Proposta de um Ambiente de Simulação e Aprendizado Inteligente para RAID

Daniel Corrêa Lobato

Orientadora: Profa. Dra. Regina Helena C. Santana

*Dissertação apresentada ao Instituto de
Ciências Matemáticas e de Computação da
Universidade de São Paulo (ICMC/USP)
como parte dos requisitos para a obtenção do
título de Mestre em Ciências. Área: Ciências
de Computação e Matemática Computacional.*

USP – São Carlos
Abril, 2000

Este trabalho teve o apoio financeiro do CNPq.

“As coisas não mudam. Mudamos nós.”

Henry David Thoreau

Esta página foi deixada intencionalmente em branco.

Aos meus pais, Tomaz e Denise, e meu irmão Leonardo

À Renatinha

Agradecimentos

Ao “Homem lá de cima” e seus “auxiliares imediatos”, por estarem ao meu lado em todos os momentos, sempre acompanhando, incentivando e protegendo não só a mim, mas todos aqui “embaixo”.

Ao meus pais, pela amizade, educação e incentivo em todos os momentos da minha vida. Obrigado por acreditarem em mim e apoiarem incondicionalmente as minhas decisões. Papai, obrigado por me “contaminar” com um pouco do seu jeito “Zen”. Mamãe, obrigado por ter sempre conversado comigo, todas as vezes em que liguei durante as noites e madrugadas de trabalho.

À Renatinha, pelo amor e carinho, pela ajuda nas simulações e revisões (não se preocupe: não encontraremos mais erros de digitação nem letras trocadas!), e por ter estar sempre alerta para dar um “empurrão” quando as coisas estagnavam e/ou “travavam”.

À profa. Regina Santana, que sempre esteve disponível para sanar todas as minhas dúvidas, inclusive as mais básicas, além de ter incentivado e acreditado no trabalho desde o início. Obrigado por tentar me manter sempre com os pés no chão, lembrando-me que o Mestrado dura apenas dois anos.

Ao prof. Marcos Santana, por estar, também, disponível e acessível para discutir qualquer assunto. Obrigado pela possibilidade de “experimentar” a vida acadêmica através das chances de monitoria no Programa PAE.

À Sarita, minha “consultora” em C/C++, SMPL e SSS, por ter me ajudado com o primeiro protótipo do simulador de discos e ter encontrado os ponteiros do C que eu perdi no meio da primeira versão do código.

Ao pessoal do LaSDPC – sem nenhuma ordem especial: Adriana, Aletéia, Álvaro, Andreza, Ana Elisa, Arion, Boca (João Carlos), Célia, Edmilson, Fernanda, Flávio, Helder, Jorge, João Carlos, Kalinka Regina, Mário, Mara, Márcio Augusto, Omar, Paulo Sérgio, Carlos Renato, Renata, Roberta, Sarita, Tatiana, Tomas e Vera – pelo companheirismo e bons momentos que passamos nestes dois anos. Obrigado por me ouvirem falar de RAID tantas vezes nos seminários do Grupo e, mesmo assim, não caírem em sono profundo. Prometo que não apago mais as áreas de vocês no Servidor.

Ao pessoal da biblioteca – Gláucia, Giselda, Gislene, Juliana, Maria, Regina, Rosimari, Rosimeire e Sandra – por me ajudar a localizar os artigos que precisei, mesmo tendo referências incompletas e, as vezes, apenas um nome de autor.

Ao companheiros de apartamento Jorge, Andrezza, Alessandro e Dennys, pelo companheirismo durante esses dois anos.

À todos os que de alguma forma colaboraram durante esses dois anos, através de comentários, sugestões e críticas.

Resumo

O desempenho global dos sistemas computacionais é limitado, geralmente, pelo componente de menor desempenho. Os processadores e a memória principal têm experimentado um aumento de desempenho bem maior que o da memória secundária, como os discos magnéticos. Em 1984, Johnson introduziu o conceito de fragmentação, onde um dado é gravado em uma matriz de discos, de forma que os seus fragmentos podem ser recuperados em paralelo e, por consequência, de forma mais rápida. O principal problema da fragmentação é a redução da confiabilidade da matriz pois, a falha de um dos discos torna o dado inacessível. Patterson, Gibson e Katz propuseram, em 1988, 5 formas de armazenar informação redundante na matriz de discos e, dessa forma, aumentar sua confiabilidade. A essas formas foi dado o nome de RAID – *Redundant Arrays of Independent Disks*. Com o passar do tempo, outras formas de armazenamento de redundância foram criadas, tornando complexa a taxonomia da área. Além disso, alterações de parâmetros na matriz implicam em variações de desempenho nem sempre fáceis de se perceber em um primeiro momento. Com o objetivo de facilitar a compreensão da taxonomia e permitir que sejam feitos experimentos na matriz buscando um melhor desempenho, esta dissertação propõe um ambiente de simulação e aprendizado para RAID, onde o usuário pode interagir com diversos modelos de RAID, ou até criar o seu próprio, para avaliar seu desempenho em várias situações, além de oferecer ao usuário acesso ao conhecimento da área, agindo como um tutor. Esta dissertação apresenta, ainda, um protótipo de um simulador de discos magnéticos que pode ser utilizado como base para o desenvolvimento de um simulador de RAID para ser utilizado pelo ambiente.

Abstract

The component with the worst performance usually limits the overall performance of a computing system. The performance of processors and main memory has improved faster than the secondary memory, such as magnetic disks. Johnson, in 1984, introduced the concept of fragmentation, in which a data file is written into a disk array, in a way that its stripes can be recovered in parallel and therefore, in a faster way. The main problem with fragmentation is the reduction of the reliability. If one disk fails, all data file becomes inaccessible. Patterson, Gibson and Katz proposed, in 1988, five ways to store redundant information in the array, increasing the reliability, comprising the main RAID (Redundant Array of Independent Disks) configurations. Some other ways to store the redundant information have been proposed over the years, making the RAID taxonomy more complex. Furthermore, changes in the array parameters takes to performance variations that are not always understood. With the purpose of facilitating the comprehension of the taxonomy and allowing the execution of experiments looking forward to improve performance, this MSc Dissertation proposes an Intelligent Simulation and Learning Environment for RAID, where the user can interact with several RAID models, or even create his/her own models, in order to evaluate their performance under different situations. The environment also allows the user to interact with the knowledge of the area, acting as a tutor. This Dissertation also presents a prototype of a magnetic disk simulator, that can be used as the kernel for the development of a RAID simulator to be used by the environment.

SUMÁRIO

Agradecimentos	vi
Resumo	viii
<i>Abstract</i>	ix
Lista de figuras	xii
Lista de tabelas	xiii
Lista de equações.....	xiv
Lista de siglas	xv
Glossário de termos	xvi
Capítulo 1 Introdução	1
Capítulo 2 Discos Magnéticos e Interfaces.....	5
2.1 Discos Magnéticos	6
2.2 Barramentos	15
2.3 Interface IDE.....	16
2.4 Interface SCSI.....	19
2.5 Comparações entre IDE e SCSI	23
Capítulo 3 RAID – Redundant Array of Independent Disks	27
3.1 Níveis RAID.....	30
3.1.1 RAIDs originais	31
3.1.2 RAIDs derivados.....	35
3.2 Ortogonalidade.....	38
3.3 Comparação de desempenho e custo.....	39
3.3.1 Observações sobre a avaliação.....	40
3.3.2 Comparações.....	40
3.4 Aspectos de desempenho do RAID 5 (Chen et al., 1994).....	43
3.4.1 Buffering e Cache	43
3.4.2 Floating Parity.....	44
3.4.3 Parity Logging	45
3.5 Tamanho da unidade de fragmentação.....	45
3.6 Comparação de confiabilidade	47
3.6.1 Confiabilidade básica.....	47
3.6.2 Falha de sistema e inconsistência de paridade	48
3.6.3 Erros de bit incorrigíveis.....	49
3.6.4 Falhas de disco correlatas	49
3.6.5 Confiabilidade revista	50
3.7 Considerações finais.....	52
Capítulo 4 Simulação.....	53
4.1 Classificação dos modelos	54
4.2 Simulação Discreta.....	55
4.3 Ferramentas para simulação	56
4.4 SMPL x SSS.....	58
4.5 Considerações finais.....	60
Capítulo 5 Ambientes de Aprendizado por Simulação.....	61
5.1 Intelligent Simulation and Learning Environment.....	62
5.2 Considerações sobre interfaces com o usuário.....	64
5.3 Considerações finais.....	67
Capítulo 6 Especificação do Ambiente para Simulação e Aprendizado de RAID	69
6.1 Visão Geral.....	70
6.1.1 Meta-módulo de inicialização do Ambiente	72
6.1.2 Meta-módulo de transmissão do conhecimento.....	74
6.1.3 Módulo de avaliação de conhecimento.....	77
6.2 Detalhamento de Depósitos.....	78

