il programma d'avviamento trova il kernel del sistema operativo nei dischi, lo carica nella memoria, e salta a un indirizzo iniziale per avviare l'esecuzione del sistema operativo.

Per la maggior parte dei calcolatori, il programma d'avviamento è memorizzato in una **memoria a sola lettura** (*read-only memory*, ROM), il che è conveniente, perché la ROM non richiede inizializzazione, e ha un indirizzo iniziale fisso dal quale la CPU può cominciare l'esecuzione ogniqualvolta si accende o si riavvia la macchina. Inoltre, visto che la ROM è a sola lettura, non può essere contaminata da un virus informatico. Il problema, però, è che cambiare il programma d'avviamento richiede in questo caso la sostituzione dei circuiti integrati della ROM. A causa di questo inconveniente, molti sistemi memorizzano nella ROM un piccolo caricatore d'avviamento (*bootstrap loader*) il cui solo compito è quello di caricare da un disco il programma di bootstrap completo. Quest'ultimo si può facilmente modificare: se ne scrive semplicemente una nuova versione nel disco. Il programma d'avviamento completo è registrato nei "blocchi di avviamento" in una posizione fissa del disco; un disco contenente una tale partizione si chiama **disco d'avviamento** o **disco di sistema**.

Il codice contenuto nella ROM d'avviamento istruisce il controllore dell'unità a disco affinché trasferisca il contenuto dei blocchi d'avviamento nella memoria (si noti che a questo fine non si carica alcun driver di dispositivo), quindi comincia a ese-

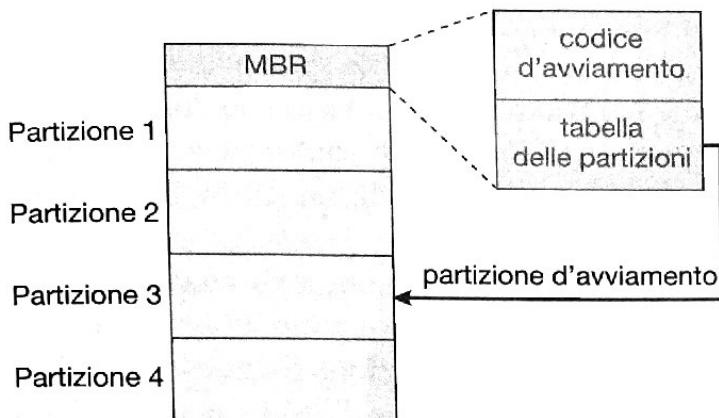


Figura 10.9 Avviamento dal disco di Windows.

guire il codice. Il programma d'avviamento completo è più complesso del suo caricatore, ed è capace di trasferire nella memoria l'intero sistema operativo inizialmente residente in un disco, in una locazione non fissata a priori, e di avviare il sistema operativo stesso. Anche così, il codice di bootstrap è abbastanza breve.

Consideriamo come esempio il processo d'avviamento in Windows. Osserviamo innanzitutto che Windows consente di suddividere il disco rigido in una o più partizioni; in una di esse, detta **partizione d'avviamento**, sono contenuti il sistema operativo e i driver dei dispositivi. Windows colloca il proprio codice d'avviamento nel primo settore del disco rigido, denominato **MBR** (*master boot record*). La procedura d'avviamento inizia con l'esecuzione del codice residente nella memoria ROM del sistema. Questo codice fa sì che il sistema legga il codice d'avviamento dall'MBR. Oltre al codice d'avviamento, l'MBR contiene una tabella che elenca le partizioni del disco rigido e un flag che indica da quale partizione si debba avviare il sistema, come è illustrato nella Figura 10.9. Dopo aver identificato la partizione d'avviamento, il sistema legge da tale partizione il primo settore (chiamato **settore d'avviamento**) e svolge le restanti procedure d'avviamento, tra cui il caricamento dei vari sottosistemi e dei servizi del sistema.

10.5.3 Blocchi difettosi

Le unità a disco sono strutturalmente portate ai malfunzionamenti perché sono costituite da parti mobili con basse tolleranze (si ricordi che una testina è sospesa appena sopra la superficie del disco). A volte si può verificare un guasto irreparabile, e l'unità a disco deve essere sostituita: il suo contenuto dovrà essere recuperato da una copia di backup e trasferito nella nuova unità a disco. Più di frequente, uno o più settori divengono malfunzionanti; in effetti, la maggior parte dei dischi messi in commercio contiene già **blocchi difettosi** (*bad blocks*). Essi sono trattati in diversi modi a seconda del controllore e del tipo di disco.

Nel caso di dischi semplici come quelli gestiti da un controllore IDE, i blocchi difettosi sono gestiti "manualmente". Una possibile strategia consiste nell'effettuare la scansione del disco durante la formattazione, per rilevare la presenza di blocchi di-

fettosi. Se si trova un blocco difettoso lo si marca come inutilizzabile al fine di segnalare alle procedure di allocazione del file system di non usarlo. Se qualche blocco diviene malfunzionante nel corso dell'ordinario uso del sistema, un programma speciale (per esempio il comando Linux `badblocks`) deve essere lanciato manualmente per individuare i blocchi difettosi e isolarli. Di solito, i dati residenti nei blocchi difettosi vanno perduti.

Unità a disco più complesse hanno strategie di recupero dei blocchi difettosi più raffinate. Il controllore mantiene una lista dei blocchi malfunzionanti dell'unità a disco che è inizializzata durante la formattazione fisica eseguita dal produttore, ed è aggiornata per tutto il periodo in cui l'unità a disco è operativa. La formattazione fisica mette anche da parte dei settori di riserva non visibili al sistema operativo: si può istruire il controllore affinché sostituiscia da un punto di vista logico un settore difettoso con uno dei settori di riserva inutilizzati. Questa strategia è nota come **accantonamento di settori** (*sector sparing* o *sector forwarding*).

Un tipico esempio di attuazione di questa strategia è il seguente:

- il sistema operativo tenta di leggere il blocco logico 87;
- il controllore calcola l'ECC e scopre che il settore è difettoso; quindi segnala questo malfunzionamento al sistema operativo;
- la volta successiva che il sistema viene riavviato, si esegue un comando speciale al fine di comunicare al controllore la necessità di sostituire il settore difettoso con uno di riserva;
- dopo di ciò, ogni volta che il sistema tenta di leggere il contenuto del blocco 87, il controllore traduce la richiesta nell'indirizzo del settore di rimpiazzo.

Si noti che un tale reindirizzamento da parte del controllore potrebbe inficiare ogni ottimizzazione fornita dall'algoritmo di scheduling del disco del sistema operativo. Per questa ragione la maggior parte dei dischi viene formattata in modo tale da mantenere alcuni settori di riserva in ogni cilindro, e anche un intero cilindro di riserva. Quando un blocco difettoso viene rimappato, il controllore usa settori di riserva presenti nello stesso cilindro ogniqualvolta è possibile.

Un'alternativa all'accantonamento dei settori è data da quei controllori capaci di sostituire i settori difettosi tramite la **traslazione dei settori** (*sector slipping*). Si supponga per esempio che il blocco logico 17 divenga malfunzionante, e che il primo settore di riserva disponibile sia quello successivo al settore 202. La traslazione dei settori sposta in avanti di una posizione tutti i settori dal 17 al 202: quindi, il settore 202 viene copiato sul settore di riserva, il settore 201 sul 202, il 200 sul 201, e così via, fino a che il settore 18 non viene copiato sul 19. Questa traslazione dei settori libera lo spazio del settore 18, e il settore 17 può essere mappato su quest'ultimo.

La sostituzione di un blocco difettoso non è in genere un processo totalmente automatico, perché i dati contenuti nel blocco in questione di solito vanno perduti. I soft error possono attivare un processo in cui viene effettuata una copia dei dati del blocco e il blocco viene messo in riserva o traslato. Un *errore non recuperabile* o **hard error**, tuttavia, causa una perdita di dati. Un file che usava quel blocco deve quindi essere

riparato (per esempio, ricopiandolo da un nastro contenente le copie di riserva), e ciò richiede un intervento manuale.

10.6 Gestione dell'area d'avvicendamento

L'avvicendamento (*swapping*) è stato introdotto, inizialmente, nel Paragrafo 8.2, dove abbiamo trattato lo spostamento di interi processi tra disco e memoria centrale. In quel contesto, l'avvicendamento interviene quando l'ammontare della memoria fisica si abbassa fino al punto di raggiungere la soglia critica e i processi vengono trasferiti dalla memoria all'area d'avvicendamento, per liberare memoria. Nella pratica, pochissimi sistemi operativi moderni realizzano l'avvicendamento nel modo descritto: essi, infatti, combinano l'avvicendamento con tecniche di memoria virtuale (Capitolo 9) ed effettuano lo swapping di pagine, e non necessariamente interi processi. Infatti alcuni sistemi considerano *avvicendamento* e *paginazione* termini intercambiabili, a riprova della moderna tendenza a convergere di questi due concetti.

La **gestione dell'area d'avvicendamento** è un altro compito di basso livello del sistema operativo. La memoria virtuale usa lo spazio dei dischi come estensione della memoria centrale: poiché l'accesso alle unità a disco è molto più lento dell'accesso alla memoria centrale, l'uso dell'area d'avvicendamento riduce notevolmente le prestazioni del sistema. L'obiettivo principale nella progettazione e realizzazione di un'area di avvicendamento è di fornire il migliore throughput per il sistema di memoria virtuale. In questo paragrafo sono trattati l'uso, la collocazione nei dischi e la gestione dell'area d'avvicendamento.

10.6.1 Uso dell'area d'avvicendamento

L'area d'avvicendamento (*area di swapping*) è usata in modi diversi da sistemi operativi diversi, in funzione degli algoritmi di gestione della memoria utilizzati. I sistemi che adottano l'avvicendamento dei processi nella memoria, per esempio, possono usare l'area d'avvicendamento per mantenere l'intera immagine del processo, inclusi i segmenti dei dati e del codice; i sistemi a paginazione, invece, possono semplicemente memorizzarvi pagine non contenute nella memoria centrale. Lo spazio richiesto dall'area d'avvicendamento per un sistema può quindi variare da pochi megabyte a alcuni gigabyte, a seconda della quantità di memoria fisica, della quantità di memoria virtuale che esso deve sostenere, e del modo in cui quest'ultima è usata.

