Capítulo 21

Discos rígidos

21.1 Introdução

Discos rígidos estão presentes na grande maioria dos computadores pessoais e servidores. Um disco rígido permite o armazenamento persistente (não-volátil) de grandes volumes de dados com baixo custo e tempos de acesso razoáveis. Além disso, a leitura e escrita de dados em um disco rígido é mais simples e flexível que em outros meios, como fitas magnéticas ou discos óticos (CDs, DVDs). Por essas razões, eles são intensivamente utilizados em computadores para o armazenamento de arquivos do sistema operacional, das aplicações e dos dados dos usuários. Os discos rígidos também são frequentemente usados como área de armazenamento de páginas em sistemas de paginação em disco (*swapping* e *paging*, Capítulo 17).

Este capítulo inicialmente apresenta alguns aspectos de hardware relacionados aos discos rígidos, como sua estrutura física e os principais padrões de interface entre o disco e sua controladora no computador. Em seguida, detalha aspectos de software que estão sob a responsabilidade direta do sistema operacional, como o escalonamento de operações de leitura/escrita no disco. Por fim, apresenta a estratégia RAID para a composição de discos rígidos, que visam melhorar seu desempenho e/ou confiabilidade.

21.2 Estrutura física

Um disco rígido é composto por um ou mais discos metálicos que giram juntos em alta velocidade (usualmente entre 4.200 e 15.000 RPM), acionados por um motor elétrico. Para cada face de cada disco há uma cabeça de leitura móvel, responsável por ler e escrever dados através da magnetização de pequenas áreas da superfície metálica. Cada face é dividida logicamente em **trilhas** (ou **cilindros**) e **setores**; a interseção de uma trilha e um setor em uma face define um **bloco físico**[1], que é a unidade básica de armazenamento e transferência de dados no disco. Até 2010, os discos rígidos usavam blocos físicos de 512 bytes, mas o padrão da indústria migrou nos últimos anos para blocos de 4.096 bytes. A Figura 21.1 apresenta os principais elementos que compõem a estrutura de um disco rígido.

Um disco típico contém várias faces e milhares de trilhas e de setores por face [Patterson and Henessy, 2005], resultando em milhões de blocos de dados disponíveis. Cada bloco pode ser individualmente acessado (lido ou escrito) através de seu *endereço*.

[1] Alguns autores denominam essa intersecção como "setor de trilha" (*track sector*).

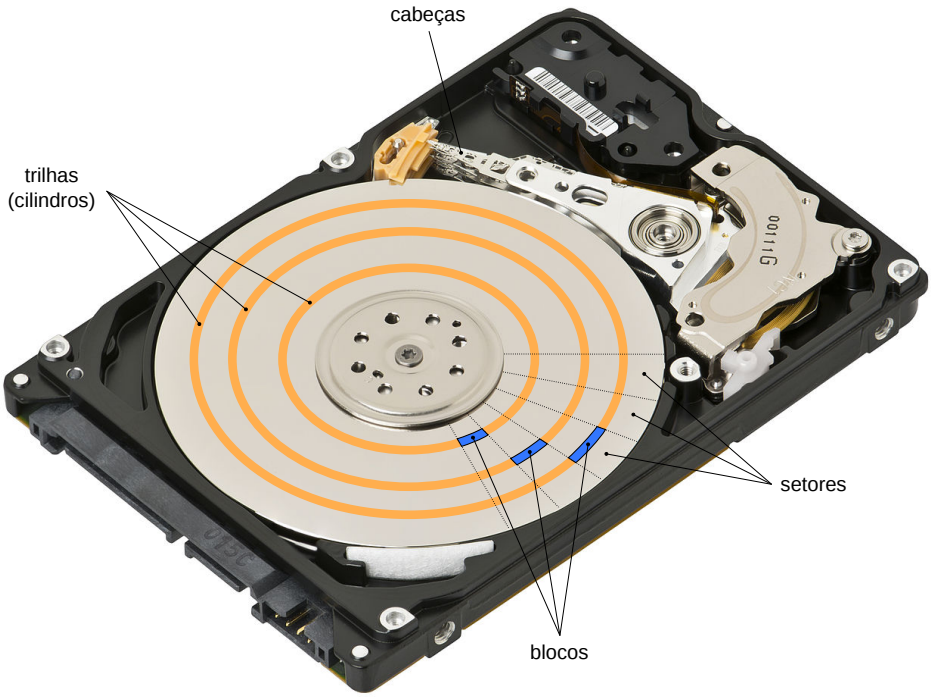

Figura 21.1: Elementos da estrutura de um disco rígido.

Historicamente, o endereçamento dos blocos usava um padrão denominado CHS (*Cylinder-Head-Sector*): para acessar cada bloco, era necessário informar a cabeça (ou seja, a face), o cilindro (trilha) e o setor do disco onde se encontra o bloco. Esse sistema foi mais tarde substituído pelo padrão LBA (*Logical Block Addressing*), no qual os blocos são endereçados linearmente (0, 1, 2, 3, ...), o que é muito mais fácil de gerenciar pelo sistema operacional. Como a estrutura física do disco rígido continua a ter faces, trilhas e setores, uma conversão entre endereços LBA e CHS é feita pelo firmware do disco rígido, de forma transparente para o restante do sistema.

Por serem dispositivos eletromecânicos, os discos rígidos são extremamente lentos, se comparados à velocidade da memória ou do processador. Para cada bloco a ser lido/escrito, a cabeça de leitura deve se posicionar na trilha desejada e aguardar o disco girar até encontrar o setor desejado. Esses dois passos definem o *tempo de busca* (t_s – *seek time*), que é o tempo necessário para a cabeça de leitura se posicionar sobre uma determinada trilha, e a *latência rotacional* (t_r – *rotation latency*), que é o tempo necessário para o disco girar até que o setor desejado esteja sob a cabeça de leitura. Valores médios típicos desses atrasos para discos de uso doméstico são $t_s \approx 10ms$ e $t_r \approx 5ms$. Juntos, esses dois atrasos podem ter um forte impacto no desempenho do acesso a disco.

21.3 Interface de acesso

Como visto na seção 20.2, o sistema operacional deve dispor de *drivers* para interagir com cada controlador de disco e solicitar operações de escrita e leitura de dados. O disco rígido é um dispositivo orientado a blocos, então cada operação de leitura/escrita é feita sobre blocos físicos individuais ou *clusters* (grupos de 2^n blocos físicos contíguos). O acesso ao disco rígido normalmente é feito usando interação por

eventos (Seção 20.5.2): o *driver* solicita uma operação de E/S ao controlador do disco rígido e suspende o fluxo de execução solicitante; quando a operação é completada, o controlador de disco gera uma interrupção que é encaminhada ao *driver*, para retomar a execução e/ou solicitar novas operações. A estratégia de acesso direto à memória (DMA, Seção 20.5.3) também é frequentemente usada, para aumentar o desempenho de transferência de dados.

O *controlador* de disco rígido normalmente é conectado a um barramento do computador. Por sua vez, os discos são conectados ao controlador através de uma interface de conexão que pode usar diversas tecnologias. As mais comuns estão descritas a seguir:

- **IDE**: *Integrated Drive Electronics*, padrão também conhecido como PATA (*Parallell ATA - Advanced Technology Attachment*); surgiu nos anos 1980 e durante muito tempo foi o padrão de interface de discos mais usado em computadores pessoais. Suporta velocidades de até 1 Gbit/s, através de cabos paralelos de 40 ou 80 vias. Cada barramento IDE suporta até dois dispositivos, em uma configuração mestre/escravo.