6.2.1	Discos, Interfaces e Regras (D1, D2 e D3)	78
6.2.2	Arranjos RAID do Ambiente e do Estudante (D4 e D12)	81
6.2.3	Conhecimento Bibliográfico e de Simulação (D5 e D6)	81
6.2.4	Aulas (D7).....	82
6.2.5	Estudantes (D8).....	83
6.2.6	Pontos de Parada (D9)	83
6.2.7	Avaliação de Estudantes e Testes (D11 e D10).....	84
6.3	Considerações finais.....	84
Capítulo 7	Simuladores de discos e RAID	87
7.1	RAIDframe – A Prototyping Tool for RAID Systems.....	87
7.1.1	Representação de RAIDs	88
7.1.2	Modelagem de discos magnéticos.....	89
7.1.3	Aspectos do simulador.....	90
7.2	DiskSim – The DiskSim Simulation Environment	93
7.2.1	Representação de RAIDs	94
7.2.2	Modelagem de discos magnéticos.....	95
7.2.3	Aspectos do simulador.....	96
7.3	Aplicabilidade ao Ambiente proposto.....	98
Capítulo 8	Protótipo de simulador de discos magnéticos	101
8.1	Modelagem e implementação.....	102
8.2	Testes efetuados e resultados obtidos.....	104
8.3	Limitações do protótipo	108
8.4	Considerações finais.....	109
Capítulo 9	Observações Finais e Conclusões	111
9.1	Considerações sobre a revisão bibliográfica	111
9.2	Conclusões e contribuições	111
9.3	Sugestões de trabalhos futuros	114
Bibliografia		115
Anexo A	Erros na utilização de interfaces	120
Anexo B	Códigos fonte e configuração do protótipo de simulador de discos magnéticos.....	122
Anexo C	Arquivos de configuração e resultados do DiskSim.....	138
Apêndice A		147

Lista de figuras

Figura 2.1 Estrutura de uma unidade de discos magnéticos	7
Figura 2.2 Visão das camadas que recobrem o substrato	7
Figura 2.3 Tempo de busca em função da distância em cilindros segundo Lee et al., 1991b	10
Figura 2.4 Mapeamento vertical	13
Figura 2.5 Mapeamento horizontal.....	14
Figura 2.6 Mapeamento horizontal e vertical	14
Figura 2.7 Transição de fases do protocolo SCSI para operação de leitura.....	22
Figura 3.1 Exemplos de speed up com a Lei de Amdahl.....	28
Figura 3.2 RAID 1	31
Figura 3.3 RAID 2	32
Figura 3.4 RAID 3	33
Figura 3.5 RAID 4	33
Figura 3.6 RAID 5	34
Figura 3.7 RAID 0	35
Figura 3.8 RAID 6	36
Figura 3.9 RAID 7	36
Figura 3.10 RAID 10	37
Figura 3.11 RAID 53	38
Figura 3.12 RAID Ortogonal.....	39
Figura 3.13 <i>Throughput</i> máximo por dólar <i>versus</i> tamanho do grupo	41
Figura 3.14 Comparação de desempenho entre RAIDs 3, 5 e 6	42
Figura 4.1 Processo de abstração e inferência	53
Figura 4.2 Relação entre evento, atividade e processo	55
Figura 4.3 Estrutura de programa de simulação em SMPL e em SSS.....	59
Figura 5.1 Estrutura de um ISLE	63
Figura 5.2 Relação entre modelo de projeto, de usuário e do sistema.....	67
Figura 6.1 Visão geral do Ambiente através de um diagrama de blocos.....	70
Figura 6.2 Diagrama de blocos do meta-módulo de inicialização do Ambiente	72
Figura 6.3 Diagrama de blocos do meta-módulo de transmissão do conhecimento.....	75
Figura 6.4 Diagrama de blocos do meta-módulo de avaliação de conhecimento.....	77
Figura 7.1 DAG de pequena escrita bem sucedida em RAID 6	88
Figura 7.2 Configuração de disco no RAIDframe	90
Figura 7.3 Exemplo de carga de trabalho no RAIDframe	91
Figura 7.4 Exemplo de arquivo de configuração de simulação no RAIDframe	91
Figura 7.5 Exemplo de arquivo de configuração de matriz no RAIDframe	92
Figura 7.6 Exemplo de arquivo de configuração de cargas de trabalho no RAIDframe	93
Figura 7.7 Exemplo de arquivo de saída no RAIDframe.....	93
Figura 7.8 Exemplo de organização física dos elementos no DiskSim	95
Figura 7.9 <i>Throughput</i> máximo por dólar <i>versus</i> tamanho do grupo	97
Figura 8.1 Camadas de um simulador de RAID	101
Figura 8.2 Modelo de rede de filas para um disco magnético	102
Figura 8.3 Curvas de tempo de busca para o Disco 1 variando-se o cálculo do tempo médio.....	105
Figura 8.4 Curvas de tempo de busca para o Disco 2 variando-se o cálculo do tempo médio.....	107

Lista de tabelas

Tabela 2.1 Modos PIO	17
Tabela 2.2 Modos DMA	17
Tabela 2.3 Comparação de padrões IDE/ATA	19
Tabela 2.4 Sumário dos protocolos e padrões SCSI disponíveis.....	23
Tabela 3.1 <i>Throughput</i> máximo por dólar em relação ao RAID 0	40
Tabela 3.2 Sobrecarga de armazenamento e grau máximo de concorrência	41
Tabela 3.3 Parâmetros de confiabilidade para cálculos estimativos.....	50
Tabela 3.4 Métricas de confiabilidade para RAID 5	51
Tabela 3.5 Métricas de confiabilidade para RAID 6	51
Tabela 5.1 Paradigmas de interação homem-máquina	65
Tabela 7.1 Comparação entre dos simuladores RAIDframe e DiskSim.....	99
Tabela 8.1 Tempo médio de busca para Disco 1	104
Tabela 8.2 Constantes de Lee variando-se o tempo médio de busca do Disco 1.....	105
Tabela 8.3 Resultados de simulação do Disco 1.....	106
Tabela 8.4 Tempo médio de busca para Disco 2	106
Tabela 8.5 Constantes de Lee variando-se o tempo médio de busca do Disco 2.....	107
Tabela 8.6 Resultados de simulação do Disco 2.....	108

Lista de Equações

Equação 2.1 Tempo de busca por Lee et al., 1991	10
Equação 2.2 Tempo médio de busca	10
Equação 2.3 Tempo médio de busca ponderado.....	11
Equação 2.4 Tempo de busca por Ruemmler et al.....	11
Equação 3.1 Lei de Amdahl.....	27
Equação 3.2 Tempo médio para falha de matriz de disco	28
Equação 3.3 Unidade de fragmentação (Chen & Lee)	45
Equação 3.4 Unidade de fragmentação (Chen & Patterson - 1)	46
Equação 3.5 Fator de desvio	46
Equação 3.6 Unidade de fragmentação (Chen & Patterson - 2)	46
Equação 3.7 Unidade de fragmentação (Chen & Patterson - 3)	46
Equação 3.8 Tempo médio entre falhas para RAID 5	47
Equação 3.9 Tempo médio entre falhas para RAID 6	48