Si noti che una stima per eccesso delle dimensioni dell'area d'avvicendamento è più prudente di una per difetto, perché un sistema che esaurisca l'area d'avvicendamento potrebbe essere costretto a terminare forzatamente i processi o ad arrestarsi completamente: una stima per eccesso spreca spazio dei dischi che si potrebbe usare per i file, ma non provoca altri danni. Alcuni sistemi consigliano la quantità da riservare per l'area d'avvicendamento. Solaris, per esempio, raccomanda di riservare a tal fine uno spazio uguale alla quantità di memoria virtuale che eccede la memoria fisica paginabile. Linux ha in passato suggerito di raddoppiare l'area d'avvicendamento ri-

spetto alla memoria fisica, ma oggi questa limitazione è scomparsa e la maggior parte dei sistemi Linux usa un'area d'avvicendamento notevolmente minore.

Alcuni sistemi operativi, fra i quali Linux, permettono l'uso di aree d'avvicendamento multiple, che includono sia file che partizioni dedicate. Queste aree d'avvicendamento sono poste di solito in unità a disco distinte per distribuire su più dispositivi il carico della paginazione e dell'avvicendamento dei processi gravante sul sistema per l'I/O.

10.6.2 Collocazione dell'area d'avvicendamento

Le possibili collocazioni per un'area d'avvicendamento sono due: all'interno del normale file system, o in una partizione del disco a sé stante. Se l'area d'avvicendamento è semplicemente un grande file all'interno del file system, si possono usare le ordinarie funzioni del file system per crearla, assegnargli un nome, e allocare spazio per essa. Questo approccio, sebbene sia semplice da realizzare, è inefficiente: l'attraversamento della struttura delle directory e delle strutture dati per l'allocazione dello spazio nei dischi richiede tempo, oltre che, almeno potenzialmente, operazioni di accesso aggiuntive ai dischi. La frammentazione esterna può aumentare molto i tempi di swapping, causando seek multipli durante la scrittura o la lettura dell'immagine di un processo. Le prestazioni si possono migliorare impiegando la memoria fisica come cache per le informazioni relative alla posizione dei blocchi, e anche usando strumenti speciali per l'allocazione in blocchi fisicamente contigui del file d'avvicendamento, ma il costo dovuto all'attraversamento delle strutture dati del file system permane.

In alternativa, l'area d'avvicendamento si può creare in un'apposita **partizione del disco non formattata** (*raw partition*): in essa non è presente alcuna struttura relativa al file system e alle directory, ma si usa uno speciale gestore dell'area d'avvicendamento per allocare e rimuovere i blocchi. Esso adotta algoritmi ottimizzati rispetto alla velocità di accesso, e non rispetto allo spazio impiegato su disco, dato che all'area d'avvicendamento (quando viene utilizzata) si accede molto più frequentemente che al file system. La frammentazione interna può aumentare, ma questo prezzo da pagare è ragionevole perché i dati nell'area d'avvicendamento hanno una vita media molto più breve dei file ordinari. Poiché l'area d'avvicendamento è reinizializzata all'avvio del sistema, la frammentazione ha vita breve. Il metodo della raw partition assegna una dimensione fissa all'area d'avvicendamento al momento della creazione delle partizioni del disco, e l'aumento delle dimensioni dell'area d'avvicendamento deve quindi passare attraverso il ripartizionamento del disco (che implica lo spostamento o l'eliminazione e la sostituzione con copie di riserva delle altre partizioni del disco), oppure attraverso la creazione di un'altra area d'avvicendamento in qualche altra unità a disco del sistema.

Alcuni sistemi operativi non adottano una strategia rigida e possono costruire aree d'avvicendamento sia su partizioni specifiche sia all'interno del file system: Linux ne è un esempio. I metodi di gestione e la loro implementazione sono diversi nei due casi, e la scelta fra le due soluzioni è lasciata all'amministratore del sistema: sul piatto

della bilancia pesano da un lato la facilità dell'allocazione e della gestione dell'area d'avvicendamento all'interno del file system, e dall'altro le migliori prestazioni ottenibili grazie all'uso di una partizione non formattata.

10.6.3 Gestione dell'area d'avvicendamento: un esempio

Si può comprendere come l'area d'avvicendamento venga utilizzata ripercorrendo l'evoluzione dello swapping e della paginazione nei vari sistemi UNIX. Il kernel tradizionale di UNIX implementava inizialmente una tecnica d'avvicendamento che spostava interi processi tra regioni contigue del disco e la memoria. Più tardi, con la disponibilità dell'hardware per la paginazione, UNIX progredì verso una combinazione d'avvicendamento e paginazione.

Per migliorare l'efficienza e sfruttare le innovazioni tecnologiche, in Solaris 1 (SunOS) i progettisti cambiarono le tecniche tradizionali di UNIX. Quando un processo va in esecuzione, le pagine contenenti il codice eseguibile sono prelevate dal file system, poste in memoria centrale e, qualora vengano selezionate per essere sovrascritte, eliminate. Rileggere una pagina dal file system è più efficiente che scriverla nell'area d'avvicendamento e rileggerla da lì: l'area d'avvicendamento è adoperata esclusivamente come area di stoccaggio per le pagine della **memoria anonima**, di cui fanno parte la memoria allocata per lo stack, quella per lo heap e i dati non inizializzati dei processi.

Ulteriori cambiamenti sono stati introdotti nelle versioni più recenti di Solaris. Il più importante prevede che l'area d'avvicendamento sia allocata solo quando una pagina è portata fuori dalla memoria fisica, anziché quando la pagina è creata per la prima volta in memoria virtuale. Questa regola produce un miglioramento delle prestazioni nei calcolatori moderni, i quali, rispetto ai sistemi più vecchi, possono contare su una maggiore quantità di memoria fisica e limitare il ricorso alla paginazione.

Così come per Solaris, l'area d'avvicendamento di Linux è utilizzata soltanto per la memoria anonima, ovvero per la memoria che non corrisponde ad alcun file. Linux permette di istituire una o più aree d'avvicendamento, sia in file di avvicendamento (*swap file*) del file system regolare, sia in una partizione dedicata. Un'area d'avvicendamento è formata da una serie di moduli di 4 KB detti **slot delle pagine**, la cui funzione è di conservare le pagine avvicendate. Ogni area d'avvicendamento ha una **mappa d'avvicendamento** (*swap map*) associata, ovvero un array di contatori interi, ciascuno dei quali corrisponde a uno slot nell'area d'avvicendamento. Se un contatore segna il valore 0, la pagina che gli corrisponde è disponibile. Valori superiori a 0 indicano che lo slot è occupato da una delle pagine avvicendate. Il valore del contatore indica il numero di collegamenti alla pagina; se esso è 3, per esempio, allora la pagina avvicendata è associata a tre processi differenti, eventualità possibile nel caso che la pagina rappresenti una regione di memoria condivisa da tre processi. Le strutture dati relative all'avvicendamento nei sistemi Linux sono rappresentate dalla Figura 10.10.

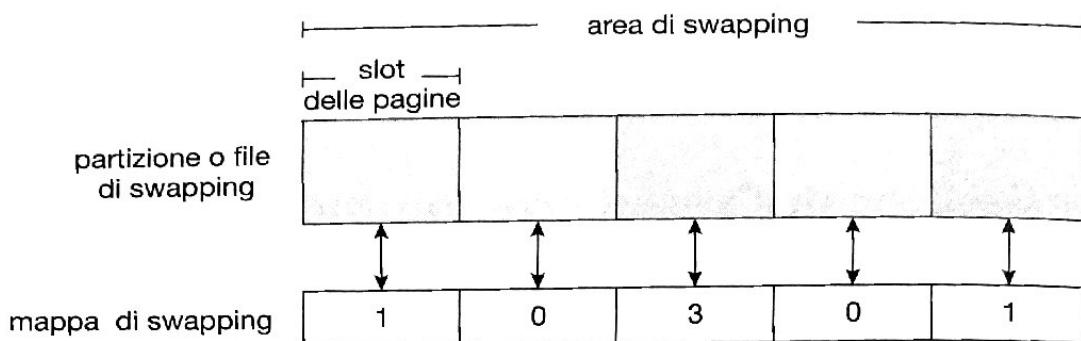


Figura 10.10 Strutture dati per lo swapping nei sistemi Linux.

10.7 Strutture RAID

L’evoluzione tecnologica ha reso le unità a disco progressivamente più piccole e meno costose, tanto che oggi è economicamente possibile equipaggiare un sistema elaborativo con molti dischi. La presenza di più dischi, qualora si possano usare in parallelo, rende possibile l’aumento della frequenza a cui i dati si possono leggere o scrivere. Inoltre, una configurazione di questo tipo permette di migliorare l’affidabilità della memoria secondaria, poiché diventa possibile memorizzare le informazioni in più dischi in modo ridondante. In questo caso, un guasto a uno dei dischi non comporta la perdita di dati. Ci sono varie tecniche per l’organizzazione dei dischi, note col nome comune di **batterie ridondanti di dischi** (*redundant array of independent disks*, RAID), che hanno lo scopo di affrontare i problemi di prestazioni e affidabilità.

Nel passato, strutture RAID composte da piccoli dischi economici erano viste come un’alternativa economicamente vantaggiosa rispetto a costosi dischi di grande capacità; oggi, le strutture RAID s’impiegano per la loro maggiore affidabilità e velocità di trasferimento dei dati, piuttosto che per ragioni economiche. Quindi, la *I* in RAID viene attualmente letta *independent* anziché *inexpensive* com’era originariamente.

STRUTTURA DEI DISPOSITIVI RAID

Le memorie RAID si prestano a essere strutturate con modalità diverse. Un sistema, per esempio, può collegare direttamente i dischi ai propri bus, nel qual caso la funzionalità RAID può essere realizzata dal sistema operativo o dai programmi di sistema. In alternativa, un controllore intelligente può gestire diversi dischi collegati alla macchina e implementare una struttura RAID per quei dischi a livello hardware. Infine, si può ricorrere a una **batteria RAID** (*RAID array*), un’unità a sé stante, dotata di un controllore, di una cache (nella maggioranza dei casi) e di dischi autonomi. L’array è collegato alla macchina attraverso uno o più controlleri (ad esempio FC). Questa diffusa organizzazione consente a programmi e sistemi operativi di per sé privi della funzionalità RAID di usufruirne comunque. Persino sistemi che, in realtà, implementano le funzionalità RAID a livello software adottano a volte la tecnica descritta, per via della sua semplicità e flessibilità.