- **SCSI**: *Small Computer System Interface*, padrão de interface desenvolvida nos anos 1980, foi muito usada em servidores e estações de trabalho de alto desempenho. Um barramento SCSI suporta até 16 dispositivos e atinge taxas de transferência de até 2,5 Gbit/s (divididos entre os dispositivos conectados no mesmo barramento).

- **SATA**: *Serial ATA*, é o padrão de interface de discos em *desktops* e *notebooks* atuais. A transmissão dos dados entre o disco e a controladora é serial, atingindo taxas de transferência de 6 Gbit/s através de cabos com 7 vias.

- **SAS**: *Serial Attached SCSI*, é uma evolução do padrão SCSI, permitindo atingir taxas de transferência de até 12 Gbit/s em cada dispositivo conectado ao controlador. É usado em equipamentos de alto desempenho, como servidores.

É importante observar que esses padrões de interface não são de uso exclusivo em discos rígidos, muito pelo contrário. Há vários tipos de dispositivos que podem se conectar ao computador através dessas interfaces, como discos de estado sólido (SSD), leitores óticos (CD, DVD), unidades de fita magnética, *scanners*, etc.

21.4 Escalonamento de acessos

Em um sistema operacional multitarefas, várias aplicações e processos podem solicitar acessos ao disco simultaneamente, para escrita e leitura de dados. Devido à sua estrutura mecânica, um disco rígido só pode atender a uma requisição de acesso por vez, o que torna necessário criar uma fila de acessos pendentes. Cada nova requisição de acesso ao disco é colocada nessa fila e o processo solicitante é suspenso até seu pedido ser atendido. Sempre que o disco concluir um acesso, ele informa o sistema operacional, que deve buscar nessa fila a próxima requisição de acesso a ser atendida. A ordem de atendimento das requisições pendentes na fila de acesso ao disco é denominada **escalonamento de disco** e pode ter um grande impacto no desempenho do sistema operacional.

Na sequência do texto serão apresentados alguns algoritmos de escalonamento de disco clássicos. Para exemplificar seu funcionamento, será considerado um disco hipotético com 1.000 blocos (enumerados de 0 ao 999), cuja cabeça de leitura se encontra inicialmente sobre o bloco 500. A fila de pedidos de acesso pendentes contém pedidos de acesso aos seguintes blocos do disco, em sequência:

$$278,\ 914,\ 447,\ 71,\ 161,\ 659,\ 335$$

Todos esses pedidos são de processos distintos, portanto podem ser atendidos em qualquer ordem. Para simplificar, considera-se que nenhum pedido de acesso novo chegará à fila durante a execução do algoritmo de escalonamento.

FCFS (*First Come, First Served*): este algoritmo consiste em atender as requisições na ordem da fila, ou seja, na ordem em foram pedidas pelos processos. É a estratégia mais simples de implementar, mas raramente oferece um bom desempenho. Se os pedidos de acesso estiverem muito espalhados pelo disco, este irá perder muito tempo movendo a cabeça de leitura de um lado para o outro. A sequência de blocos percorridos pela cabeça de leitura é:

$$500 \xrightarrow{222} 278 \xrightarrow{636} 914 \xrightarrow{467} 447 \xrightarrow{376} 71 \xrightarrow{90} 161 \xrightarrow{498} 659 \xrightarrow{324} 335 \text{ (2.613 blocos)}$$

Percebe-se que, para atender os pedidos de leitura na ordem indicada pelo algoritmo FCFS, a cabeça de leitura teve de deslocar-se por 2.613 blocos do disco ($222 + 636 + 467 + \cdots$). A Figura 21.2 mostra os deslocamentos da cabeça de leitura para atender os pedidos de acesso da fila de exemplo.

SSTF (*Shortest Seek Time First – Menor Tempo de Busca Primeiro*): esta estratégia de escalonamento de disco consiste em sempre atender o pedido que está mais próximo da posição atual da cabeça de leitura (que é geralmente a posição do último pedido atendido). Dessa forma, ela busca reduzir os movimentos da cabeça de leitura, e com isso o tempo perdido entre os acessos.

A sequência de acesso efetuadas pelo algoritmo SSTF está indicada a seguir e na Figura 21.3. Pode-se observar uma grande redução da movimentação da cabeça de leitura em relação à estratégia FCFS, que passou de 2.613 para 1.272 blocos percorridos. Contudo, a estratégia SSTF não garante obter sempre um percurso mínimo.

$$500 \xrightarrow{53} 447 \xrightarrow{112} 335 \xrightarrow{57} 278 \xrightarrow{117} 161 \xrightarrow{90} 71 \xrightarrow{588} 659 \xrightarrow{255} 914 \text{ (1.272 blocos)}$$

Apesar de oferecer um ótimo desempenho, a estratégia SSTF pode levar à inanição (*starvation*) de requisições de acesso: caso existam muitas requisições em uma determinada região do disco, pedidos de acesso a blocos distantes dessa região podem ficar esperando indefinidamente. Para resolver esse problema, torna-se necessário implementar uma estratégia de *envelhecimento* dos pedidos pendentes.

SCAN: neste algoritmo, a cabeça "varre" (*scan*) continuamente o disco, do início ao final, atendendo os pedidos que encontra pela frente; ao atingir o final do