Lista de siglas

<i>Sigla</i>	<i>Significado</i>
ATA	<i>AT Attachment</i>
ATA-PI	<i>ATA Packet Interface</i>
BER	<i>Bit Error Rate</i> Taxa de Erro de Bits
DMA	<i>Direct Memory Access</i> Acesso direto à memória
EIDE	<i>Enhanced IDE</i>
ES/s	Entradas e saídas por segundo
GUI	<i>Graphical User Interface</i> Interface Gráfica com o Usuário
IDE	<i>Integrated Drive Electronics, Integrated Device Electronics</i>
ISLE	<i>Intelligent Simulation Learning Environment</i> Ambiente Inteligente de Simulação e Aprendizado
LUN	<i>Logical unit number</i>
MTBF	<i>Mean Time Between Failures</i> Tempo Médio entre Falhas
MTTF	<i>Mean Time to Failure</i> Tempo Médio para Falha
MTTR	<i>Mean Time to Repair</i> Tempo Médio para Reparo
PIO	<i>Programmed Input/Output</i> Entrada/Saída programada
RAID	<i>Redundant Array of Independent Disks</i> Matriz Redundante de Discos Independentes
RISC	<i>Reduced Instruction Set Computer</i>
RMW	<i>Read-modify-write operation</i> Operação de leitura-modificação-escrita
SCSI	<i>Small Computer System Interface</i>
SIMM	<i>Single In-Line Memory Module</i>
SNMP	<i>Simple Network Management Protocol</i> Protocolo simples para gerenciamento de rede
TPI	<i>Tracks per Inch</i> Trilhas por polegada
WINP	<i>Windows, icons, menus and pointing devices</i> Janelas, ícones, menus e dispositivos apontadores

Glossário de termos

<i>Termo</i>	<i>Significado</i>
<i>avSeek</i>	Tempo médio de busca.
<i>avSettle</i>	Tempo médio de posicionamento da cabeça de leitura/gravação.
<i>Buffer</i>	Uma área de memória intermediária entre dois componentes para armazenar mensagens trocadas entre eles. Ao contrário do cache, não explora localidade temporal ou espacial dos dados.
<i>Bus mastering</i>	Recurso suportado por algumas arquiteturas de barramento que permite que uma controladora se comunique diretamente com outro dispositivo sem passar pela UCP. As arquiteturas de barramento mais modernas, como a PCI, dispõem desse recurso.
<i>Cache</i>	Memória pequena e rápida, que armazena dados acessados recentemente (localidade temporal) ou dados próximos a eles (localidade espacial). Geralmente usado entre processador e memória, mas o conceito pode ser aplicado a outros elementos.
<i>Cycle time</i>	Período de tempo decorrido entre duas operações sucessivas de acesso à memória eletrônica, sejam de escrita ou de leitura. O tempo de ciclo é, nas memórias do tipo DRAM, maior que o tempo de acesso declarado da memória (Monteiro, 1995).
<i>Cylinder skewing</i>	Deslocamento, em setores, da posição do setor lógico 0 da primeira trilha de um cilindro em relação à última trilha do cilindro anterior.
<i>Full stroke</i>	Movimento do braço do disco magnético, deslocando-se do primeiro para o último cilindro.
<i>Head positioning time</i>	Tempo de posicionamento da cabeça de leitura/gravação, composto pelo tempo de busca acrescido do atraso rotacional.
<i>Hot spare</i>	Componentes que aceitam a substituição com o sistema “no ar”, sem necessidade de reiniciar o mesmo.
<i>maSeek</i>	Tempo máximo de busca. Geralmente coincidente com o tempo para efetuar um <i>full stroke</i> .
<i>miSeek</i>	Tempo de troca de trilha.
<i>numCyl</i>	Número de cilindros no disco magnético.
<i>Platters</i>	Pratos componentes do substrato de um disco magnético.
<i>Programmed Input/Output</i>	Método para transferir dados entre dois dispositivos que usa o processador central como parte do caminho de dados.
<i>Read-modify-write</i>	Operação de acesso ao dados onde eles são lidos, modificados e re-escritos.
<i>Rotational delay</i>	Tempo a esperar até que, uma vez no cilindro correto, o setor correto passe sob a cabeça de leitura/gravação.
<i>Seek time</i>	Tempo de busca por um determinado cilindro em um disco magnético.
<i>seSize</i>	Tamanho de um setor do disco.
<i>Speed up</i>	Razão entre o tempo de execução de uma tarefa em um processador e a execução em múltiplos processadores.
<i>Spin down</i>	Processo de redução da velocidade rotacional, podendo chegar ao repouso.
<i>Spin up</i>	Processo de aumento da velocidade rotacional, do repouso até que ela atinja seu valor nominal.

Glossário de termos

continuação

<i>Termo</i>	<i>Significado</i>
<i>Think time</i>	Em sistemas interativos, o tempo entre a chegada da resposta de uma requisição e a emissão da próxima requisição.
<i>(arquivo de) Trace</i>	Arquivo que contém informações sobre requisições efetuadas a uma unidade de discos. Armazena, normalmente, o instante do acesso, se a operação foi de leitura ou gravação e qual posição do disco foi acessada.
<i>Track skewing</i>	Deslocamento, em setores, da posição do setor lógico 0 de uma trilha em relação à trilha anterior.

Esta página foi deixada intencionalmente em branco.

Capítulo 1

Introdução

A demanda crescente de processamento tem motivado a evolução dos computadores, viabilizando a implementação de aplicações que envolvem um intenso processamento e grandes volumes de dados. Para atingir um alto desempenho, é necessário que tanto o processamento quanto a obtenção de dados sejam rápidos. A velocidade de acesso aos dados pode ser dividida entre o tempo para obter dados que estão armazenados na memória principal e o tempo para acessar os dados na memória secundária.

Considerando-se estes três componentes (processador, acesso à memória principal e acesso à memória secundária), observa-se que os processadores têm apresentado uma evolução espantosa. Alguns dos motivos que podem ser citados para essa evolução são (Sato et al., 1996):

- ## Aumento da velocidade do relógio do sistema;
- ## Desenvolvimento de processadores RISC, vetoriais e super-escalares;
- ## Utilização de *pipelines* de instrução e aritméticos;

O tempo de acesso à memória principal não tem acompanhado a evolução acelerada no desempenho dos processadores, mas também apresenta uma melhora considerável. Dentre as razões para a queda no tempo de acesso à memória principal pode-se observar:

- ## Evolução tecnológica;
- ## Utilização de memórias *cache*;
- ## Utilização de *interleaved memory*;

O problema é que a velocidade de obtenção dos dados armazenados em memória secundária, ou seja, dos sub sistemas de entrada e saída de dados, é dependente de evoluções mecânicas nos discos magnéticos utilizados. A evolução observada nos dispositivos de armazenamento secundário, apesar de expressivos, tem ficado bastante aquém dos processadores e das memórias principais, principalmente devido a limites mecânicos.

De forma a superar as limitações mecânicas, uma alternativa proposta em 1984 foi a fragmentação dos dados no momento do armazenamento nos discos (Johnson⁸⁴ *apud* Chen et al., 1990). Nestes casos, os discos são arranjados em matrizes ou vetores. A fragmentação melhora o desempenho de E/S por permitir que múltiplas E/S sejam executadas em paralelo. Com a

fragmentação duas situações podem ocorrer: requisições múltiplas e independentes podem ser atendidas em paralelo por discos distintos e uma requisição por um dado que esteja fragmentado em vários discos é atendida por múltiplos discos atuando em coordenação.

Da primeira situação obtêm-se a redução do tempo de espera das solicitações de E/S, uma vez que as solicitações são “divididas” entre os vários discos da matriz. A segunda situação oferece um aumento da taxa de transferência efetiva percebida por uma requisição, já que cada disco é responsável por recuperar um ou mais fragmentos do dado, e essa recuperação pode ser executada em paralelo.

Quanto mais discos na matriz, maiores os benefícios potenciais de desempenho. Infelizmente, um grande número de discos reduz a confiabilidade geral da matriz de discos para um valor abaixo do aceitável, uma vez que para que o dado seja recuperado com sucesso, é necessário que todos os fragmentos dele sejam recuperados com sucesso. Para se obter um aumento da confiabilidade dessas matrizes de disco torna-se necessário o armazenamento de informações redundantes, de forma que, na falha de um ou mais dos discos componentes da matriz, o fragmento de dado armazenado no disco em questão possa ser recuperado de alguma forma.

Em 1988, Patterson et al. (Patterson et al., 1988), propuseram uma forma de organização da matriz de discos e das informações redundantes, à qual foi dado o nome de **Matriz Redundante de Discos Independentes (RAID)**. No trabalho original de Patterson et al. foram propostas cinco formas de organização dos dados e das informações redundantes (níveis RAID), mas das formas originais surgiram outras tantas, algumas novas, outras que não passam de combinações das cinco formas originais. Estudos adicionais (Chen et al., 1990; Chervenak, 1990; Lee et al., 1991a; Lee et al., 1991b; Weikum, 1992; Stodolsky et al., 1993; Chen et al., 1994; Lee et al., 1996; Alvarez et al., 1998) mostram ou propõem características de desempenho e confiabilidade para situações específicas.