10.7.1 Miglioramento dell'affidabilità tramite la ridondanza

Consideriamo in primo luogo l'affidabilità dei RAID. La possibilità che uno dei dischi in un insieme di n dischi si guasti è molto più alta della possibilità che uno specifico disco presenti un guasto. Si supponga che il **tempo medio di guasto** di un singolo disco sia 100.000 ore. In questo caso, il tempo medio di guasto per un qualsiasi disco in una batteria di 100 dischi sarebbe $100.000/100 = 1000$ ore, o 41,66 giorni: non molto tempo! Se si memorizzasse una sola copia dei dati, allora ogni guasto di un disco comporterebbe la perdita di una notevole quantità di dati; una frequenza di perdita di dati così alta sarebbe inaccettabile.

La soluzione al problema dell'affidabilità sta nell'introdurre una certa **ridondanza**, cioè nel memorizzare informazioni che non sono normalmente necessarie, ma che si possono usare nel caso di un guasto a un disco per ricostruire le informazioni perse.

Il metodo più semplice (ma anche il più costoso) di introduzione di ridondanza è quello della duplicazione di ogni disco. Questo metodo è detto **mirroring** (*copiatura speculare*): ogni disco logico consiste di due dischi fisici e ogni scrittura si effettua in entrambi i dischi. Si ottiene un disco duplicato (detto **mirrored volume**). Se uno dei dischi si guasta, i dati si possono leggere dall'altro. I dati si perdono solo se il secondo disco si guasta prima della sostituzione del disco già guasto.

Il tempo medio di guasto di un disco duplicato, dove per *guasto* s'intende ora la perdita di dati, dipende da due fattori: il tempo medio di guasto di un singolo disco e il **tempo medio di riparazione**, cioè il tempo richiesto (in media) per sostituire un disco guasto e ripristinarvi i dati. Supponendo che i possibili guasti dei due dischi siano **indipendenti**, vale a dire che il guasto di un disco non sia mai legato a quello dell'altro, se il tempo medio di guasto di un singolo disco è 100.000 ore e il tempo medio di riparazione è di 10 ore, allora il tempo medio di perdita di dati di un sistema con mirroring è $100.000^2/(2 * 10) = 500 * 10^6$ ore, che corrispondono a 57.000 anni!

Occorre però notare che non è possibile assumere l'ipotesi di indipendenza tra i guasti dei dischi. Improvvisi cali di tensione e disastri naturali, quali terremoti, incendi e alluvioni, danneggerebbero con tutta probabilità entrambi i dischi. Inoltre, difetti di fabbricazione in una partita di dischi possono causare guasti correlati. Con l'invecchiamento del disco, la probabilità di un guasto aumenta, accrescendo la probabilità che un secondo disco si guasti mentre il primo è in riparazione. Tuttavia, nonostante tutte queste considerazioni, i sistemi con mirroring offrono un'affidabilità assai più alta dei sistemi a disco singolo.

I casi di caduta di alimentazione elettrica costituiscono un problema particolarmente sentito, poiché avvengono con una frequenza molto più alta dei disastri naturali. Anche impiegando il mirroring, se si sta svolgendo un'operazione di scrittura nello stesso blocco in entrambi i dischi e si verifica una caduta di alimentazione prima che sia completata la scrittura dell'intero blocco, i due blocchi possono ritrovarsi in uno stato incoerente. Una soluzione prevede la scrittura di una delle due copie e solo successivamente la scrittura della seconda. Un'altra soluzione è aggiungere una memoria non volatile a stato solido (NVRAM, *non-volatile RAM*) alla batteria RAID, protetta dalla perdita di dati causata dalle cadute di alimentazione: se è dotata di forme

di correzione d'errore come ECC o mirroring, la scrittura dei dati nella cache può essere considerata completa anche in quei casi.

10.7.2 Miglioramento delle prestazioni tramite il parallelismo

Vediamo come l'accesso in parallelo a più dischi può migliorare le prestazioni. Con il mirroring, la frequenza con la quale si possono gestire le richieste di lettura raddoppia, poiché ciascuna richiesta si può inviare indifferentemente a uno dei due dischi (sempre che entrambi i dischi siano funzionanti, condizione che è quasi sempre soddisfatta). La capacità di trasferimento di ciascuna lettura è la stessa di quella di un sistema a singolo disco, ma il numero di letture per unità di tempo raddoppia.

Attraverso l'uso di più dischi è possibile anche (o alternativamente) migliorare la capacità di trasferimento distribuendo i dati in sezioni su più dischi. La forma più semplice di questa distribuzione (*data striping*), consiste nel distribuire i bit di ciascun byte su più dischi; in questo caso si parla di **sezionamento** o **striping a livello dei bit**. Per esempio, se il sistema impiega un array di otto dischi, si scriverà il bit i di ciascun byte nel disco i . L'array di otto dischi si può trattare come un unico disco avente settori che hanno una dimensione otto volte superiore a quella normale e, soprattutto, che hanno una capacità di trasferimento otto volte superiore. In un'organizzazione di questo tipo, ogni disco è coinvolto in ogni accesso (lettura o scrittura che sia), così che il numero di accessi che si possono gestire nell'unità di tempo è circa lo stesso di quello per un sistema a disco singolo, ma ogni accesso permette di leggere una quantità di dati pari a otto volte quella che si può leggere con un singolo disco.

Lo striping a livello dei bit si può generalizzare a un numero di dischi multiplo di 8 o che divide 8. Per esempio, se un sistema adopera un array di quattro dischi, i bit i e $i + 4$ di ciascun byte si memorizzano nel disco i . Inoltre, lo striping non si deve realizzare necessariamente a livello dei bit di un byte: nello **striping a livello di blocco**, per esempio, i blocchi di un file si distribuiscono su più dischi; con n dischi, il blocco i di un file si memorizza nel disco $(i \bmod n) + 1$. Sono possibili anche altri livelli di striping, come quelli basati sui byte di un settore o sui settori di un blocco, ma lo striping a livello di blocco è il più comune.

Riassumendo, gli obiettivi principali del parallelismo mediante striping in un sistema di dischi sono due:

1. l'aumento, tramite il bilanciamento del carico, del throughput per accessi multipli a piccole porzioni di dati (cioè accessi a pagine);
2. la riduzione del tempo di risposta relativo ad accessi a grandi quantità di dati.

10.7.3 Livelli RAID

La tecnica di mirroring offre un'alta affidabilità ma è costosa; la tecnica di striping (sezionamento) offre un'alta capacità di trasferimento dei dati, ma non migliora l'affidabilità. Sono stati proposti numerosi schemi per fornire ridondanza a basso costo usando l'idea dello striping combinata con i bit di parità (che vedremo a breve). Que-

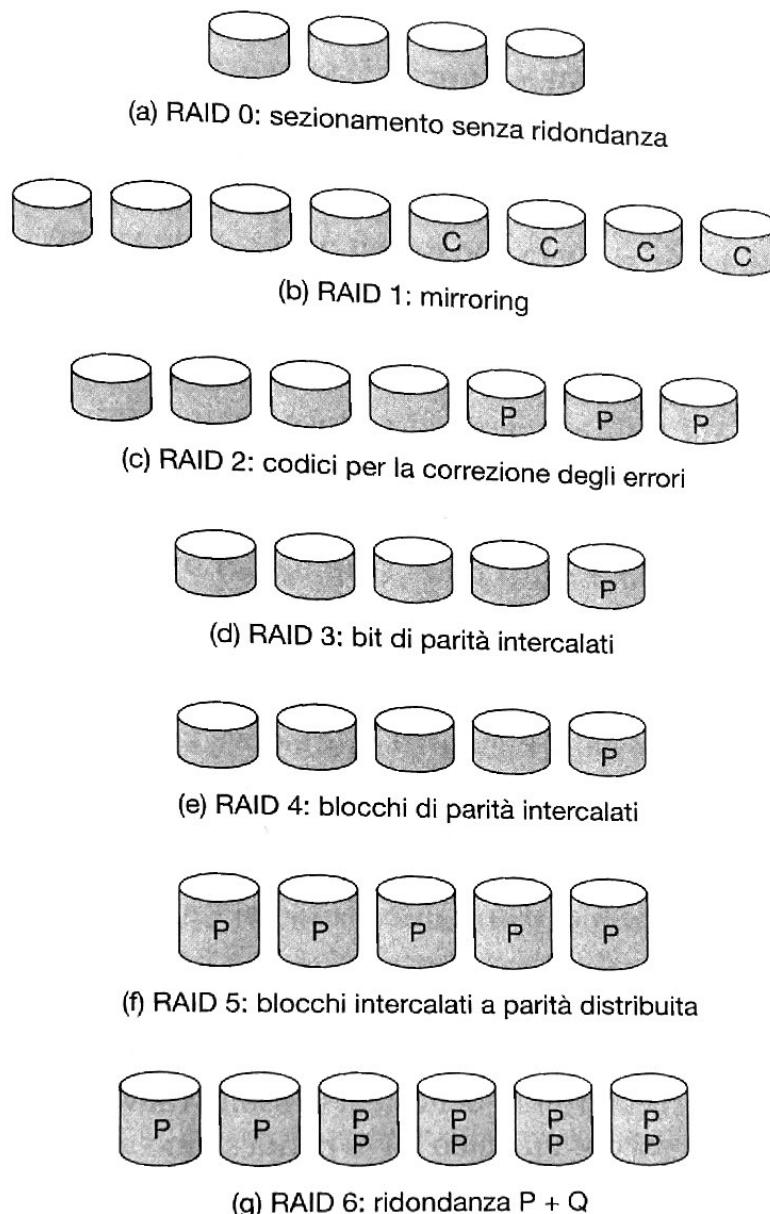


Figura 10.11 Livelli RAID.

sti schemi realizzano diversi compromessi tra costi e prestazioni e sono stati classificati in livelli chiamati **livelli RAID**, che la Figura 10.11 mostra graficamente (nella figura, la lettera *P* indica i bit di correzione degli errori, la lettera *C* indica una seconda copia dei dati). In tutti i casi riportati nella figura, sono presenti quattro dischi di dati, mentre i dischi supplementari s'impiegano per memorizzare le informazioni ridondanti per il ripristino dai guasti.