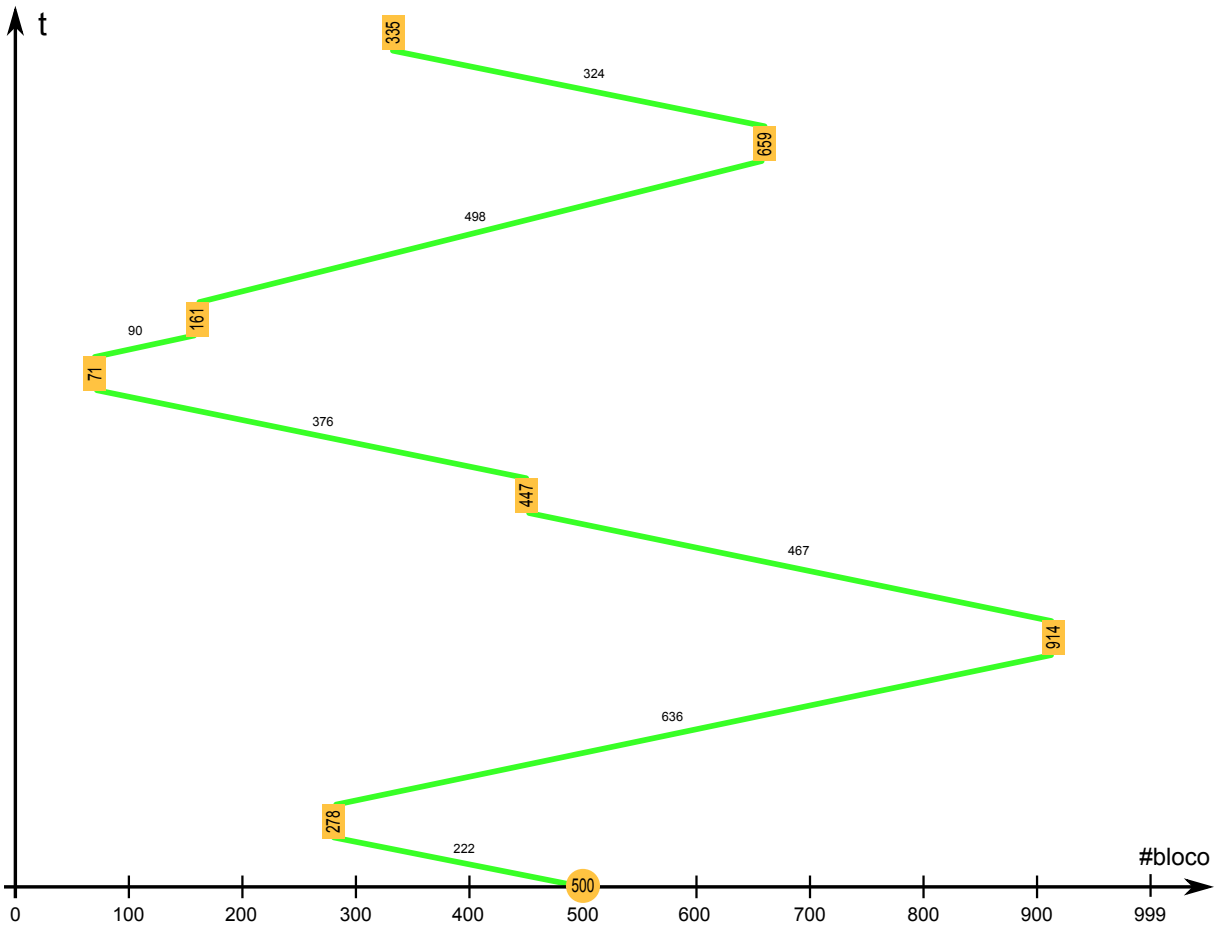

Figura 21.2: Escalonamento de disco FCFS.

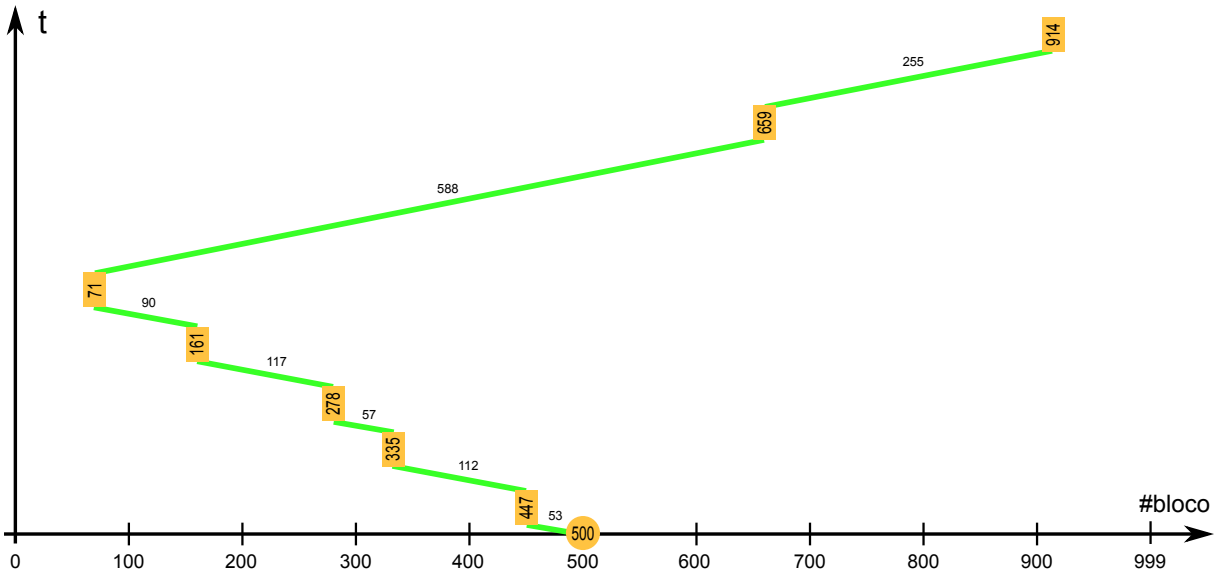

Figura 21.3: Escalonamento de disco SSTF.

disco, ela inverte seu sentido de movimento e volta, atendendo os próximos pedidos. Apesar de ser mais lento que SSTF, este algoritmo atende os pedidos de forma mais uniforme ao longo do disco, eliminando o risco de inanição de pedidos e mantendo um desempenho equilibrado para todos os processos.

Ele é adequado para sistemas com muitos pedidos simultâneos de acesso a disco, como servidores de arquivos. O comportamento deste algoritmo para a sequência de requisições de exemplo está indicado a seguir e na Figura 21.4:

$$500 \xrightarrow{159} 659 \xrightarrow{255} 914 \xrightarrow{85} 999 \xrightarrow{552} 447 \xrightarrow{112}$$

$$\longrightarrow 335 \xrightarrow{57} 278 \xrightarrow{117} 161 \xrightarrow{90} 71 \text{ (1.427 blocos)}$$

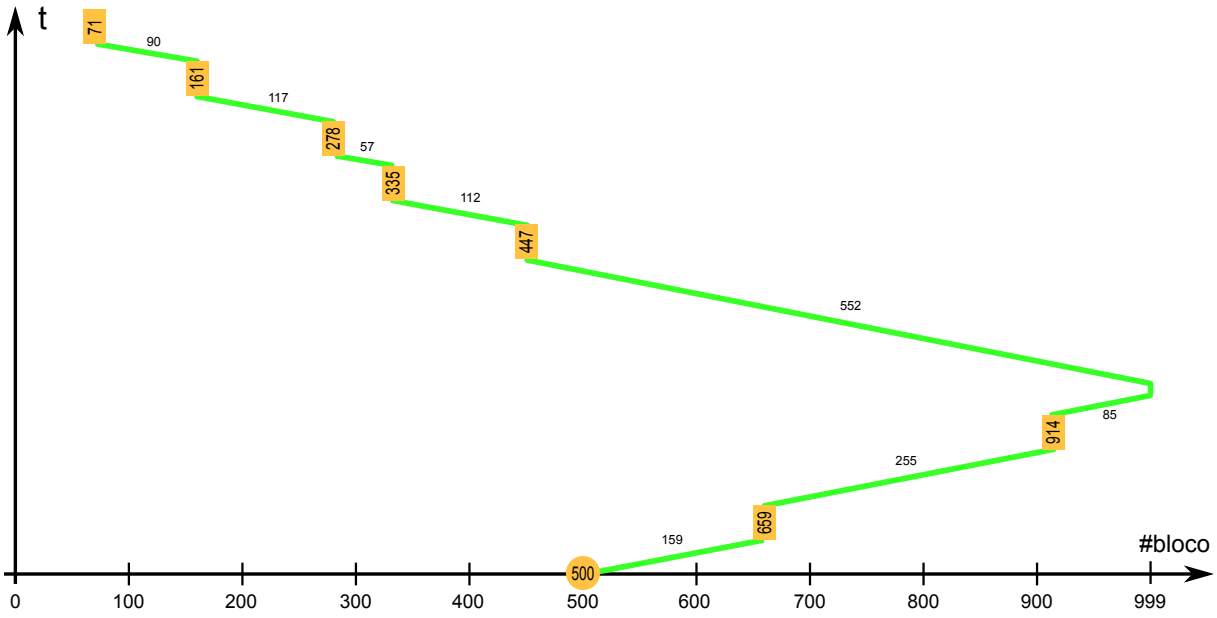

Figura 21.4: Escalonamento de disco com o algoritmo SCAN.