RAID encontra aplicação nos mais diversos segmentos de sistemas computacionais, como computadores executando simulações atômicas (Garg et al., 1998; Mattson et al., 1998), servidores de imagens de satélite (Microsoft, 1998) e supercomputadores convencionais (LoVerso93 *apud* Chen et al., 1994), além de qualquer outro segmento que necessite alto desempenho, disponibilidade e confiabilidade dos dados.

Os diferentes níveis RAID propostos apresentam características lógicas distintas, isto é, que informação será armazenada em qual disco da matriz, como será calculada e armazenada a redundância, a organização da matriz RAID em termos de controladoras e discos, etc. Estes diferentes níveis conduzem a uma taxonomia vasta e características de desempenho e confiabilidade nem sempre compreendidas em um primeiro contato.

O objetivo desta dissertação é a especificação de um ambiente que auxilie no aprendizado e na avaliação de desempenho de diferentes arranjos RAID. O Ambiente de Simulação e Aprendizado Inteligente para RAID deve oferecer para o usuário um ambiente amigável onde seja possível a visualização e avaliação dos diferentes níveis RAID.

Para a especificação deste ambiente foi feita uma revisão bibliográfica crítica sobre discos magnéticos e principais interfaces utilizadas na conexão dos mesmos aos sistemas computacionais, e o resultado desta está apresentado no Capítulo 2. Uma revisão bibliográfica sobre RAID e os aspectos de desempenho e confiabilidade de seus principais níveis foi também efetuada, sendo esta apresentada no Capítulo 3.

Uma vez que o ambiente se propõe a simular tanto para avaliar como para auxiliar o aprendizado, no Capítulo 4 são apresentados os principais aspectos de simulação e modelagem relevantes para o problema aqui abordado. Os ambientes de simulação e aprendizado inteligente são abordados no Capítulo 5, juntamente com aspectos importantes do projeto de interfaces gráficas.

O Capítulo 6 apresenta o Ambiente proposto e suas principais características. No Capítulo 7 são apresentados aspectos relativos aos simuladores estudados e sua aplicabilidade ao Ambiente. No Capítulo 8 é feito um estudo sobre o desenvolvimento de um simulador de discos próprio que pode ser expandido e estendido para simular RAID e vir a se tornar o simulador padrão do ambiente.

No Capítulo 9 são apresentadas as conclusões e contribuições do ambiente aqui proposto, bem como sugestões de trabalhos futuros. As referências bibliográficas e os anexos são apresentados por fim.

Esta página foi deixada intencionalmente em branco

Capítulo 2

Discos Magnéticos e Interfaces

A primeira unidade de disco (RAMAC 305) surgiu em 1957, desenvolvida pela IBM. Utilizava 50 discos, de 24 polegadas de diâmetro, e era capaz de armazenar 5Mb de informações. Seu custo era de US\$ 35.000 por ano, na opção de leasing (US\$ 7.000/Mb). No início dos anos 80, foi lançada a primeira unidade de disco com 5¼" de diâmetro, com a capacidade de 10Mb. Em 1987, surgiram as primeiras unidades de 3½" e, depois, as de 2½". Em 1992, surgiu a primeira unidade de disco com 1,8" e, por fim, as unidades com 1,3". A capacidade típica das unidades de disco de 3½" hoje está em torno de 17Gb, custando pouco mais de US\$180,00 (US\$ 0,01/Mb ou US\$10,58/Gb).

Nos sistemas computacionais, os primeiros dispositivos de armazenamento em massa foram os cartões perfurados, usados desde 1804 em teares automáticos. Os dispositivos de armazenamento em massa atuais incluem todos os tipos de unidades de disco e de fita. Ao contrário da memória principal dos sistemas computacionais, que é volátil, os dispositivos de armazenamento em massa mantêm os dados mesmo quando deixam de receber corrente elétrica.

Tipos de armazenamento em massa mais utilizados atualmente são:

Discos flexíveis (floppy disks): Relativamente lentos e possuem baixa capacidade (hoje, até 2,88Mb), mas são portáteis, baratos e universais;

Discos rígidos: Rápidos e com capacidade bem maior que a dos discos flexíveis. Alguns são portáteis (cartuchos removíveis), mas a maior parte não o é;

Discos removíveis: Estes discos possuem capacidades inferiores às dos discos rígidos e superiores às dos discos flexíveis, oscilando entre 100Mb e 1Gb. Os tempos de acesso são, também, intermediários. Os mais comuns são o ZIP Disk, JAZ Disk, Super Disk e Bernoulli.

Discos ópticos: Ao contrário dos discos flexíveis e rígidos, que usam eletromagnetismo para codificar os dados, os sistemas de disco ópticos utilizam *laser* para ler e escrever dados. Os discos ópticos possuem alta capacidade, mas são mais lentos e mais caros que os discos rígidos;

Fitas magnéticas: Relativamente baratas e com grande capacidade de armazenamento. Não permitem acesso aleatório aos dados.

Durante muitos anos, unidades de discos magnéticos (discos rígidos) ficaram confinados a *mainframes* e mini computadores. Com a revolução dos computadores pessoais no início dos anos 80,

os primeiros discos rígidos para computadores pessoais foram introduzidos, com capacidades entre 5 e 10Mb.

Os primeiros PCs usavam, na sua grande maioria, apenas discos flexíveis para armazenamento. O termo “flexível” foi perfeitamente adequado para os discos flexíveis de 8 e de 5¼”. O disco ficava dentro de um envelope plástico, que podia ser curvado facilmente. Os discos flexíveis de 3½” ficam dentro de uma caixa plástica mais rígida, durável e resistente.

Com a chegada dos IBM PC/XT em 1983, os discos rígidos se tornaram um componente padrão na maior parte dos computadores pessoais.

O estado-da-arte em armazenamento envolve não apenas armazenamento magnético, mas também armazenamento óptico e até holografia. A Lucent (Lucent, 1999) está trabalhando no desenvolvimento de discos holográficos do tamanho dos CD-ROMs de hoje. Nesses discos holográficos, a informação é armazenada não apenas na superfície, mas ao longo de todo meio transparente que recobre a superfície, conseguindo armazenar até 125 Gbytes. Outras tecnologias ainda estão nos estágios iniciais de desenvolvimento, como a tecnologia de armazenamento fluorescente multi camada, proposta pela C3D (C3D, 1999). Nessa tecnologia, ao invés de uma luz coerente (laser) é utilizada luz incoerente (difusa) para armazenar os dados em até dez camadas de um disco transparente. Com essa tecnologia, a capacidade de armazenamento pode chegar a 140 Gbytes, com uma taxa de transferência de até 1 Gbyte por segundo.

A impressão que se tem, ao saber dessas novas tecnologias é que armazenamento magnéticos estaria ficando obsoleto. Na realidade, ele continua sendo usado e aprimorado. Em junho de 1999, a densidade de armazenamento magnético chegava, no máximo, a 20 bilhões de bits por polegada quadrada. Em novembro do mesmo ano, a IBM anunciou ter conseguido uma densidade de armazenamento de 35,3 bilhões de bits por polegada quadrada. Com essa densidade é possível ter cerca de 512 Gbytes em uma unidade comum de discos magnéticos de 3½” (Harrow, 1999).

2.1 Discos Magnéticos

A Figura 2.1 (adaptada de Chen et al., 1994 e Ruemmler et al., 1994) mostra a estrutura básica de uma unidade de discos magnéticos. Um disco consiste, basicamente, de um conjunto de **pratos** (*platters*) e um conjunto de **braços** com **cabeças magnéticas** de leitura/escrita que se movem radialmente sobre as superfícies dos pratos através da ação de um **motor pivô**. Os pratos são discos de um substrato – alumínio, vidro ou cerâmica – cobertos por uma mídia magnética girando com uma velocidade angular constante. O conjunto é deslocado por um servo-motor (Chen et al., 1994). Em agosto de 1999, a Sony Corporation iniciou as pesquisas para utilizar plástico como substrato, com o

objetivo de reduzir o custo de fabricação de unidades de discos magnéticos em aproximadamente 35% (Harrow, 1999).