- **RAID di livello 0.** Il livello 0 si riferisce ad array di dischi con striping a livello di blocchi, ma senza ridondanza (come il mirroring o i bit di parità), come illustrato nella Figura 10.11(a).
- **RAID di livello 1.** Il livello 1 si riferisce alla tecnica di mirroring. La Figura 10.11(b) mostra un'organizzazione basata sul mirroring.

- **RAID di livello 2.** Il livello 2 è anche noto come **organizzazione con codici per la correzione degli errori di memoria**. Da molto tempo i sistemi di memorizzazione impiegano tecniche di riconoscimento degli errori basate su bit di parità. In un sistema di questo tipo, ogni byte di memoria può avere associato un bit di parità che indica se i bit con valore 1 nel byte sono in numero pari (parità = 0) oppure dispari (parità = 1). Se si altera uno dei bit nel byte (un valore 1 diventa 0 o viceversa), la parità del byte cambia e quindi non concorda più con la parità memorizzata. Analogamente, se si altera il bit di parità, esso non concorda più con la parità calcolata. In questo modo s'identificano tutti gli errori di un singolo bit nel sistema di memoria. Gli schemi di correzione degli errori memorizzano più bit supplementari e possono ricostruire i dati nel caso di un singolo bit danneggiato.

La stessa idea alla base degli ECC (*error-correcting codes*) si può usare immediatamente negli array di dischi eseguendo lo striping dei byte presenti nei dischi. Per esempio, il primo bit di ogni byte si potrebbe memorizzare nel disco 1, il secondo bit nel disco 2, e così via fino alla memorizzazione dell'ottavo bit nel disco 8 e alla memorizzazione dei bit di correzione degli errori in ulteriori dischi. Questo schema è rappresentato graficamente nella Figura 10.11(c), dove i dischi etichettati con la lettera *P* contengono i bit di correzione. Se uno dei dischi si guasta, i bit rimanenti del byte e i bit di correzione a esso associati si possono leggere dagli altri dischi e usare per ricostruire i dati danneggiati. Si noti che il RAID di livello 2 richiede tre soli dischi di overhead per quattro dischi di dati, a differenza del RAID di livello 1, che richiede quattro dischi in più.

- **RAID di livello 3.** Con il livello 3, o **organizzazione con bit di parità intercalati**, si migliora l'organizzazione del livello 2 considerando che, a differenza dei sistemi di memoria centrale, i controllori dei dischi possono rilevare se un settore è stato letto correttamente, così che un unico bit di parità si può usare sia per individuare gli errori sia per correggerli. L'idea è la seguente: se uno dei settori è danneggiato, si conosce esattamente di quale settore si tratta e, per ogni bit nel settore, è possibile determinare se debba avere valore 1 o 0 calcolando la parità dei bit corrispondenti dai settori negli altri dischi. Se la parità dei rimanenti bit è uguale a quella memorizzata, il bit mancante è 0, altrimenti è 1. Il RAID di livello 3 è altrettanto valido del livello 2 ma meno costoso in quanto richiede un solo disco supplementare, quindi il RAID di livello 2 non è utilizzato nella pratica. La Figura 10.11(d) mostra il livello 3.

Il livello 3 presenta due vantaggi rispetto al livello 1. Il primo vantaggio è che si usa un solo disco per la parità dei dati memorizzati in diversi dischi di dati, anziché un disco di mirroring per ciascun disco di dati come nel livello 1. Il secondo vantaggio è che, essendo le letture e le scritture dei byte distribuite su più dischi con uno striping a n vie, la velocità di trasferimento di un singolo blocco è pari a n volte quella dei RAID di livello 1. D'altro canto, però, il livello 3 permette meno operazioni di I/O al secondo, perché ogni disco è coinvolto da tutte le richieste.

Un altro problema di prestazioni riguardante il RAID di livello 3 (come per tutti i livelli RAID basati sui bit di parità) è il tempo richiesto dal calcolo e dalla scrittura

della parità. Questo tempo aggiuntivo determina operazioni di scrittura significativamente più lente rispetto ad array RAID senza parità. Per limitare questo calo di prestazioni, molti array RAID dispongono di un controllore capace di gestire il calcolo della parità. Questo sposta il carico dovuto al calcolo della parità dalla CPU all'array di dischi. L'array ha anche una cache NVRAM per memorizzare i blocchi mentre viene calcolata la parità e per memorizzare transitoriamente le scritture dal controllore ai dischi. Questa combinazione può rendere la tecnica RAID con parità altrettanto veloce di quella senza parità; infatti, un array RAID con cache e bit di parità può avere prestazioni migliori di un'organizzazione RAID senza cache e senza parità.

- **RAID di livello 4.** Nel livello 4, o **organizzazione con blocchi di parità intercalati**, s'impiega lo striping a livello di blocchi, come nel RAID di livello 0 e inoltre si tiene un blocco di parità in un disco separato per i blocchi corrispondenti presenti negli altri n dischi. Tale schema è illustrato nella Figura 10.11(e). Se uno dei dischi si guasta, il blocco di parità si può usare insieme ai blocchi corrispondenti degli altri dischi per ripristinare i blocchi nel disco guasto.

La lettura di un blocco richiede l'accesso a un solo disco, permettendo la gestione di altre richieste da parte di altri dischi. Quindi, la capacità di trasferimento dei dati per ciascun accesso è minore, ma gli accessi in lettura possono procedere in modo parallelo ottenendo una velocità complessiva di I/O più alta. La capacità di trasferimento per la lettura di molti dati è alta, poiché si possono leggere in modo parallelo tutti i dischi e anche le operazioni di scrittura di grandi quantità di dati presentano un'alta capacità di trasferimento, poiché i dati e i bit di parità si possono scrivere in parallelo.

Scritture indipendenti di modesta entità non si possono eseguire in parallelo. La scrittura da parte del sistema operativo di una quantità di dati inferiore a un blocco richiede la lettura del blocco, la sua modifica, e la scrittura del blocco modificato; anche il blocco di parità deve essere aggiornato. Si parla a questo proposito del **ciclo lettura-modifica-scrittura**. Una singola richiesta di scrittura comporta pertanto quattro accessi al disco, due in lettura e due in scrittura.

Nel Capitolo 12 sarà presentato WAFL: esso adotta RAID di livello 4, che permette l'aggiunta di dischi al sistema senza soluzione di continuità. Inizializzando i nuovi dischi a zero, la parità non cambia, e l'array RAID è ancora nello stato corretto.

- **RAID di livello 5.** Il livello 5, o **organizzazione con blocchi intercalati a parità distribuita**, differisce dal livello 4 per il fatto che, invece di memorizzare i dati in n dischi e la parità in un disco separato, i dati e le informazioni di parità sono distribuite tra gli $n + 1$ dischi. Per ogni blocco, uno dei dischi memorizza la parità e gli altri i dati. Per esempio, considerando una batteria di cinque dischi, la parità per il blocco m -esimo si memorizza nel disco $(m \bmod 5) + 1$, mentre i blocchi m -esimi degli altri quattro dischi contengono i dati effettivi per quel blocco. Questo schema è illustrato nella Figura 10.11(f), dove i simboli P sono distribuiti su tutti i dischi. Un blocco di parità non può contenere informazioni di parità per blocchi

che risiedono nello stesso disco, poiché un guasto al disco provocherebbe sia la perdita di dati sia la perdita dell'informazione di parità e quindi i dati non sarebbero ripristinabili. Con la distribuzione della parità sui diversi dischi, il RAID di livello 5 evita un uso intensivo del disco dove risiede la parità, che invece si ha con il RAID di livello 4. RAID 5 è il più comune sistema di parità RAID.

- **RAID di livello 6.** Il livello 6, detto anche **schemma di ridondanza P + Q**, è molto simile al RAID di livello 5, ma memorizza ulteriori informazioni ridondanti per poter gestire guasti contemporanei di più dischi. Invece di usare la parità, s'impiegano codici per la correzione degli errori come i **codici di Reed-Solomon**. Nello schema mostrato nella Figura 10.11(g), sono memorizzati 2 bit di dati ridondanti ogni 4 bit di dati effettivi (a differenza di 1 bit di parità usato nel livello 5) e il sistema risultante può tollerare due guasti dei dischi.
- **RAID di livello 0 + 1 e 1 + 0.** Il livello 0 + 1 consiste in una combinazione dei livelli RAID 0 e 1. Il livello 0 fornisce le prestazioni, mentre il livello 1 l'affidabilità. Di solito, questo schema porta a prestazioni migliori rispetto a livello 5 e si usa prevalentemente negli ambienti in cui sono importanti sia le prestazioni sia l'affidabilità. Sfortunatamente, questo schema richiede, come il RAID di livello 1, un raddoppio del numero di dischi necessario per memorizzare i dati, quindi è anche relativamente costoso. Nel RAID di livello 0 + 1, si effettua lo striping su un insieme di dischi e si duplica ogni sezione con la tecnica del mirroring.

Un altro metodo che sta diventando disponibile commercialmente è il RAID di livello 1 + 0, in cui si fa prima il mirroring dei dischi a coppie, e poi lo striping di queste coppie. Questo schema RAID ha alcuni vantaggi teorici rispetto al RAID 0 + 1. Per esempio, se si guasta un singolo disco nel RAID 0 + 1, l'intera sezione di dati diventa inaccessibile, lasciando disponibile solo l'altra sezione. Con un guasto nel RAID 1 + 0, il singolo disco diventa inaccessibile, ma il suo duplicato è ancora disponibile, come tutti gli altri dischi (Figura 10.12).

Sono state proposte numerose altre varianti agli schemi RAID di base illustrati sopra e questo ha portato anche una certa confusione nelle precise definizioni dei diversi livelli RAID.

Un altro aspetto soggetto a molte varianti è l'implementazione del RAID. Esamiammo i diversi strati architetturali a cui è possibile implementare un sistema RAID.

- Il software per la gestione dei volumi può implementare un sistema RAID all'interno del kernel o a livello dei programmi di sistema. In questo caso, nonostante i dispositivi per la memorizzazione possano fornire funzionalità minime, è possibile ottenere un sistema RAID completo. Il metodo RAID con parità è alquanto lento se realizzato tramite software, quindi gli si preferisce solitamente RAID 0,1 oppure 0 + 1.
- Il RAID può essere implementato a livello hardware dall'adattatore del bus della macchina (*host bus adapter*, HBA). Solo i dischi connessi direttamente all'HBA possono costituire parte integrante di una dato array RAID. Questa soluzione è a basso costo, ma non è molto flessibile.