C-SCAN: esta é uma variante "circular" do algoritmo SCAN, na qual a cabeça de leitura varre o disco somente em um sentido. Ao atingir o final do disco, ela retorna diretamente ao início do disco, sem atender os pedidos intermediários, e recomeça a varredura. O nome "circular" é devido ao disco ser visto pelo algoritmo como uma lista circular de blocos. Sua vantagem em relação ao algoritmo SCAN é prover um tempo de espera mais homogêneo aos pedidos pendentes, o que é importante em servidores. O comportamento deste algoritmo para a sequência de requisições de exemplo está indicado a seguir e na Figura 21.5:

$$500 \xrightarrow{159} 659 \xrightarrow{255} 914 \xrightarrow{85} 999 \xrightarrow{999} 0 \xrightarrow{71} 71 \xrightarrow{90}$$

$$\longrightarrow 161 \xrightarrow{117} 278 \xrightarrow{57} 335 \xrightarrow{112} 447 \text{ (1.945 blocos)}$$

LOOK: é uma otimização do algoritmo SCAN, na qual a cabeça do disco não avança até o final do disco, mas inverte seu movimento assim que tiver tratado o último pedido em cada sentido do movimento:

$$500 \xrightarrow{159} 659 \xrightarrow{255} 914 \xrightarrow{467} 447 \xrightarrow{112} 335 \xrightarrow{57} 278 \xrightarrow{117} 161 \xrightarrow{90} 71 \text{ (1.257 blocos)}$$

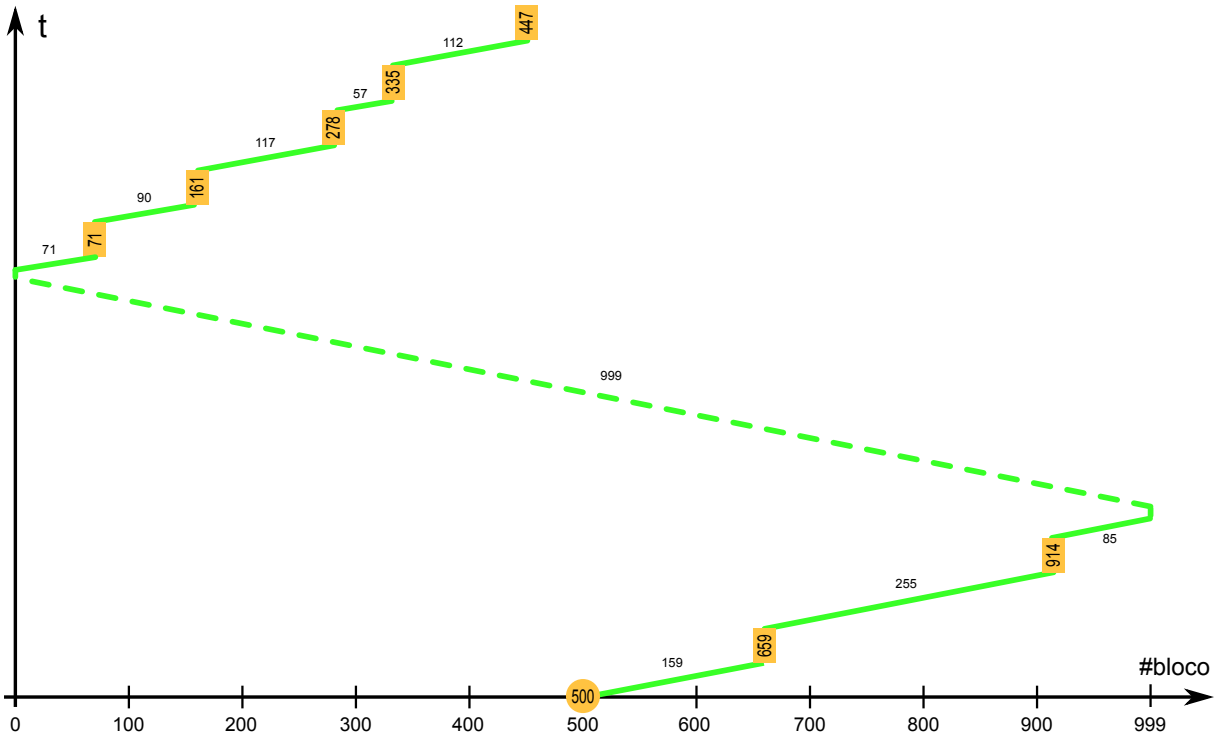

Figura 21.5: Escalonamento de disco com o algoritmo C-SCAN.

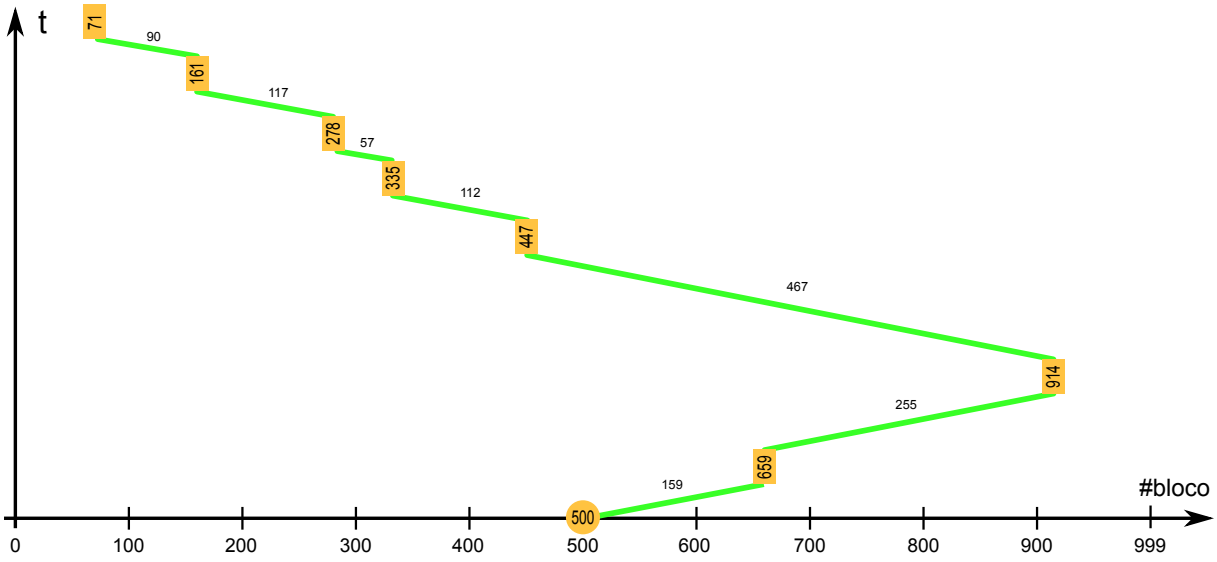

Figura 21.6: Escalonamento de disco com o algoritmo LOOK.