Figura 2.1 Estrutura de uma unidade de discos magnéticos

A mídia magnética que recobre o substrato é composta de vários níveis, apresentados na Figura 2.2 e listados a seguir:

1. Composto de níquel e fósforo;
2. Cromo;
3. Camada magnética que armazena a informação. Geralmente um composto de cobalto (CoCrTa, CoPtCr, CoPtCrTa ou outro);
4. Camada de carbono, para proteger contra atrito e corrosão eventualmente causados pelas cabeças de leitura/gravação.

Geralmente sobre a última camada é aplicado algum lubrificante, para reduzir uma possível fricção entre os pratos e as cabeças (Scherer, 1999).

Figura 2.2 Visão das camadas que recobrem o substrato

Quando as cabeças estão corretamente posicionadas, os dados são lidos/escritos em pequenos arcos chamados **setores** na superfície dos pratos durante a rotação relativa dos mesmos em relação às cabeças. Apesar das cabeças se moverem coletivamente, em quase todas as unidades de discos magnéticos apenas uma cabeça pode ler/escrever de cada vez. Um conjunto de setores do mesmo prato com a mesma distância em relação ao centro é chamado **trilha**, e a superfície de um prato é composta

por um conjunto de anéis concêntricos de trilhas. Um conjunto vertical de trilhas na mesma posição radial é chamado de **cilindro**.

Considerando os elementos descritos nesta seção, os tempos de serviço em discos magnéticos podem ser divididos em três componentes primários (Chen et al., 1994): **tempo de busca** (*seek time*), **atraso rotacional** (*rotational delay*) e **tempo de transferência de dados** (*data transfer time*). O tempo de busca e o atraso rotacional são algumas vezes chamados, coletivamente, de tempo de posicionamento de cabeça (*head positioning time*).

O **atraso rotacional** está relacionado com o tempo necessário para que, uma vez no cilindro correto, o setor correto passe sob as cabeças de leitura. O atraso rotacional médio é menor, quanto maior for a velocidade angular da unidade de disco, medida em rotações por minutos. Os discos mais modernos possuem uma velocidade angular entre 7.200 e 10.000 rpm. Quanto maior a velocidade angular, maior o ruído e o calor gerado pela unidade de disco.

O **tempo de busca** é o tempo necessário para deslocar as cabeças para a posição radial (cilindro) correta, e varia de 0,8 (passar para a próxima trilha) a 17ms (*full stroke*, ou seja, passar da primeira para a última trilha), nos discos atuais, dependendo da posição anterior das cabeças e do disco em questão. O tempo médio de busca típico é inferior a 10ms nos disco atuais.

O tempo de posicionamento de cabeça é limitado pela potência disponível para o motor pivô que desloca os braços e pela robustez dos mesmos. Para reduzir o tempo de busca pela metade é necessário quadruplicar a potência aplicada. Se o braço não for suficientemente robusto, ele pode vergar com a torção aplicada pelo pivô e atingir a superfície dos pratos. O tempo de busca é composto de quatro fases:

##Aceleração: os braços são acelerados até a metade da distância ou uma velocidade máxima definida pelo fabricante;

##Coast: em busca longas, trajeto no qual o braço se desloca na sua velocidade máxima;

##Redução: onde os braços são colocados próximo da trilha desejada;

##Assentamento: a controladora do disco posiciona a cabeça na trilha correta.

Buscas muito pequenas (até 4 cilindros) são dominadas pelo tempo de assentamento (de 1 a 3ms) – na realidade pode nem haver busca, pois as cabeças são apenas reassentadas. As buscas pequenas (até 400 cilindros) são dominadas pelo tempo gasto na fase de aceleração, e o tempo pode ser considerado proporcional à raiz quadrada da distância de busca mais o tempo de assentamento. Já nas buscas mais longas (acima de 400 cilindros), a fase de *coast* domina o tempo de busca, que se torna proporcional à distância acrescido de uma constante.

Considerando esses aspectos, pode-se afirmar que, quanto menos cilindros o disco possuir, mais dominado pelo tempo de assentamento o tempo de busca é.

O ajuste fino de posicionamento na fase de assentamento é responsabilidade do **sistema de acompanhamento de trilha**. É função dele, também, manter a cabeça sobre a trilha desejada. As informações para permitir a localização e manutenção estão armazenadas, ou em trilhas, ou numa superfície dedicada. Com o aumento da densidade de trilhas por polegada (TPI), as trilhas dedicadas se tornam mais importantes para o posicionamento fino, e uma superfície adicional pode ser utilizada, como auxílio, para um posicionamento preliminar. Entretanto, em ambientes onde o disco pode sofrer choques, o posicionamento apenas por trilhas dedicadas tende a não funcionar muito bem, uma vez que as informações estão disponíveis, apenas, de forma intermitente entre os setores de dados.

O sistema de acompanhamento de trilha é usado, também, para fazer a troca de superfície. Quando a controladora troca o canal de dados de uma superfície para outra no mesmo cilindro, a nova cabeça precisa ser reposicionada para acomodar pequenas diferenças no alinhamento de trilhas nas diferentes superfícies. O tempo dessa troca (de 0,5 a 1,5ms) é, geralmente, 33,3 a 50% do tempo de assentamento. De forma similar, quando ocorre um troca de cilindro (da ultima trilha de um cilindro para a primeira do seguinte), o tempo necessário é quase o de assentamento. Como o tempo de assentamento aumenta com o aumento da densidade de TPI, e as trilhas nos diferentes pratos estão ficando menos alinhadas devido a esse aumento de densidade, o tempo de troca de cabeças está se aproximando do tempo de troca de trilha.

Nos discos de hoje, o assentamento das cabeças para operações de leitura utiliza uma técnica agressiva e otimista, onde a unidade de disco tenta efetuar a leitura tão logo a cabeça esteja próxima da trilha correta: se o dado estiver ilegível pelo fato de a cabeça ainda não ter sido corretamente assentada, nada é perdido. Por outro lado, se o dado estiver disponível, podem ter sido economizados alguns milissegundos de atraso rotacional e de assentamento de cabeça. Há informações de correção de erro (tipicamente através de códigos de Reed-Solomon) e identificação de dados suficientes para garantir que os dados não sejam interpretados de forma incorreta. Por razões óbvias, essa abordagem otimista não pode ser aplicada em assentamentos imediatamente anteriores à operações de escrita. A diferença dos tempos de assentamento para escrita e leitura pode chegar a 0,75ms!

A Figura 2.3 mostra o comportamento do tempo de busca em função da distância a ser percorrida para a localização do cilindro desejado para o disco SCSI IBM 0661 de 3½" de diâmetro com 949 cilindros. A curva é obtida com base na Equação 2.1 (Lee et al., 1991b), onde x é a distância em cilindros a ser percorrida, $miSeek$, $avSeek$ e $maSeek$ são os tempos mínimo, médio e máximo de busca no disco em questão, e $numCyl$ é o número de cilindros do disco. O tempo mínimo é o tempo

para troca de trilha (na maioria dos casos, o tempo de assentamento); o tempo máximo é o tempo de *full stroke*..

$$\text{tempo de busca} = \begin{cases} 0, & \text{se } x = 0 \\ a\sqrt{x} + b/x + c, & \text{se } x \neq 0 \end{cases}$$

$$a = (410\text{miSeek} + 215\text{avSeek} + 45\text{maSeek}) / (3\sqrt{\text{numCyl}})$$

$$b = (7\text{miSeek} + 415\text{avSeek} + 28\text{maSeek}) / (3\text{numCyl})$$

$$c = \text{miSeek}$$

Equação 2.1 Tempo de busca por Lee et al., 1991

Figura 2.3 Tempo de busca em função da distância em cilindros segundo Lee et al., 1991b

O tempo médio pode ser calculado de várias formas. Para Ruemmler (Ruemmler et al., 1994), se as requisições forem totalmente independentes umas das outras, então a distância média de busca é de um terço do número de cilindros do disco. Por isso, segundo Ruemmler, alguns afirmam que o tempo médio de busca é um terço do tempo de *full stroke* ($\text{avSeek} = \text{tempo de busca}(\text{numCyl})/3$), enquanto outros dizem que é o tempo de busca por um terço do número de cilindros ($\text{avSeek} = \text{tempo de busca}/\text{numCyl}/30$). Outra forma é utilizar a Equação 2.2, que calcula o tempo médio como uma média simples dos tempos de busca de 1 a numCyl cilindros (Ruemmler et al., 1994):

$$\text{avSeek} = \frac{\sum_{x=1}^{\text{numCyl}} \text{tempo de busca}(x)}{\text{numCyl}}$$