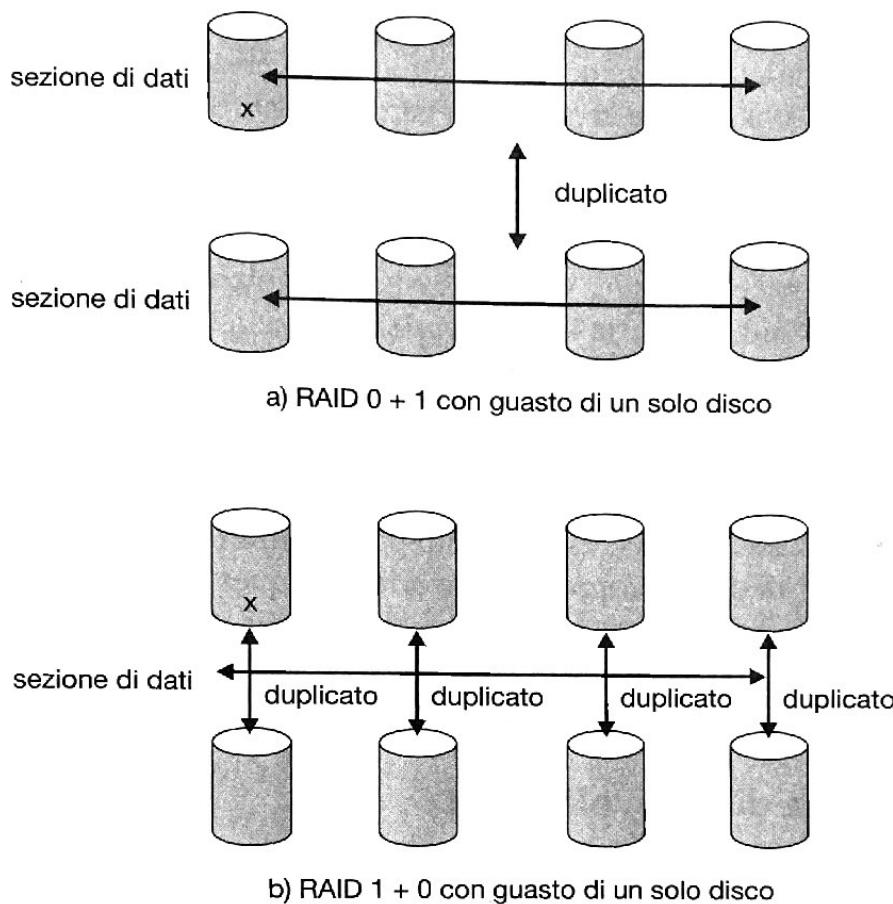


Figura 10.12 RAID 0 + 1 e 1 + 0.

- Il RAID può essere implementato a livello hardware dall'array di dischi. È così possibile creare sistemi RAID a vari livelli, e persino ricavare da essi volumi più piccoli, che sono quindi presentati al sistema operativo, che avrà solo da realizzare il file system su ciascuno dei volumi. Gli array possono disporre di connessioni multiple o far parte di una rete di memorizzazione secondaria (SAN), consentendo a varie macchine di sfruttare le funzionalità dell'array.
- Il RAID può essere implementato da dispositivi di virtualizzazione del disco a livello di interconnessione SAN. In questo caso, un dispositivo funge da intermediario tra le macchine e l'area di memorizzazione, accettando istruzioni dai server e gestendo l'accesso alla memoria secondaria. Esso potrebbe, per esempio, attuare il mirroring, trascrivendo ciascun blocco su due distinti dispositivi di memorizzazione.

Ulteriori funzionalità, come quella di istantanea e di replica, possono essere implementate a ognuno di questi livelli. Un'**istantanea** (*snapshot*) è un'immagine del file system così com'era prima dell'ultimo aggiornamento (le istantanee saranno trattate più in dettaglio nel Capitolo 12). La **replica** prevede la duplicazione automatica di scritture su siti diversi, per finalità di ridondanza, o di ripristino in caso di eventi disastrosi (*disaster recovery*). La replica può essere sincrona o asincrona. Se è sincrona,

ciascun blocco deve essere scritto sia localmente, sia in remoto, prima che la scrittura sia considerata completa; se è asincrona, si effettuano periodicamente scritture a gruppi. La replica asincrona espone al rischio di perdere i dati, se il sito principale fallisce, ma è più veloce e non ha limiti di distanza.

L'implementazione di queste funzionalità varia a seconda dello strato scelto per realizzare il sistema RAID. Qualora RAID, per esempio, sia implementato a livello software, ciascuna macchina può aver necessità di implementare e gestire la replica per proprio conto. Tuttavia, se la replica avviene a livello dell'array di dischi o dell'interconnessione SAN, si possono replicare i dati della macchina a prescindere dal suo sistema operativo e dalle relative funzionalità.

Un'altra caratteristica spesso presente nei sistemi RAID è la previsione di **dischi di scorta** (*hot spare*), che possono sostituire quelli normali in caso di guasti. Per esempio, un disco di scorta può essere usato per sostituire un disco danneggiato, ricostruendo l'integrità di una coppia in mirroring. In questo modo, si può ristabilire automaticamente lo stato corretto del livello RAID, senza attendere che il disco difettoso sia sostituito. È possibile riparare più di un guasto, senza l'intervento di un operatore, con l'allocazione di più dischi di scorta.

10.7.4 Scelta di un livello RAID

Viste le svariate possibilità esistenti, ci si potrebbe chiedere quali siano i criteri di scelta dei progettisti nei confronti del livello RAID. Una considerazione è il tempo di ricostruzione. Se un disco si guasta, il tempo necessario a ricostruire i dati che contiene può essere rilevante. Questo fattore può essere importante nel caso in cui venga richiesto un flusso continuo di dati, come nei database ad alte prestazioni o interattivi. Inoltre il tempo di ricostruzione influenza il tempo medio di guasto.

Il tempo di ricostruzione varia a seconda del livello RAID utilizzato. La ricostruzione più semplice si ha per RAID di livello 1, poiché i dati possono essere copiati da un altro disco; per gli altri livelli, per ricostruire i dati in un disco guasto è necessario accedere a tutti gli altri dischi dell'array. Il tempo necessario per la ricostruzione dei dati può essere di ore nel caso di sistemi RAID di livello 5 con molti dischi.

RAID di livello 0 si usa nelle applicazioni ad alte prestazioni in cui le perdite di dati non sono critiche. Il RAID di livello 1 si usa comunemente nelle applicazioni che richiedono un'alta affidabilità e un rapido ripristino. I livelli RAID 0 + 1 e 1 + 0 si usano dove sia le prestazioni sia l'affidabilità sono importanti, per esempio per piccole basi di dati. A causa dell'elevata richiesta di spazio del RAID di livello 1, per la memorizzazione di grandi quantità di dati, spesso si preferisce impiegare il RAID di livello 5. Il livello 6, attualmente non disponibile in molte implementazioni RAID, dovrebbe offrire una migliore affidabilità rispetto a livello 5.

Progettisti e amministratori di sistemi RAID devono prendere anche altre decisioni importanti, per esempio riguardo al numero ottimale di dischi in un array e al numero di bit che ciascun bit di parità deve proteggere. Maggiore è il numero di dischi in un array, maggiore sarà la capacità di trasferimento dei dati, ma il sistema sarà anche più costoso. Maggiore è il numero di bit protetti da un singolo bit di parità, minore

L'ARRAY InServ

L'innovazione, grazie al quale sono di continuo introdotte soluzioni migliori, più veloci e meno costose, ridefinisce spesso i confini che separano le tecnologie già esistenti. Si consideri, per esempio, l'array InServ di 3Par. A differenza di molti altri, esso non richiede la configurazione di un insieme di dischi a un livello RAID specifico, ma scomponete invece ogni disco in porzioni da 256 MB. Il metodo RAID è pertanto applicato a livello di queste porzioni. Di conseguenza, vari livelli RAID possono interessare lo stesso disco, e le sue porzioni vengono utilizzate per formare diversi volumi.

InServ mette inoltre a disposizione le istantanee, una funzionalità simile a quella del file system WAFL. Le istantanee di InServ prevedono sia il formato lettura-scrittura sia il formato a sola lettura, consentendo a utenti multipli di montare copie di un dato file system senza doverne possedere una copia completa. Le modifiche eventualmente apportate dagli utenti alla propria copia sono di copiatura su scrittura, ragion per cui non hanno effetto sulle altre copie.

Un'altra innovazione è il cosiddetto **utility storage**. Alcuni file system non possono essere espansi, né compressi. I file system di questo genere mantengono costantemente le dimensioni originali: per qualsiasi modifica è necessaria la copiatura di dati. Un amministratore può configurare InServ per fornire a una macchina cospicue quantità di memoria logica, che all'inizio occupano solamente un piccolo spazio di memoria fisica. Mentre la macchina comincia a usare lo spazio di memorizzazione, dischi non ancora utilizzati sono assegnati alla macchina, fino a raggiungere il livello logico originale. In tal modo, la macchina è indotta a credere di possedere un vasto spazio di memorizzazione permanente, dove creare i propri file system, e così via. InServ può aggiungere o rimuovere dischi dal file system senza che il file system se ne accorga. Questa caratteristica può ridurre il numero complessivo di unità a disco necessarie alle macchine, o perlomeno ritardare l'acquisizione di nuovi dischi finché non divengano realmente necessari.

Utility storage: una macchina può avere disponibili spazi di memoria logici molto superiori a quelli fisici.

sarà lo spazio richiesto dai bit di parità; sarà però maggiore anche la probabilità che un secondo disco si guasti prima che un disco guasto sia riparato e questo porterebbe alla perdita di dati.

10.7.5 Estensioni

I concetti relativi ai sistemi RAID sono stati generalizzati ad altri dispositivi di memorizzazione, comprese le unità a nastri e anche alla diffusione dei dati tramite sistemi senza fili (*wireless*). Con strutture RAID applicate alle unità a nastri si possono ripristinare i dati anche se uno dei nastri della batteria è danneggiato. Se applicate alla trasmissione dei dati, si divide ogni blocco di dati in unità più piccole che si trasmettono insieme a un'unità di parità; se per qualsiasi ragione una delle unità non viene ricevuta, può essere ricostruita dalle altre. Di solito, con l'uso di unità automatiche dotate di molte unità a nastro si esegue lo striping dei dati su tutte le unità per aumentare il throughput e diminuire il tempo di backup.