C-LOOK: idem, otimizando o algoritmo C-SCAN:

$$500 \xrightarrow{159} 659 \xrightarrow{255} 914 \xrightarrow{843} 71 \xrightarrow{90} 161 \xrightarrow{117} 278 \xrightarrow{57} 335 \xrightarrow{112} 447 \text{ (1.644 blocos)}$$

Os algoritmos SCAN, C-SCAN e suas variantes LOOK e C-LOOK são denominados coletivamente de *Algoritmo do Elevador*, pois seu comportamento reproduz o comportamento do elevador em um edifício: a cabeça de leitura avança em um sentido, atendendo as requisições dos usuários; ao chegar ao final, inverte seu sentido e retorna [Silberschatz et al., 2001].

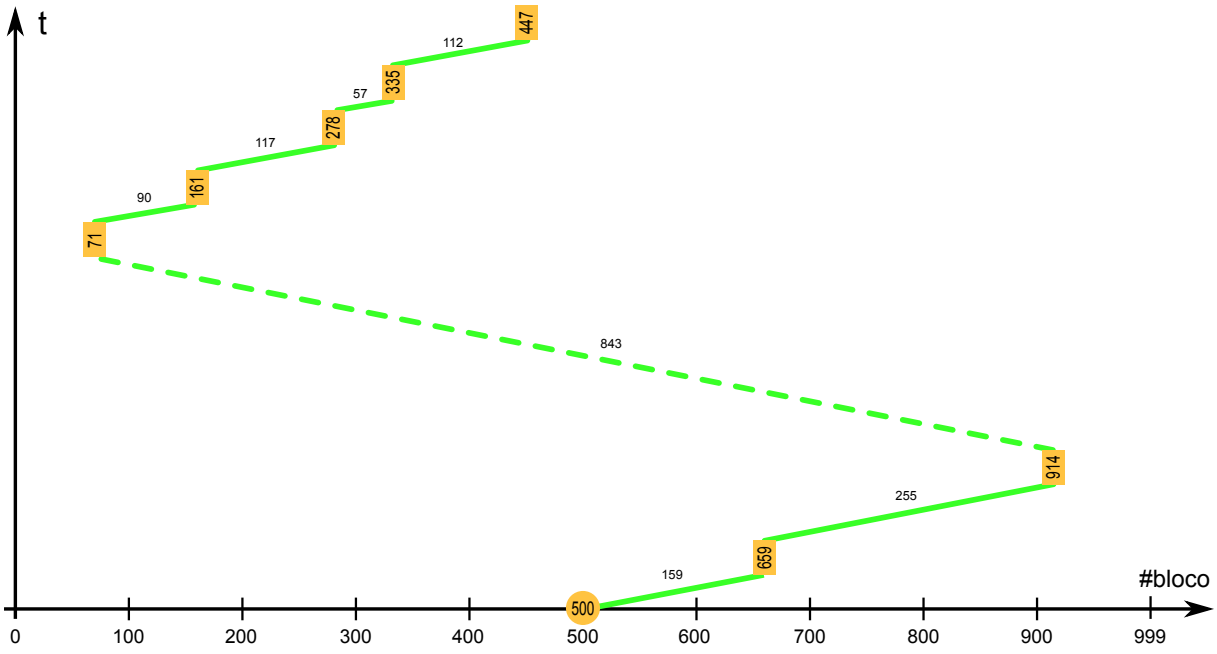

Figura 21.7: Escalonamento de disco com o algoritmo C-LOOK.

Sistemas operacionais reais, como Solaris, Windows e Linux, utilizam escalonadores de disco bem mais sofisticados. No caso do sistema Linux, por exemplo, os seguintes escalonadores de disco estão presentes no núcleo, podendo ser configurados pelo administrador do sistema em função das características dos discos e da carga de trabalho do sistema [Love, 2010; Bovet and Cesati, 2005]:

Noop (*No-Operation*): é o escalonador mais simples, baseado em FCFS, que não reordena os pedidos de acesso, apenas agrupa os pedidos direcionados ao mesmo bloco ou a blocos adjacentes. Este escalonador é voltado para discos de estado sólido (SSD, baseados em memória *flash*) ou sistemas de armazenamento que façam seu próprio escalonamento interno, como sistemas RAID (vide Seção 21.5).

Deadline: este escalonador é baseado no algoritmo do elevador circular (C-SCAN), mas associa um prazo (*deadline*) a cada requisição, para evitar problemas de inanição. Como os pedidos de leitura implicam no bloqueio dos processos solicitantes, eles recebem um prazo de 500 *ms*; pedidos de escrita podem ser executados de forma assíncrona, sem bloquear o processo solicitante, por isso recebem um prazo maior, de 5 segundos. O escalonador processa os pedidos usando o algoritmo do elevador, mas prioriza os pedidos cujos prazos estejam esgotando.

Anticipatory: este algoritmo é baseado no anterior (*deadline*), mas busca se antecipar às operações de leitura de dados feitas pelos processos. Como as operações de leitura são geralmente feitas de forma sequencial (em blocos contíguos ou próximos), a cada operação de leitura realizada o escalonador aguarda um certo tempo (por default 6 *ms*) por um novo pedido de leitura naquela mesma região do disco, que é imediatamente atendido. Caso não surja nenhum pedido novo, o escalonador volta a tratar a fila de pedidos pendentes normalmente. Essa espera por pedidos adjacentes melhora o desempenho das operações de leitura emitidas pelo mesmo processo.

CFQ (*Completely Fair Queuing*): os pedidos dos processos são divididos em várias filas (64 filas por default); cada fila recebe uma fatia de tempo para acesso ao disco, que varia de acordo com a prioridade de entrada/saída dos processos contidos na mesma. Este é o escalonador default do Linux na maioria das distribuições mais recentes.

21.5 Sistemas RAID

Apesar dos avanços dos sistemas de armazenamento em estado sólido (como os dispositivos baseados em memórias *flash*), os discos rígidos continuam a ser o principal meio de armazenamento não-volátil de grandes volumes de dados. Os discos atuais têm capacidades de armazenamento impressionantes: encontram-se facilmente no mercado discos rígidos com capacidade da ordem de terabytes para computadores domésticos.

Entretanto, o desempenho dos discos rígidos evolui a uma velocidade muito menor que a observada nos demais componentes dos computadores, como processadores, memórias e barramentos. Com isso, o acesso aos discos constitui um dos maiores gargalos de desempenhos nos sistemas de computação. Boa parte do baixo desempenho no acesso aos discos é devida aos aspectos mecânicos do disco, como a latência rotacional e o tempo de posicionamento da cabeça de leitura do disco (vide Seção 21.4) [Chen et al., 1994].

Outro problema relevante associado aos discos rígidos diz respeito à sua confiabilidade. Os componentes internos do disco podem falhar, levando à perda de dados. Essas falhas podem estar localizadas no meio magnético, ficando restritas a alguns setores, ou podem estar nos componentes mecânicos/eletrônicos do disco, levando à corrupção ou mesmo à perda total dos dados armazenados.