Equação 2.2 Tempo médio de busca

Outra forma é ponderar o tempo de busca pelo número de buscas possíveis para cada número de cilindros. Assim, há $(\text{numCyl}-1)$ buscas diferentes de tamanho 1 que podem ser feitas em um disco de numCyl cilindros, mas apenas uma busca de tamanho numCyl . Essa abordagem prioriza as pequenas buscas. A Equação 2.3 (Ruemmler et al., 1994) retrata essa situação.

$$avSeek = \frac{\frac{numCyl^{41}}{x+1} / (numCyl - 4x) \cdot tempo_de_busca(x)}{\frac{numCyl^{41}}{x+1} (numCyl - 4x)}$$

Equação 2.3 Tempo médio de busca ponderado

Segundo a literatura disponível, a equação de Lee dá uma boa aproximação do tempo de busca, mas Worthington et al. (Worthington et al., 1995; Worthington et al., 1996) afirmam que os tempos de busca para distâncias até aproximadamente 40 cilindros não se comportam como o previsto pela equação de Lee et al. mas, quanto maior a distância em cilindros a ser percorrida, mais próximo da curva fica o tempo real.

Ruemmler (Ruemmler et al., 1994) propõe outras formas de cálculo do tempo médio: interpolação linear simples ou o uso da Equação 2.4. A interpolação linear simples utiliza os tempos de busca por um cilindro (*miSeek*) e o de *full stroke* (*maSeek*) para o cálculo da interpolação. A equação proposta (Equação 2.4) é apresentada abaixo:

$$tempo_de_busca = \begin{cases} a + b\sqrt{x}, & se\ x \leq B \\ c + d \Delta x, & se\ x > B \end{cases}$$

Equação 2.4 Tempo de busca por Ruemmler et al.

O principal problema da utilização da Equação 2.4 é a escolha do valor da barreira B e das constantes a , b , c e d . Ruemmler afirma que a escolha dos valores deve ser feita experimentalmente.

Ruemmler et al. mostram experimentalmente que o tempo médio de busca do disco HP C2200A para uma determinada carga de trabalho é de 25,36 ± 0,09ms. Com o uso da interpolação linear para cálculo do tempo de busca, o tempo médio de busca ficou em 22,08 ± 0,08ms e, com a utilização da Equação 2.4 devidamente parametrizada para o disco em questão, o tempo médio de busca ficou em 24,31 ± 0,08ms.

O tempo calculado pode ser aproximado ainda mais do tempo real com a inclusão do cálculo do atraso rotacional. Sabendo-se que o disco roda a uma velocidade angular constante, é possível saber, em um determinado momento, a posição angular de um setor em relação às cabeças de leitura/gravação e, assim, calcular o atraso rotacional. Incluir esta variável no cálculo do tempo de busca faz com que o tempo médio se aproxime ainda mais da realidade (25,49 ± 0,09ms).

Tanto a Equação 2.1 quanto a Equação 2.4 estão fundamentadas no fato de os tempos de busca para pequenas distâncias ser proporcional à raiz quadrada da distância, e linearmente proporcional

para grandes distâncias. A vantagem da Equação 2.1 é que não é necessário estimar o ponto onde a relação deixa de ser proporcional a raiz quadrada e passa a ser linearmente proporcional.

Lee afirma (Lee, 2000) que a Equação 2.1, entretanto, não fornece valores muito próximos da realidade para os discos atuais, pois as características de tempo de busca hoje são diferentes das características dos discos antigos (até a metade dos anos 90). Mesmo assim, a equação pode ser usada como uma aproximação razoável do tempo de busca, desde que as três constantes sejam positivas.

Outra forma de efetuar a estimativa do tempo de busca é elaborar uma tabela desses tempos através de medidas experimentais para uma determinada quantidade de distâncias. De posse desses tempos, os valores não tabelados são obtidos por interpolação linear. Essa abordagem é a que dá a melhor estimativa para os discos atuais, sendo que o seu maior problema é a necessidade da disponibilidade do disco para efetuar as medições.

Os tempos de busca podem ser afetados pela **recalibração térmica**. A recalibração é disparada por mudanças de temperatura (que causam expansão do braço e, em alguns casos, pratos) ou por um temporizador. Durante a recalibração, o mecanismo do disco pára de atender requisições para atualizar a tabela de forças que devem ser aplicadas ao pivô para deslocar os braços um certo número de cilindros. Isso pode causar problemas em ambientes que precisam de uma largura de banda garantida ou em sistemas de tempo real.

Os discos magnéticos utilizam, também, técnicas de deslocamento de trilha (*track skewing*) e de cilindro (*cylinder skewing*). Nessa técnica, cada trilha tem seu setor zero lógico deslocado de um determinado fator, que varia conforme a zona (ou *notche*, conforme explicado à frente). Esse deslocamento deve coincidir com o pior caso de tempo de troca de cabeça ou trilha para que, ao chegar na trilha e ser assentada, a cabeça já esteja no setor zero, ou muito próxima dele, e possa continuar a leitura ou escrita. Isso tende a reduzir o atraso rotacional. No caso de deslocamento de cilindro, o setor zero lógico do cilindro vizinho é deslocado de um determinado fator, de forma que, ao trocar de cilindro durante uma leitura ou escrita, o disco tenha que esperar o mínimo possível para continuar a operação.

É interessante observar que, em função do lento posicionamento das cabeças e grande taxa de transferência de dados, o desempenho de leitura varia em função da localização de cada bloco que deve ser lido. Por exemplo, se for necessário transferir 1Mb de um disco Seagate ST43401N Elite 3 (Chen et al., 1994), e os dados estiverem organizados de duas formas – seqüencialmente dentro de um cilindro, ou aleatoriamente distribuído em setores de 8Kb – o tempo para a transferência de 1Mb é de aproximadamente 200ms, mas o tempo de posicionamento das cabeças varia de 16ms (arranjo seqüencial) a 2.000ms (no arranjo aleatório). Estas características permitem enquadrar as aplicações

em duas grandes famílias: **altas taxas de dados**, que necessitam de poucos posicionamentos de cabeças através de longos acessos seqüenciais, ou **altas taxas de E/S**, que geram vários posicionamentos de cabeças através de acessos pequenos e mais aleatórios.

O **tempo de transferência de dados** depende da taxa com a qual os dados são transferidos de/para a superfície dos pratos, e é função da velocidade angular de rotação, da densidade da mídia magnética, e da distância radial da cabeça em relação ao centro do prato. Praticamente todos os discos modernos usam uma técnica chamada *zone-bit-recording*, que permite gravar mais dados nas trilhas externas (que são linearmente maiores), que nas trilhas internas (que são linearmente menores), mantendo a densidade linear dos dados próximo do máximo suportado pela mídia. O número de setores varia em passos. Há, nos discos atuais, entre 10 e 20 zonas (chamadas *notches*) com um número constante de setores. A taxa de transferência de uma unidade de discos magnéticos também depende de como os dados são fisicamente armazenados no disco. Dependendo do tipo de cabeças de leitura/gravação, do servo-motor e do pivô utilizados, é mais rápido trocar de cabeça ou de trilha, gerando as duas formas básicas de mapeamento de dados no disco: vertical ou horizontal.

No mapeamento vertical (Figura 2.4, adaptada de Gross, 1997), utilizado pela grande maioria das unidades de disco, os dados são escritos primeiro em um cilindro, até completá-lo, para, depois, passar para o cilindro seguinte. Nessa forma de mapeamento, a taxa de transferência (Gross, 1997) é função inversa, e o tempo de busca, função direta da quantidade de dados armazenados na unidade.

Figura 2.4 Mapeamento vertical

Já no mapeamento horizontal (Figura 2.5, adaptada de Gross, 1997), o dado é escrito a partir da trilha mais externa para a mais interna, antes de trocar de cabeça e de prato. Neste mapeamento, a taxa de transferência começa alta e reduz, até chegar a um mínimo, e volta a subir, repetindo esse processo a cada troca de cabeça. O tempo de busca começa baixo e aumenta até chegar a um máximo e depois volta ao mínimo, repetindo o processo a cada troca de cabeça. Esta forma de mapeamento é bem menos comum que o mapeamento vertical.