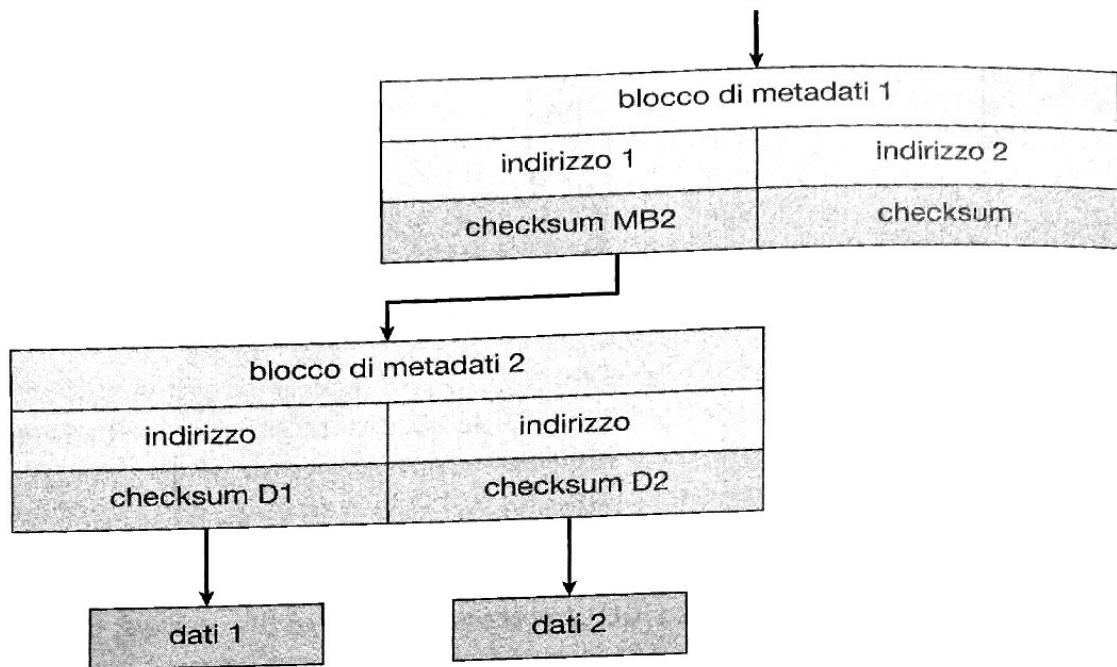


Figura 10.13 ZFS applica una checksum a tutti i metadati e i dati.

10.7.6 Problemi connessi a RAID

I sistemi RAID, purtroppo, non assicurano sempre la disponibilità dei dati al sistema operativo e agli utenti. Un puntatore a un file potrebbe essere errato, per esempio, e lo stesso potrebbe accadere ai puntatori nella struttura interna dei file. Le operazioni incomplete di scrittura, se non ripristinate in maniera adeguata, possono alterare i dati. Altri processi, inoltre, potrebbero scrivere accidentalmente sulle strutture del file system. RAID protegge dagli errori derivanti dai supporti fisici per la memorizzazione, ma non da altri tipi di errori dovuti all'hardware e ai programmi. I pericoli potenziali, per i dati di un sistema, si estendono alla totalità degli errori derivanti dal software e dall'hardware.

Per risolvere tali problemi, il file system **Solaris ZFS** ricorre a una strategia innovativa per verificare l'integrità dei dati. Esso applica una **checksum** (*somma di controllo*) interna a ogni blocco, dati e metadati inclusi. Le checksum non risiedono nel blocco sottoposto a controllo: ciascuna di esse, invece, è memorizzata insieme al puntatore a quel blocco (Figura 10.13). Si consideri un **inode** – una struttura dati per memorizzare i metadati del file system - con puntatori ai propri dati. All'interno dell'inode si trova la checksum per ciascun blocco di dati. Se si verifica un problema con i dati, la checksum darà un valore errato e il file system verrà a conoscenza del problema. Qualora sia attivo il mirroring, il sistema ZFS, in presenza di un blocco con una checksum corretta e di uno con una checksum errata, sostituirà automaticamente il blocco errato con quello valido. In maniera simile, l'elemento della directory che punta all'inode possiede una checksum relativa all'inode. Qualunque problema ri-

guardi l'inode, quindi, è rilevato al momento dell'accesso alla directory. Queste checksum, che sono applicate a tutte le strutture di ZFS, producono risultati molto più efficaci degli ambienti RAID o dei file system tradizionali, per livello di coerenza, rilevazione degli errori e capacità di correggerli. Il sovraccarico di gestione determinato dal calcolo delle checksum e dai cicli supplementari di lettura-modifica-scrittura dei blocchi non condizionano il funzionamento complessivo di ZFS, che mantiene un'alta velocità nelle prestazioni.

Un altro problema della maggior parte delle implementazioni RAID è la mancanza di flessibilità. Considerate un array di memorizzazione dotato di venti dischi divisi in quattro insiemi da cinque dischi. Ogni gruppo di cinque dischi è un insieme RAID di livello 5. Ne risultano quattro volumi separati, ciascuno contenente un proprio file system. Ma che cosa succede se un file system ha una dimensione troppo grande per un insieme RAID di livello 5 a cinque dischi? E se un altro file system necessita di un'area molto ridotta? Se tali fattori sono noti in anticipo, allora i dischi e i volumi possono essere allocati adeguatamente. Frequentemente, però, l'utilizzo del disco e le richieste variano nel tempo.

Anche se l'array di memorizzazione ha permesso all'intero insieme di venti dischi di essere creato come un grande insieme RAID, potrebbero insorgere altre problematiche. Nell'insieme potrebbero essere costruiti diversi volumi di dimensioni differenti. Alcuni gestori del volume non ci permettono però di cambiare la dimensione di un volume. In quel caso si ripresenterebbe la stessa situazione descritta in precedenza, ovvero dimensioni non adatte al file system. Alcuni gestori di volume permettono cambiamenti di dimensione, ma alcuni file system non permettono l'aumento o la diminuzione della dimensione del file system stesso. I volumi potrebbero cambiare dimensione, ma i file system dovrebbero essere ricreati per poter usufruire di quei cambiamenti.

ZFS combina la gestione dei file system e quella dei volumi in una unità in grado di offrire una maggiore funzionalità rispetto a quella permessa dalla tradizionale separazione di tali funzioni. I dischi, o le partizioni di dischi, sono riuniti in **gruppi di memorizzazione** (*pools of storage*) attraverso insiemi RAID. Un gruppo o **pool** può contenere uno o più file system ZFS. Tutta l'area libera di un pool è a disposizione di tutti i file system contenuti all'interno di quel pool. ZFS utilizza il modello di gestione della memoria basato su `malloc()` e `free()` per allocare e rilasciare memoria nei file system quando i blocchi vengono utilizzati e liberati all'interno del file system. Di conseguenza non ci sono limiti artificiali all'utilizzo della memoria e non sussiste la necessità di ridistribuire i file system tra i volumi né di ridimensionare i volumi. ZFS stabilisce delle quote per limitare la dimensione di un file system e definisce dei meccanismi di prenotazione per assicurarsi che il file system possa aumentare all'interno di una dimensione specificata, ma queste variabili possono essere sempre cambiate dal proprietario del file system. La Figura 10.14(a) illustra i volumi e i file system tradizionali, mentre la Figura 10.14(b) rappresenta il modello ZFS.

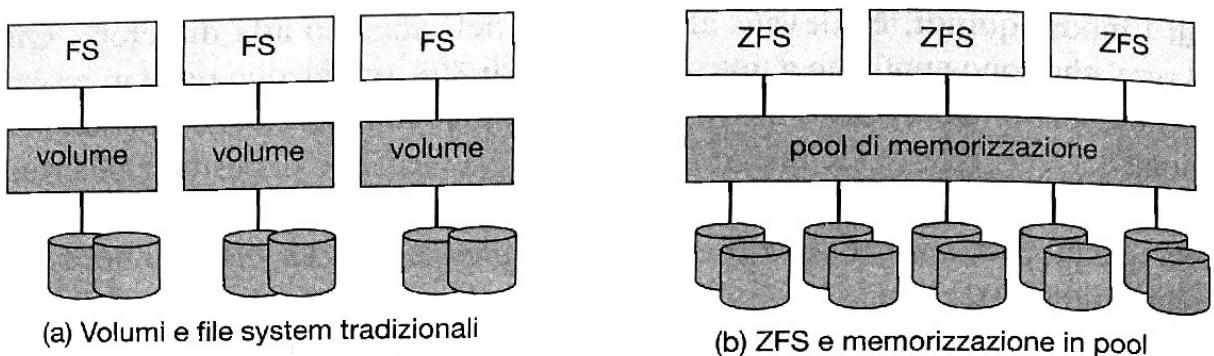


Figura 10.14 (a) Volumi e file system tradizionali. (b) Pool e file system ZFS.

10.8 Realizzazione della memoria stabile

In molte applicazioni relative alle basi di dati è necessario disporre di una **memoria stabile**. Per definizione, le informazioni residenti in questo tipo di memoria non vanno *mai* perse. Per realizzare un tale tipo di memoria si devono replicare le informazioni necessarie in più dispositivi di memorizzazione (di solito dischi) con meccanismi di guasto indipendenti. Inoltre è necessario coordinare l'aggiornamento delle informazioni in modo tale che un eventuale malfunzionamento durante l'aggiornamento non lasci tutte le copie in uno stato danneggiato, e che ripristinando le informazioni a seguito di un guasto sia possibile riportare ogni copia alla sua forma coerente e corretta anche se si verifica un altro malfunzionamento proprio durante il ripristino. Nel resto del paragrafo si presentano le possibili soluzioni a queste esigenze.

Un'operazione di scrittura in un disco può avere uno dei seguenti esiti.