Buscando soluções eficientes para os problemas de desempenho e confiabilidade dos discos rígidos, pesquisadores da Universidade de Berkeley, na Califórnia, propuseram em 1988 a construção de discos virtuais compostos por conjuntos de discos físicos, que eles denominaram RAID – *Redundant Array of Inexpensive Disks*[2] [Patterson et al., 1988], que em português pode ser traduzido como *Conjunto Redundante de Discos Econômicos*.

Um sistema RAID é constituído de dois ou mais discos rígidos que são vistos pelo sistema operacional e pelas aplicações como um único disco lógico, ou seja, um grande espaço contíguo de armazenamento de dados. O objetivo central de um sistema RAID é proporcionar mais desempenho nas operações de transferência de dados, através do paralelismo no acesso aos vários discos, e também mais confiabilidade no armazenamento, usando mecanismos de redundância dos dados armazenados nos discos, como cópias de dados ou códigos corretores de erros.

Um sistema RAID pode ser construído "por hardware", usando uma placa controladora dedicada a esse fim, à qual estão conectados os discos rígidos. Essa placa controladora oferece a visão de um disco lógico único ao restante do computador. Também pode ser usada uma abordagem "por software", na qual são usados *drivers* apropriados dentro do sistema operacional para combinar os discos rígidos conectados ao computador em um único disco lógico. Obviamente, a solução por software é mais flexível e econômica, por não exigir uma placa controladora dedicada, enquanto a

[2]Mais recentemente alguns autores adotaram a expressão *Redundant Array of Independent Disks* para a sigla RAID, buscando evitar a subjetividade da palavra *Inexpensive* (econômico).

solução por hardware é mais robusta e tem um desempenho melhor. É importante observar que os sistemas RAID operam abaixo dos sistemas de arquivos, ou seja, eles se preocupam apenas com o armazenamento e recuperação de blocos de dados.

Há várias formas de se organizar um conjunto de discos rígidos em RAID, cada uma com suas próprias características de desempenho e confiabilidade. Essas formas de organização são usualmente chamadas *Níveis RAID*. Os níveis RAID padronizados pela *Storage Networking Industry Association* são [SNIA]:

RAID 0 (linear): neste nível os discos físicos (ou partições) são simplesmente concatenados em sequência para construir um disco lógico. Essa abordagem, ilustrada na Figura 21.8, é denominada por alguns autores de *RAID 0 linear*, enquanto outros a denominam JBoD (*Just a Bunch of Disks* – apenas um punhado de discos). Na figura, $d_i.b_j$ indica o bloco j do disco físico i.

Em teoria, esta estratégia oferece maior velocidade de leitura e de escrita, pois acessos a blocos em discos físicos distintos podem ser feitos em paralelo. Entretanto, esse ganho pode ser pequeno caso os acessos se concentrem em uma área pequena do disco lógico, pois ela provavelmente estará mapeada em um mesmo disco físico (o acesso a cada disco é sempre sequencial). Além disso, alguns discos tendem a ser mais usados que outros.

Esta abordagem não oferece nenhuma redundância de dados, o que a torna suscetível a erros de disco: caso um disco falhe, todos os blocos armazenados nele serão perdidos. Como a probabilidade de falhas aumenta com o número de discos, esta abordagem acaba por reduzir a confiabilidade do sistema de discos.

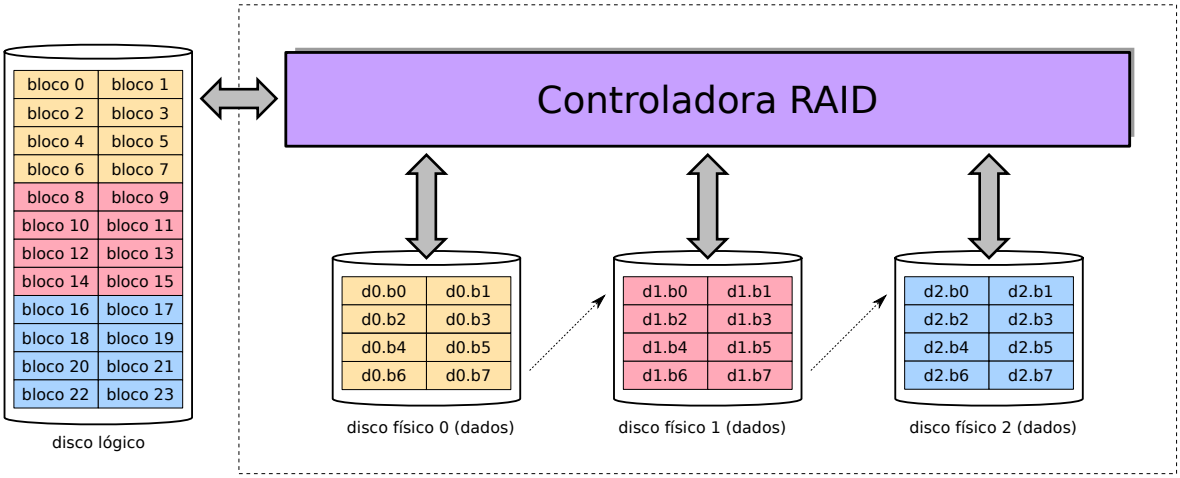

Figura 21.8: RAID nível 0 (*linear*).

RAID 0 (striping): neste nível os discos físicos são divididos em áreas de tamanhos fixo chamadas *fatias* ou *faixas* (*stripes*). Cada fatia de disco físico armazena um ou mais blocos do disco lógico (tipicamente são usadas faixas de 32, 64 ou 128 KBytes). As fatias são concatenadas usando uma estratégia *round-robin* para construir o disco lógico, como mostra a Figura 21.9.

O maior espalhamento dos blocos sobre os discos físicos contribui para distribuir melhor a carga de acessos entre eles e assim ter um melhor desempenho. Suas características de suporte a grande volume de dados e alto desempenho em

leitura/escrita tornam esta abordagem adequada para ambientes que precisam processar grandes volumes de dados temporários, como os sistemas de computação científica [Chen et al., 1994].

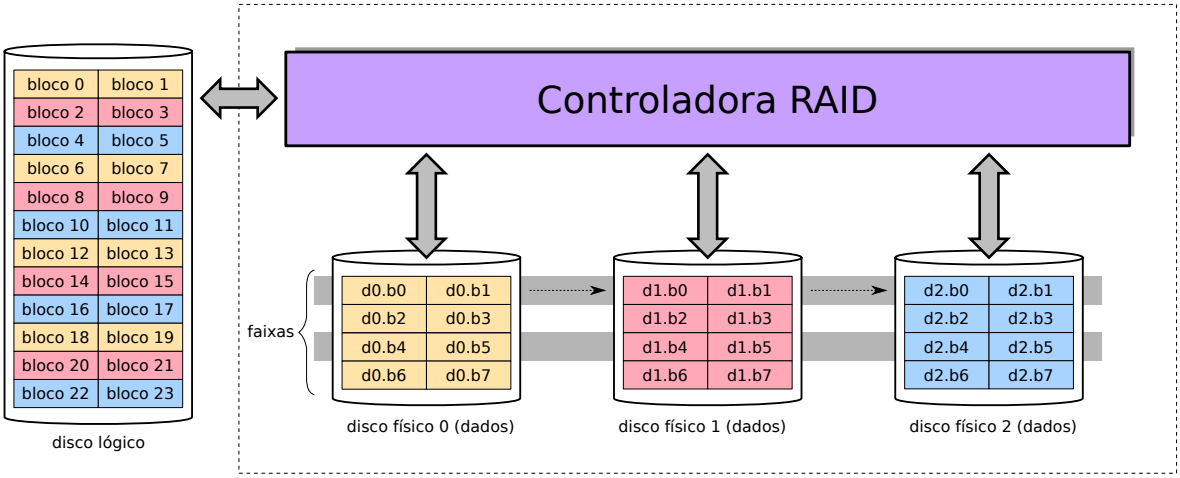

Figura 21.9: RAID nível 0 (*striping*).