Como pôde ser observado na Figura 2.4 e na Figura 2.5, a taxa de transferência é maior nos cilindros externos. A razão é a utilização dos *zone-bit-recording*. Esta é a razão dos saltos nas taxas de transferência apresentadas acima.

Figura 2.5 Mapeamento horizontal

Algumas unidades de disco utilizam uma combinação de mapeamento horizontal e vertical, mostrada na Figura 2.6. O mapeamento horizontal é utilizado dentro das zonas, e o vertical entre as zonas. O comportamento da taxa de transferência e do tempo de busca é o mesmo do mapeamento vertical.

Figura 2.6 Mapeamento horizontal e vertical

Taxas de transferência típicas variam entre 5 e 20 Mbytes/s nas unidades de discos atuais. O conjunto de pratos é acondicionado, junto com os braços e o pivô, em uma embalagem rígida e hermeticamente fechada para proteger os pratos de contato com poeira, líquidos, partículas de sujeira e ar.

As **cabeças de leitura/gravação flutuam** a cerca de 0,013mm da superfície dos pratos, o que leva a necessidade da embalagem hermeticamente fechada (uma partícula de poeira tem, em média, 0,076mm). A flutuação é causada pelo fluxo de ar gerado pela rotação dos pratos. Exceto em casos especiais, as cabeças nunca tocam a superfície do prato. Se o contato ocorrer, os dados armazenados no ponto de contato podem ser perdidos devido a remoção da mídia magnética pela cabeça de leitura/gravação.

O contato planejado das cabeças de leitura/gravação com a superfície dos pratos ocorre no momento em que a unidade de disco pára de girar, isto é, é desligada. Durante o período de redução da velocidade (*spin down*), o fluxo de ar diminui e então pára completamente, fazendo com que as cabeças caiam sobre os pratos. O contato não planejado ocorre durante um choque físico da unidade,

fazendo com que as cabeças de leitura/gravação atravessem o fluxo de ar e atinjam a superfície dos pratos.

Durante a redução de velocidade programada, o pivô guia os braços e as cabeças de leitura/gravação para uma área especial do disco chamada de **zona de pouso** (*landing zone* – LZ). Nos computadores mais antigos, essa área precisava ser informada manualmente ao BIOS, mas os sistemas mais modernos são capazes de obter essa informação automaticamente a partir dos discos. A LZ nunca contém dados e é uma área de repouso segura para as cabeças. O deslocamento dos braços para a LZ pode se dar por interferência direta do operador (através de algum comando), ou automaticamente no momento do corte do fornecimento de energia para a unidade (Gross, 1997; Randall, 1998a).

As unidades de disco são conectadas ao equipamento computacional através de alguma espécie de **barramento ou interface**. Por definição, “uma interface é alguma coisa que permite que dois componentes separados ou não similares trabalhem em conjunto ou se comuniquem” (Quantum, 1999). Nos periféricos, como os discos rígidos, a comunicação entre eles e o sistema computacional é feita através placas adaptadoras ou chips na placa-mãe que se conectam ao barramento do sistema.

2.2 Barramentos

O **barramento** serve como um condutor comum para o transporte de sinais de e para os vários componentes do sistema. Várias arquiteturas de barramentos foram desenvolvidas, algumas abertas (especificações públicas, tornando possível o desenvolvimento de produtos para o barramento por qualquer pessoa), outras fechadas (especificações restritas, sendo que os produtos para o barramento só podem ser desenvolvidos pelos proprietários da especificação).

As arquiteturas abertas, principalmente o *PC/AT Integrated System Attachment* (ISA) dos primeiros PCs IBM-compatíveis, permitem que desenvolvedores externos projetem, construam e vendam placas de expansão de uso geral ou específico. Algumas das arquiteturas utilizadas em computadores pessoais atualmente são:

##ISA: Um barramento de baixo custo, simples, disponível em versões de 8 e 16 bits;

##Local Bus: Desenvolvido originalmente para aplicações com uso intensivo de gráficos, utilizado também para a conexão de discos de alta velocidade em computadores IBM-compatíveis;

##Micro Channel Architecture (MCA): um barramento proprietário de 32 bits desenvolvido pela IBM para a sua linha OS/2 de computadores pessoais;

##NuBus: barramento proprietário de 32 bits desenvolvido pela Apple para os computadores Macintosh;

- ## *Personal Computer Memory Card International Association (PCMCIA)*: barramento de 16 bits utilizado com frequência em computadores portáteis;
- ## *Peripheral Components Interface (PCI)*: barramento com 32 bits, rodando a 33MHz, podendo chegar a 64 bits a 66MHz.

O uso eficiente do barramento é um fator chave para o aumento de desempenho dos sistemas computacionais. A velocidade de transferência de dados através do barramento depende da largura do barramento e do tempo de ciclo (*cycle time*). Os barramentos, que em suas primeiras versões tiveram 4 e 8 bits de largura e taxas de transferências de dados de 1Mb/s, hoje possuem larguras típicas de 16 ou 32 bits e taxas de transferência de 10, 20, chegando a 132Mb/s no barramento PCI. O próximo desenvolvimento lógico para os barramentos é a largura de 64 bits, permitindo velocidades de transferência ainda maiores.

Entre o barramento do sistema e o disco rígido deve existir uma **interface de disco** responsável pela comunicação entre disco e barramento. As informações especificadas pelas interfaces de disco são, entre outras, quão rápido o disco e a controladora devem conversar entre si, que tipo de comandos eles entendem, a localização das linhas de controle e de dados nos cabos de interconexão, e que níveis de tensão eles devem usar para transferência de dados. Dessa forma, “uma interface de disco é uma combinação padronizada de configuração de conectores, níveis de sinal e protocolos de funções, comandos e transferência de dados” (Quantum, 1999)

Nos computadores pessoais de hoje, são duas as interfaces mais utilizadas: IDE e SCSI, descritas nas seções seguintes.

2.3 Interface IDE

IDE (*Integrated Drive Electronics*) e **ATA** (*AT Attachment*) são essencialmente a mesma coisa: uma implementação de interface para unidade de disco projetada para integrar a controladora no próprio disco, reduzindo o custo da interface e facilitando a implementação do *firmware*. Em razão de seu baixo custo, desde a sua criação, em 1988, ATA se tornou padrão para sistemas computacionais de baixo custo (Wehman et al., 1996).

Há vários padrões diferentes associados com IDE e ATA, o que gera uma grande confusão quando se tenta entendê-los e verificar o que é compatível com o que. Este problema é causado pela não concordância, entre as companhias que produzem discos rígidos, em um padrão de nomeação para as implementações e melhoramentos do padrão ATA real. O resultado é uma confusão de padrões, alguns dos quais são reais enquanto outros são apenas estratégia de *marketing*.

Os novos padrões são geralmente criados para definir modos de transferência novos e mais rápidos, e outros melhoramentos de desempenho, confiabilidade ou compatibilidade. A maior parte desses novos padrões surge informalmente, alguns são eventualmente transformados em padrões ANSI.

As taxas de transferência alcançadas pelas primeiras unidades de disco conectadas a uma interface IDE eram da ordem de 800Kb/s, e essa transferência era feita pelo próprio processador através de PIO (*Programmed Input/Output*). Enquanto um processador está executando uma PIO, ele não pode executar outra tarefa (ASP, 1998). Os modos de PIO estão representados na Tabela 2.1.

<i>Modo PIO</i>	<i>Taxa Máxima de Transferência (Mb/s)</i>
0	3,3
1	5,2
2	8,3
3	11,1
4	16,6

Tabela 2.1 Modos PIO

O padrão ATA-2 é uma extensão compatível com o padrão ATA que introduz melhoramentos, basicamente, no desempenho da interface. Esse melhoramento é obtido com a inclusão de características como PIO e modos de DMA (*Direct Memory Access*) mais rápidos. O padrão também incluiu a possibilidade de o próprio dispositivo informar suas características para a controladora (*Identify Drive*), ponto fundamental para sistemas *plug'n' play*. Há vários modos de DMA, listados na Tabela 2.2, que são usados para definir as taxas de transferência quando DMA é utilizado para transferências através da interface IDE/ATA.