1. **Operazione riuscita.** I dati sono stati scritti correttamente nel disco.
2. **Insuccesso parziale.** Si è verificato un guasto durante il trasferimento, e solo alcuni tra i settori coinvolti sono stati correttamente aggiornati, mentre il settore interessato dalla scrittura al momento del malfunzionamento può essere stato danneggiato.
3. **Insuccesso totale.** Il malfunzionamento è avvenuto prima dell'avvio del processo di scrittura nel disco; i dati già residenti nel disco sono rimasti inalterati.

Se si verifica un guasto durante la scrittura di un blocco, il sistema deve rilevarlo e invocare una procedura di ripristino per riportare il blocco in uno stato coerente. A tale scopo il sistema deve mantenere due blocchi fisici per ciascun blocco logico, eseguendo ogni scrittura nel modo seguente:

1. scrittura delle informazioni nel primo blocco fisico;
2. completata con successo la prima scrittura, scrittura delle stesse informazioni nel secondo blocco fisico;
3. l'operazione è considerata completa solamente dopo che la seconda scrittura è stata eseguita correttamente.

Durante il ripristino da un malfunzionamento si esamina ogni coppia di blocchi fisici; se essi contengono gli stessi dati e non c'è traccia d'errori, non è necessario intraprendere alcuna azione ulteriore. Se un blocco contiene un errore riscontrabile, se ne sostituisce il contenuto con quello del secondo blocco. Se in nessuno dei due blocchi si riscontra un errore, ma ciascuno contiene informazioni differenti, si sostituisce il contenuto del primo blocco con quello del secondo. Questa procedura di ripristino assicura che gli unici due esiti possibili di un'operazione di scrittura nella memoria stabile siano o la completa riuscita dell'operazione oppure il suo insuccesso totale, in quest'ultimo caso i dati memorizzati non subiscono alcun cambiamento.

Questa procedura si può facilmente estendere all'uso di un numero arbitrariamente grande di copie per ciascun blocco di memoria stabile. Benché un gran numero di queste copie riduca la probabilità di malfunzionamenti, di solito è ragionevole simulare la memoria stabile con due sole copie. I dati di una memoria stabile non andranno mai persi purché un guasto non distrugga tutte le copie.

Poiché le attese per il completamento delle scritture (I/O sincrono) richiedono molto tempo, molti array di memorizzazione aggiungono NVRAM come cache. Poiché la memoria è non volatile (di solito è dotata di una pila per l'alimentazione elettrica di riserva), si può ritenere affidabile per la memorizzazione dei dati in transito verso i dischi, e si può considerare parte della memoria stabile. Le scritture in NVRAM sono molto più rapide di quelle nei dischi, quindi le prestazioni migliorano notevolmente.

10.9 Sommario

Per la maggior parte dei calcolatori le unità a dischi sono il principale dispositivo di I/O di memoria secondaria. La gran parte dei dispositivi per la memorizzazione secondaria usa i dischi o i nastri magnetici, anche se i dischi a stato solido stanno acquisendo un'importanza sempre maggiore. Le moderne unità a disco sono strutturate come un grande array monodimensionale di blocchi logici, solitamente di 512 byte. I dischi possono essere collegati al computer in due modi: (1) tramite le porte per l'I/O locali della macchina o (2) tramite connessioni di rete.

Le richieste di I/O sui dischi sono generate sia dal file system sia dai sistemi di memoria virtuale, e ognuna di esse specifica l'indirizzo cui fare riferimento nel disco sotto forma di numero di un blocco logico. Gli algoritmi di scheduling per i dischi possono aumentare l'ampiezza di banda, e ridurre il tempo di risposta medio e la variabilità del tempo di risposta. Algoritmi come l'SSTF, SCAN, C-SCAN, LOOK e C-LOOK sono progettati per realizzare questi miglioramenti tramite criteri di ordinamento della coda di richieste di accesso ai dischi. Le prestazioni degli algoritmi di scheduling sui dischi magnetici possono variare notevolmente. Al contrario, nel caso dei dischi a stato solido, che non hanno parti in movimento, non ci sono grandi differenze di prestazioni tra i vari algoritmi e molto spesso viene utilizzata una semplice strategia FCFS.

Il sistema operativo gestisce i blocchi di un disco: innanzitutto si deve formattare fisicamente il disco per creare i settori direttamente sui suoi piatti – i dischi nuovi so-

no generalmente venduti già formattati; in seguito, il disco può essere diviso in partizioni, si può creare il file system, e si possono assegnare i blocchi d'avviamento che conterranno il programma d'avviamento del sistema; infine, quando un blocco diviene difettoso, il sistema deve essere in grado di isolarlo o di sostituirlo logicamente con un blocco di riserva.

Poiché un'area d'avvicendamento efficiente è essenziale per un sistema dalle buone prestazioni, molti sistemi bypassano il file system e usano un accesso di basso livello per l'I/O di paginazione. Certi sistemi riservano all'area d'avvicendamento una partizione di basso livello, altri impiegano un ordinario file all'interno del file system. Altri sistemi lasciano la scelta fra queste due possibilità all'utente o all'amministratore di sistema.

A causa della quantità di spazio di memorizzazione secondaria richiesta nei grandi sistemi, i dischi sono spesso resi ridondanti tramite algoritmi RAID. Questi algoritmi permettono l'uso di più dischi per una data operazione, e consentono la prosecuzione del funzionamento del sistema e anche il ripristino automatico dei dati a fronte del guasto di un disco. Gli algoritmi RAID sono classificati in livelli che offrono diverse combinazioni di affidabilità ed elevate velocità di trasferimento.

Esercizi di ripasso

- 10.1** Lo scheduling del disco, con algoritmi diversi dall'FCFS, è utile in un ambiente con un solo utente? Argomentate.
- 10.2** Spiegate perché l'SSTF tende a favorire i cilindri centrali rispetto a quelli più interni e più esterni.
- 10.3** Perché la latenza di rotazione non viene solitamente presa in considerazione nello scheduling del disco? Come potreste modificare lo scheduling SSTF, lo SCAN e quello C-SCAN per includervi l'ottimizzazione della latenza?
- 10.4** Perché è importante bilanciare gli I/O del file system tra i dischi e i controllori su un sistema in un ambiente multitasking?
- 10.5** Quali sono i tradeoff fra la rilettura delle pagine di codice da un file system e l'utilizzo dell'area di avvicendamento per memorizzarli?
- 10.6** Esiste un modo per realizzare una memorizzazione delle informazioni veramente stabile? Argomentate la risposta.
- 10.7** A volte un nastro è detto mezzo ad accesso sequenziale, mentre un disco magnetico è considerato un mezzo ad accesso diretto. In realtà, l'idoneità di un dispositivo di memorizzazione all'accesso diretto dipende dalla grandezza del trasferimento. Il termine *velocità di trasferimento in streaming* denota la velocità di un trasferimento di dati che è in corso, escluso l'effetto della latenza di accesso. La *velocità effettiva di trasferimento*, invece, è il rapporto dei byte totali per il totale dei secondi, inclusi i tempi di overhead come la latenza di accesso.

Ipotizzate che, in un computer, la cache di livello 2 abbia una latenza di accesso di 8 nanosecondi e una velocità di trasferimento in streaming di 800 megabyte al secondo, che la memoria principale abbia una latenza di accesso di 60 nanosecondi e una velocità di trasferimento in streaming di 80 megabyte al secondo, che il disco magnetico abbia una latenza di accesso di 15 millisecondi e una velocità di trasferimento in streaming di 5 megabyte al secondo, e che una unità a nastro abbia una latenza di accesso di 60 secondi e una velocità di trasferimento in streaming di 2 megabyte al secondo.

- L'accesso diretto causa la diminuzione della velocità effettiva di trasferimento di un dispositivo, perché non vengono trasferiti dati durante il tempo di accesso. Per il disco descritto, qual è la velocità di trasferimento effettiva se in media un accesso è seguito da un trasferimento in streaming di (1) 512 byte, (2) 8 kilobyte, (3) 1 megabyte, e (4) 16 megabyte?
- L'utilizzazione di un dispositivo è definita come il rapporto tra la velocità effettiva di trasferimento e la velocità di trasferimento in streaming. Calcolate l'utilizzazione dell'unità a disco per ognuna delle quattro grandezze di trasferimento indicate al punto a.
- Supponete che un'utilizzazione del 25 per cento o più sia considerata accettabile. Utilizzando i valori di prestazione indicati, calcolate la dimensione minima di trasferimento per un disco che dia un'utilizzazione accettabile.
- Completate la frase seguente: Un disco è un dispositivo ad accesso diretto per trasferimenti superiori a _____ byte ed è un dispositivo ad accesso sequenziale per trasferimenti inferiori.
- Calcolate le dimensioni minime di trasferimento che diano utilizzazioni accettabili per cache, memoria e nastro.
- Quando un nastro può essere considerato un dispositivo ad accesso diretto e quando invece è un dispositivo ad accesso sequenziale?

10.8 Può un'organizzazione RAID a livello 1 ottenere prestazioni migliori sulle richieste di lettura rispetto a un'organizzazione RAID a livello 0 (con striping senza ridondanza)? Se sì, come?

Esercizi

10.9 Eccetto l'FCFS, nessuno tra i criteri di scheduling del disco descritti è veramente *equo* (si può avere un'attesa indefinita).

- Spiegate perché questa affermazione è vera.
- Descrivete una maniera di modificare gli algoritmi come lo SCAN in modo che risultino equi.

- c. Spiegate perché l'equità è un obiettivo importante in un sistema a partizione del tempo.
- d. Date tre o più esempi di circostanze in cui è importante che il sistema operativo non sia equo nel servire le richieste di I/O.

10.10 Spiegate perché per i dischi SSD viene spesso utilizzato un algoritmo di scheduling FCFS.

10.11 Supponete che un'unità a disco abbia 5000 cilindri numerati da 0 a 4999. L'unità serve attualmente una richiesta relativa al cilindro 2150, e la richiesta precedente era relativa al cilindro 1850. La coda di richieste inevase, in ordine FIFO, è composta di richieste riguardanti i cilindri

2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681

Partendo dalla posizione attuale della testina, calcolate la distanza totale (in cilindri) che il braccio del disco percorre per soddisfare tutte le richieste inevase usando i seguenti algoritmi di scheduling:

- a. FCFS;
- b. SSTF;
- c. SCAN;
- d. LOOK;
- e. C-SCAN;
- f. C-LOOK.