RAID 1: neste nível, o conteúdo é replicado em dois ou mais discos, sendo por isso comumente chamado de *espelhamento de discos*. A Figura 21.10 mostra uma configuração usual deste nível de RAID, com dois discos físicos (embora não seja usual, o conteúdo pode ser replicado em N discos, para tolerar $N-1$ falhas). O espelhamento também pode ser associado a conjuntos de discos em RAID 0, levando a configurações híbridas como RAID 0+1, RAID 1+0 ou RAID 1E.

Esta abordagem oferece uma excelente confiabilidade, pois cada bloco lógico está escrito em dois ou mais discos distintos; caso um deles falhe, os demais continuam acessíveis. O desempenho em leituras também é beneficiado, pois a controladora pode distribuir as leituras entre as cópias dos dados. Contudo, não há ganho de desempenho em escrita, pois cada operação de escrita deve ser replicada em todos os discos. Além disso, seu custo de implantação é elevado, pois os n discos físicos são vistos como apenas um disco lógico do mesmo tamanho.

RAID 2: neste nível os dados são "fatiados" em bits individuais que são escritos nos discos físicos em sequência; discos adicionais são usados para armazenar códigos corretores de erros (*Hamming Codes*), em um arranjo similar ao usado nas memórias RAM. Esses códigos corretores de erros permitem resgatar dados no caso de falha em blocos ou discos de dados. Por ser pouco eficiente e complexo de implementar, este nível não é usado na prática.

RAID 3: de forma similar ao RAID 2, este nível fatia os dados em bytes escritos nos discos em sequência. Um disco adicional é usado para armazenar um byte com os bits de paridade dos bytes correspondentes em cada disco, sendo usado para a recuperação de erros nos demais discos. A cada leitura ou escrita, os dados de paridade devem ser atualizados, o que transforma o disco de paridade em um gargalo de desempenho.

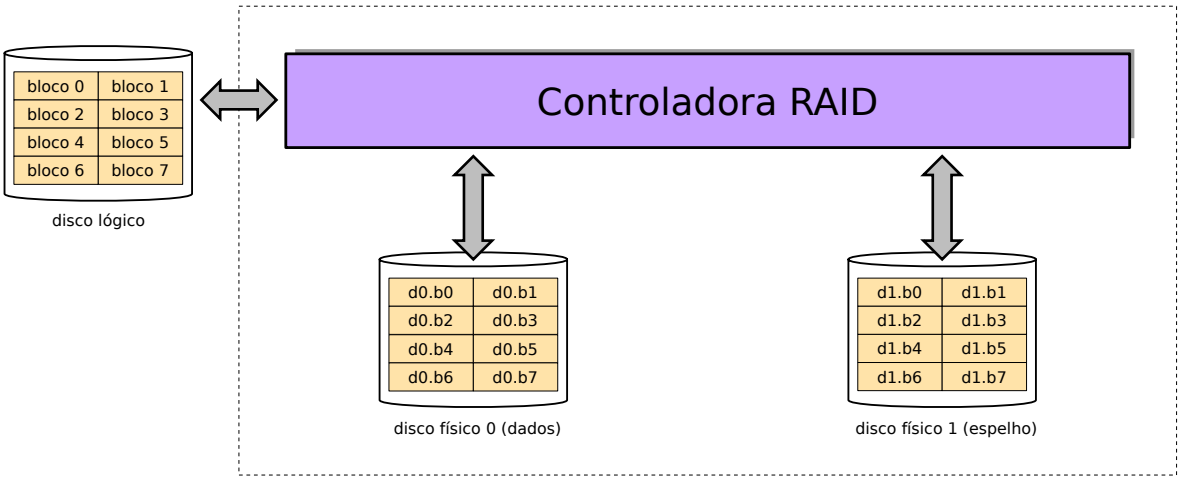

Figura 21.10: RAID nível 1 (espelhamento) com dois discos.

RAID 4: esta abordagem é similar ao RAID 3, com a diferença de que o fatiamento é feito em blocos ao invés de bits, como mostra a Figura 21.11. Ela sofre dos mesmos problemas de desempenho que o RAID 3, sendo por isso pouco usada. Todavia, ela serve como base conceitual para o RAID 5.

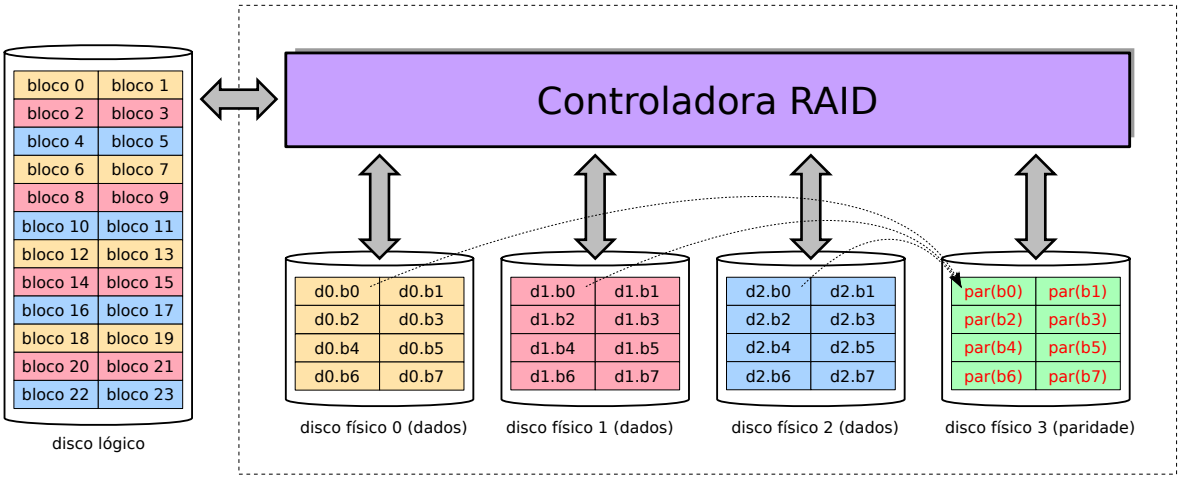

Figura 21.11: RAID nível 4 (disco de paridade).