10.12 È noto dalla fisica elementare che quando un oggetto è sottoposto a un'accelerazione costante a , la relazione fra la distanza d e il tempo t è data da $d = 1/2 at^2$. Supponete che l'unità a disco dell'Esercizio 10.11, durante la ricerca di un cilindro, imprima al braccio del disco un'accelerazione costante per la prima metà del tragitto richiesto dalla ricerca, e imprima invece una decelerazione costante della stessa intensità per la seconda metà del tragitto. Ipotizzate che l'unità possa portare a termine in 1 millisecondo la ricerca di un cilindro adiacente, e in 18 millisecondi una ricerca a tutto raggio lungo i 5000 cilindri.

- a. La distanza di una ricerca è il numero di cilindri attraverso i quali la testina deve passare. Spiegate perché il tempo di ricerca è proporzionale alla radice quadrata della distanza percorsa.
- b. Scrivete il tempo di ricerca in funzione della distanza da percorrere. L'equazione dovrebbe avere la forma $t = x + y \sqrt{L}$, dove t è il tempo in millisecondi e L è la distanza da percorrere in cilindri.
- c. Calcolate il tempo totale di ricerca per ogni algoritmo di scheduling dell'Esercizio 10.11 relativamente alla coda di richieste lì descritta. Determinate quale algoritmo sia più veloce, cioè implica un tempo di ricerca totale minore.

d. L'*aumento percentuale di velocità* è il tempo risparmiato diviso il tempo originariamente necessario. Calcolate l'aumento percentuale di velocità dell'algoritmo più veloce rispetto all'FCFS.

10.13 Supponete che il disco dell'Esercizio 10.12 ruoti alla velocità di 7200 giri al minuto.

- Calcolate la latenza di rotazione media di quest'unità a disco.
- Dite quale distanza di ricerca è possibile coprire nel tempo calcolato al punto a).

10.14 Descrivete vantaggi e svantaggi dell'utilizzo di unità SSD come livello aggiuntivo di cache e come unico dispositivo rispetto all'utilizzo esclusivo di un disco magnetico.

10.15 Confrontate le prestazioni di SCAN e C-SCAN assumendo una distribuzione uniforme delle richieste di I/O. Considerate il tempo di risposta medio (cioè il tempo che intercorre fra l'arrivo di una richiesta e il completamento dell'operazione a essa associata), la variazione del tempo di risposta, e l'effettiva ampiezza di banda. Analizzate come le prestazioni dipendano dalle dimensioni relative del tempo di ricerca e della latenza di rotazione.

10.16 Le richieste non sono di solito uniformemente distribuite: per esempio un cilindro che contiene metadati del file system potrebbe essere visitato più spesso di un cilindro che contiene solo file. Supponete di sapere che il 50 per cento delle richieste sia relativo a un piccolo numero fisso di cilindri.

- Dite se uno fra gli algoritmi illustrati in questo capitolo sia particolarmente adatto a questa circostanza. Motivate la risposta.
- Proponete un algoritmo di scheduling che offra prestazioni anche migliori sfruttando questo "punto caldo" del disco.

10.17 Considerate un sistema RAID a livello 5, comprensivo di cinque dischi; nel quinto disco risiedono le parità per gli insiemi di quattro blocchi di quattro dischi. A quanti blocchi si deve accedere per effettuare le operazioni seguenti:

- scrittura di un blocco di dati;
- scrittura di sette blocchi contigui di dati.

10.18 Ponete a confronto il throughput ottenuto attraverso un sistema RAID a livello 5 con quello ottenuto mediante un sistema RAID a livello 1, per quel che riguarda:

- operazioni di lettura su blocchi singoli;
- operazioni di lettura su blocchi multipli contigui.

10.19 Paragonate le prestazioni raggiunte da un sistema RAID a livello 5 con quelle di un sistema RAID a livello 1, relativamente alle operazioni di scrittura.

10.20 Assumete di avere una configurazione mista, che comprenda dischi strutturati secondo il sistema RAID a livello 1, e altri dischi a livello RAID 5. Supponente che il sistema, per memorizzare un determinato file, sia libero di optare per l'una o l'altra soluzione. Quali file dovrebbero essere memorizzati nei dischi a livello RAID 1 e quali nei dischi a livello RAID 5, allo scopo di ottimizzare le prestazioni?

10.21 L'affidabilità di un'unità a disco generalmente si quantifica usando il **tempo medio fra due guasti** (*mean time between failures*, MTBF); sebbene sia chiamata “tempo”, questa quantità in effetti si misura in ore di funzionamento dell’unità per guasto.

- a. Dato un gruppo di 1000 unità a disco, ognuna delle quali ha un MTBF di 750.000 ore, dite con quale frequenza avverrà il guasto di un’unità del gruppo, scegliendo fra le seguenti possibilità quella che si adatta meglio alla situazione descritta: una volta ogni mille anni, una volta ogni cento anni, una volta ogni dieci anni, una volta al mese, una volta alla settimana, una volta al giorno, una volta ogni ora, una volta al minuto, una volta al secondo.
- b. Le statistiche di mortalità indicano che in media un individuo residente negli Stati Uniti d’America ha circa 1 probabilità su 1000 di morire fra i 20 e i 21 anni d’età. Deducete l’MTBF di un ventenne, e convertite il risultato da ore in anni. Spiegate che cosa dice questo MTBF sulle aspettative di vita di un ventenne.
- c. Un produttore asserisce che un certo modello di unità a disco abbia un MTBF di 1 milione di ore. Dite cosa si può concludere circa il numero di anni per cui una di queste unità a disco è coperta dalla garanzia.

10.22 Esponete vantaggi e svantaggi relativi all'accantonamento dei settori e alla traslazione dei settori.

10.23 Esponete i motivi per cui il sistema operativo potrebbe necessitare di informazioni accurate sulle modalità di memorizzazione dei blocchi sul disco. In termini di miglioramento delle prestazioni del file system, in che modo il sistema operativo può mettere a frutto queste informazioni?

Problemi di programmazione

10.24 Scrivete un programma che implementa i seguenti algoritmi di scheduling del disco:

- a. FCFS
- b. SSTF
- c. SCAN
- d. C-SCAN
- e. LOOK
- f. C-LOOK

Il vostro programma servirà un disco con 5000 cilindri numerati da 0 a 4.999. Il programma genererà una sequenza casuale di 1000 richieste e le servirà secondo ciascuno degli algoritmi sopra elencati. Al programma sarà passata, come parametro da riga di comando, la posizione iniziale della testina del disco. Il programma restituirà la quantità totale di movimenti della testina richiesti da ciascun algoritmo.

Note bibliografiche

[Services 2012] fornisce una panoramica dei sistemi di memorizzazione dei dati in una varietà di ambienti elaborativi moderni. [Teorey e Pinkerton 1972] presentano un'analisi comparativa di algoritmi di scheduling del disco utilizzando simulazioni che modellano un disco per il quale il tempo di ricerca è lineare nel numero di cilindri traversati. Le ottimizzazioni dello scheduling che sfruttano i tempi morti (idle time) dei dischi sono discusse in [Lumb et al. 2000]. [Kim et al. 2009] trattano gli algoritmi di scheduling per le unità SSD.

Le batterie ridondanti di dischi (RAID) sono presentate da [Patterson et al. 1988]

In [Russinovich e Solomon 2009], [McDougall e Mauro 2007] e [Love 2010] sono trattati in dettaglio i file system di Windows, Solaris e Linux, rispettivamente.

La dimensione dei trasferimenti richiesti e la casualità del carico di lavoro hanno un'influenza considerevole sulle prestazioni dell'unità a disco. [Ousterhout et al. 1985], e [Ruemmler e Wilkes 1993] riportano numerose interessanti caratteristiche dei carichi di lavoro, per esempio che la maggior parte dei file sono brevi, la maggior parte dei file creati di recente è eliminata assai presto, che il più delle volte i file aperti per lettura sono letti in modo sequenziale nella loro interezza, e che nella maggior parte dei casi le distanze di ricerca sono brevi.

Il concetto di gestione gerarchica della memorizzazione è stato studiato per più di quarant'anni: un articolo di [Mattson et al. 1970], per esempio, descrive un metodo matematico per prevedere le prestazioni di un sistema di gestione gerarchica della memorizzazione.

Bibliografia

- [Kim et al. 2009] J.Kim, Y.Oh, E.Kim, J.C.D. Lee e S.Noh, “Disk schedulers for solid state drivers”, p. 295–304, 2009.
- [Love 2010] R. Love, Linux Kernel Development, 3° Ed., Developer’s Library, 2010.
- [Lumb et al. 2000] C. Lumb, J. Schindler, G. R. Ganger, D. F. Nagle e E. Riedel, “Towards Higher Disk Head Utilization: Extracting Free Bandwidth From Busy Disk Drives”, Symposium on Operating Systems Design and Implementation, 2000.

- [Mattson et al. 1970] R. L. Mattson, J. Gecsei, D. R. Slutz e I. L. Traiger, "Evaluation Techniques for Storage Hierarchies", IBM Systems Journal, Vol. 9, Num. 2, p. 78–117, 1970.
- [McDougall e Mauro 2007] R. McDougall e J. Mauro, *Solaris Internals*, 2° Ed., Prentice Hall, 2007.
- [Ousterhout et al. 1985] J.K.Ousterhout, H.D., Costa, D. Harrison, J.A. Kunze, M. Kupfer e J. G. Thompson, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System", Proceedings of the ACM Symposium on Operating Systems Principles, p. 15–24, 1985.
- [Patterson et al. 1988] D. A. Patterson, G. Gibson e R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", Proceedings of the ACM SIGMOD International Conference on the Management of Data, p. 109–116, 1988.
- [Ruemmler e Wilkes 1993] C. Ruemmler e J. Wilkes, "Unix Disk Access Patterns", Proceedings of the Winter USENIX Conference 1993, p. 405–420.
- [Russinovich e Solomon 2009] M. E. Russinovich e D. A. Solomon, *Windows Internals: Including Windows Server 2008 and Windows Vista*, 5° Ed., Microsoft Press, 2009.
- [Services 2012] E. E. Services, *Information Storage and Management: Storing, Managing, and Protecting Digital Information in Classic, Virtualized, and Cloud Environments*, Wiley, 2012.
- [Teorey e Pinkerton 1972] T. J. Teorey e T. B. Pinkerton, "A Comparative Analysis of Disk Scheduling Policies", Communications of the ACM, Vol. 15, Num. 3, p. 177–184, 1972.