RAID 5: assim como o RAID 4, esta abordagem também armazena informações de paridade para tolerar falhas em blocos ou discos. Todavia, essas informações não ficam concentradas em um único disco físico, sendo distribuídas uniformemente entre eles. A Figura 21.12 ilustra uma possibilidade de distribuição das informações de paridade. Essa estratégia elimina o gargalo de desempenho no acesso aos dados de paridade visto no RAID 4 (embora seja necessário, a cada escrita, ler blocos de todos os discos para poder calcular o novo bloco de paridade). Esta é uma abordagem de RAID popular, por oferecer um bom desempenho e redundância de dados, desperdiçando menos espaço que o espelhamento (RAID 1).

RAID 6: é uma extensão do nível RAID 5 que utiliza blocos com códigos corretores de erros de *Reed-Solomon*, além dos blocos de paridade. Esta redundância extra

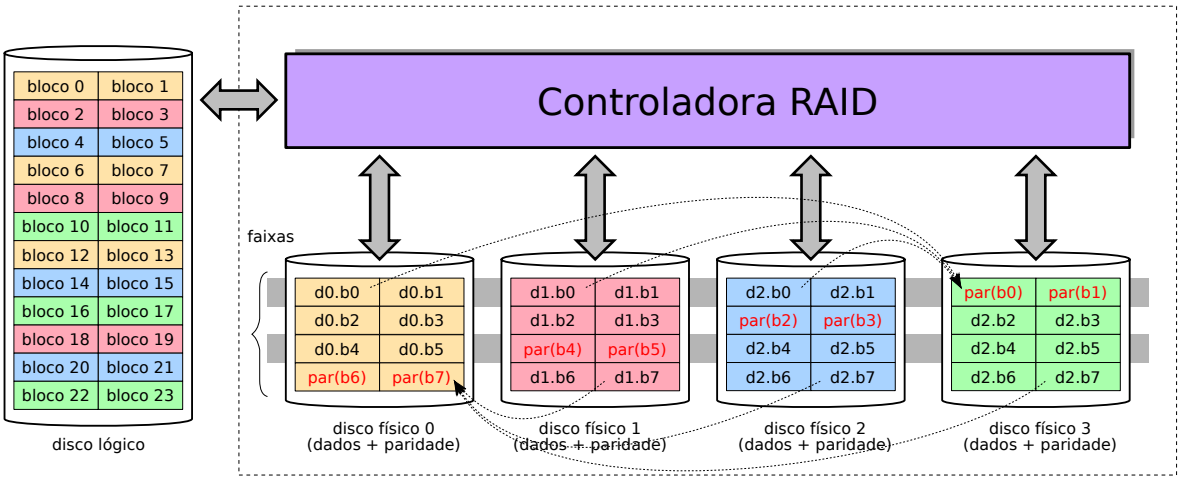

Figura 21.12: RAID nível 5 (paridade distribuída).

demanda dois discos adicionais, mas permite tolerar falhas simultâneas de até dois discos.

Além dos níveis padronizados, no mercado podem ser encontrados produtos oferecendo outros níveis RAID, como 1+0, 0+1, 1E, 50, 100, etc., que muitas vezes implementam combinações dos níveis básicos ou então soluções proprietárias. Outra observação importante é que os vários níveis de RAID não têm necessariamente uma relação hierárquica entre si, ou seja, um sistema RAID 5 não é necessariamente melhor que um sistema RAID 1, pois isso depende do campo de aplicação do sistema. Uma descrição mais aprofundada dos vários níveis RAID, de suas variantes e características pode ser encontrada em [Chen et al., 1994] e [SNIA].

A Tabela 21.1 traz um comparativo entre as principais características das diversas estratégias de RAID apresentadas nesta seção. As velocidades de leitura e de escrita são analisadas em relação às que seriam observadas em um disco isolado de mesmo tipo. A coluna *Espaço* indica a fração do espaço total dos N discos que está disponível para o armazenamento de dados, considerando discos de tamanho T. A coluna *Falhas* indica a quantidade de discos que pode falhar sem causar perda de dados. A coluna *Discos* define o número mínimo de discos necessários para a configuração.

Estratégia	Velocidade leitura	Velocidade escrita	Espaço	Falhas	Discos
RAID 0 linear	até N (acessa discos distintos em paralelo)	até N (acessa discos distintos em paralelo)	$N \times T$	0	≥ 2
RAID 0 striping	até N (idem)	até N (idem)	$N \times T$	0	≥ 2
RAID 1	até N (idem)	1 (requer atualizar todas as cópias)	T	$N-1$	≥ 2
RAID 4	até $N-1$ (acessa discos em paralelo, exceto o de paridade)	1 (requer atualizar disco de paridade)	$(N-1) \times T$	1	≥ 3
RAID 5	até N (acessa discos distintos em paralelo)	1 (lê blocos de todos os discos para calcular e atualizar bloco de paridade)	$(N-1) \times T$	1	≥ 3
RAID 6	até N (idem)	1 (lê blocos de todos os discos para calcular e atualizar blocos de paridade)	$(N-2) \times T$	2	≥ 4

Tabela 21.1: Comparativo de estratégias de RAID, considerando N discos de tamanho T

Exercícios

1. Considere um escalonador de disco com os seguintes pedidos de leitura de blocos em sua fila, nessa ordem: 95, 164, 36, 68, 17 e 115. Determine todos os deslocamentos da cabeça de leitura do disco para atender esses pedidos e o número total de blocos percorridos, para as políticas FCFS, SSTF, SCAN, C-SCAN, LOOK e C-LOOK. O disco tem 200 setores, numerados de 0 a 199, e a cabeça de leitura acabou de atender um pedido para o bloco 50 e está subindo.

2. Você possui 4 discos rígidos de 8 TBytes cada. Para os arranjos desses discos em RAID 0 *Stripping*, RAID 1 e RAID 5, apresente o espaço útil disponível, o número máximo de discos com falha e as velocidades máximas de leitura/escrita em relação a um disco isolado (por exemplo: 1×, 5×, ...).

3. Você tem 4 discos rígidos de 8 TB cada, que pode organizar de diversas formas. Indique os arranjos RAID que escolheria para obter:

   (a) O maior espaço útil de disco.
   (b) A maior tolerância a falhas de disco.
   (c) A maior velocidade média de leitura.
   (d) A maior velocidade média de escrita.
   (e) Equilíbrio entre espaço útil, velocidades e tolerância a falhas.

   Justifique/explique suas respostas.

Atividades

1. Construa um simulador de escalonamento de disco. O simulador deve receber como entrada o tamanho do disco (em blocos) e a sequência de números de

blocos a acessar. Ele deve gerar como saída a sequência de blocos acessados e o deslocamento total da cabeça do disco (em blocos), para os algoritmos apresentados neste capítulo.

Referências

D. Bovet and M. Cesati. *Understanding the Linux Kernel, 3rd edition*. O'Reilly Media, Inc, 2005.

P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26:145–185, June 1994.

R. Love. *Linux Kernel Development, Third Edition*. Addison-Wesley, 2010.

D. Patterson and J. Henessy. *Organização e Projeto de Computadores*. Campus, 2005.

D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *ACM SIGMOD International Conference on Management of Data*, pages 109–116. ACM, 1988.

A. Silberschatz, P. Galvin, and G. Gagne. *Sistemas Operacionais – Conceitos e Aplicações*. Campus, 2001.

SNIA. *Common RAID Disk Data Format Specification*. SNIA – Storage Networking Industry Association, March 2009. Version 2.0 Revision 19.