<i>Modo DMA</i>	<i>Taxa Máxima de Transferência (Mb/s)</i>
<i>Single Word 0</i>	2,1
<i>Single Word 1</i>	4,2
<i>Single Word 2</i>	8,3
<i>Multiword 0</i>	8,3
<i>Multiword 1</i>	13,3
<i>Multiword 2</i>	16,6
<i>Multiword 3 (DMA 33)</i>	33,3

Tabela 2.2 Modos DMA

Os modos DMA *single word* estão obsoletos e não são mais utilizados. A maior parte dos discos rígidos que são configurados para utilizar modos DMA utilizam modo *multiword* 2 ou 3.

O padrão ATA-2 foi implementado de duas formas: EIDE (*Enhanced IDE*, proposto pela Western Digital) e Fast-ATA (proposto pela Seagate e Quantum). A principal diferença entre as duas

implementações é que o EIDE engloba o padrão ATA-PI (*ATA Packet Interface*), utilizado por CD-ROMs e unidades de fita.

Tanto IDE quanto EIDE e Fast-ATA permitem, apenas, dois dispositivos conectados ao mesmo canal, e até quatro canais por computador, limitando o número de dispositivos IDE/EIDE/Fast-ATA a oito. Cada canal funciona de forma independente. Dentro de um canal, os dois dispositivos trocam entre si o controle sobre o mesmo. Se houver uma unidade de disco e um CD-ROM no mesmo canal, a unidade de disco tem que aguardar até o término da requisição que está sendo tratada pelo CD-ROM. Uma vez que os CD-ROMs são relativamente mais lentos, há uma degradação de desempenho. Essa é a razão pela qual normalmente CD-ROMs e unidades de disco não ficam no mesmo canal (Wehman et al., 1996; ASP, 1998).

Para diferenciar os dois dispositivos dentro de um mesmo canal é utilizada a designação de **mestre** e **escravo**. A idéia de hierarquia imposta pelos nomes não é real. O dispositivo mestre não tem nenhum *status* especial sobre o escravo e eles são iguais em quase todos os aspectos. A única diferença prática é que, nos PCs, a atribuição de nomes para os drives é feita na ordem mestre-escravo, ou seja, se houver um disco mestre e um disco escravo no mesmo canal, e cada um com uma partição primária, o mestre receberá a letra *C*, e o escravo, a *D*, no caso do DOS/Win. Desta forma, o disco mestre no canal primário é o responsável pelo *boot* do sistema.

Apesar de as taxas de transferências suportadas pela interface chegarem a 16,6Mb/s, a maior parte das unidades de disco IDE não atinge taxas maiores que 10Mb/s. Como apenas um dispositivo pode utilizar o canal por vez, dificilmente a interface se torna o gargalo de desempenho do subsistema de memória secundária em um sistema computacional. (Wehman et al., 1996).

Pelo fato de a transferência de dados entre o dispositivo e a memória ser feita pelo processador (PIO), interfaces IDE/EIDE tendem a gerar sobrecarga na UCP. Esse problema foi minimizado com as interfaces *bus mastering EIDE*, que vem incorporadas aos *chip sets* Triton e Triton II da Intel. Essas interfaces transferem dados diretamente para a memória, através de DMA, usando o *bus mastering* (Stam, 1996; Wehman et al., 1996).

O uso de *bus mastering* é, algumas vezes, uma grande idéia e pode fornecer um desempenho muito melhor em algumas situações. Deve-se, no entanto, tomar alguns cuidados. Seu uso é relativamente recente, e geralmente há problemas em colocá-lo funcionando, pois é necessário suporte por parte do disco, do sistema operacional e do BIOS, além de *drivers* de dispositivo especiais.

A Tabela 2.3 apresenta um breve resumo das variantes da interface IDE/ATA e que modos de PIO e DMA são suportados em cada uma.

<i>Padrão da interface</i>	<i>Tipo de padrão</i>	<i>Modos de PIO</i>	<i>Modos de DMA</i>	<i>Características adicionais em relação ao padrão IDE/ATA</i>
IDE/ATA	ANSI	0, 1 e 2	<i>Single word</i> 0, 1 e 2 <i>Multiword</i> 0	—
ATA-2	ANSI	0, 1, 2, 3 e 4	<i>Single word</i> 0, 1 e 2 <i>Multiword</i> 0, 1 e 2	Transferência em blocos, LBA, melhoramentos no comando identify drive
Fast ATA	<i>Marketing</i>	0, 1, 2 e 3	<i>Single word</i> 0, 1 e 2 <i>Multiword</i> 0 e 1	Mesmas do ATA-2
Fast ATA-2	<i>Marketing</i>	0, 1, 2, 3 e 4	<i>Single word</i> 0, 1 e 2 <i>Multiword</i> 0, 1 e 2	Mesmas do ATA-2
ATA-3	Não oficial	0, 1, 2, 3 e 4	<i>Single word</i> 0, 1 e 2 <i>Multiword</i> 0, 1 e 2	Mesmas do ATA-2, melhor confiabilidade e S.M.A.R.T.
Ultra ATA	Não oficial	0, 1, 2, 3 e 4	<i>Single word</i> 0, 1 e 2 <i>Multiword</i> 0, 1, 2 e 3	Mesmas do ATA-3
ATA-PI	ANSI	0, 1, 2, 3 e 4	<i>Single word</i> 0, 1 e 2 <i>Multiword</i> 0, 1 e 2	Suporte para dispositivos diferentes de discos rígidos
EIDE	<i>Marketing</i>	0, 1, 2, 3 e 4	<i>Single word</i> 0, 1 e 2 <i>Multiword</i> 0, 1 e 2	Mesmas do ATA-2 e ATA-PI

Tabela 2.3 Comparação de padrões IDE/ATA

A Intel e seis parceiros, incluindo fabricantes e desenvolvedores de sistemas, anunciaram em fevereiro de 2000 (IDG, 2000) a criação do grupo *Serial ATA Working*, que vai trabalhar na substituição da atual interface de discos ATA. O novo padrão alcançará velocidades de 1,6 Gbps, de acordo com a Intel — a ATA de hoje atinge menos do que 1 Gbps. Espera-se que a interface Serial ATA chegue até 6 Gbps. Não há previsão de quando protótipos da nova interface estarão disponíveis.

2.4 Interface SCSI

O protocolo e a interface **SCSI** (*Small Computer System Interface*) foram propostos em 1986. A primeira versão permitia que até oito dispositivos se comunicassem através de um barramento de 8 bits em velocidades entre 4 a 5Mb/s.

Cada dispositivo presente no barramento (incluindo a controladora) possui um identificador próprio, variando de 0 a 7 (nos barramentos de 8 bits) ou de 0 a 15 (nos barramentos de 16 bits), chamado de **SCSI ID**. Os dispositivos que requisitam processos de E/S são chamados iniciadores (*initiators*), e os que executam as E/S, alvos (*targets*). Através da controladora embutida, cada alvo pode acomodar e controlar até oito outros dispositivos, chamados de unidades lógicas, e a cada um é atribuído um **número lógico de unidade** (LUN – *Logical Unit Number*). Os comandos para a controladora SCSI do alvo identificam os dispositivos com base no seu LUN. Os LUNs dentro de um mesmo alvo devem ser únicos, mas podem se repetir dentro do mesmo barramento.

O protocolo SCSI consiste numa série de **fases**, durante as quais ações específicas são executadas pela controladora e pelos dispositivos. Pelo fato de ser um protocolo de alto nível com base em troca de mensagens, ele possui mais sobrecarga que os protocolos mais simples. Na primeira

versão do padrão, ao acessar quatro dispositivos simultaneamente, cerca de 25% da banda de comunicação disponível é utilizada pelo protocolo (Chervenak, 1990).

As fases do protocolo são:

##*Bus Free*: Nenhum dispositivo está usando o barramento;

##*Arbitration*: Quando o barramento fica livre, vários dispositivos podem solicitá-lo (*arbitrate*). Durante a solicitação, a prioridade é dada ao dispositivo com maior identificador¹;

##*Selection*: Depois que uma *arbitration* de um iniciador tem sucesso, a fase *selection* informa ao dispositivo alvo que ele participará de uma operação;

##*Reselection*: Depois que uma *arbitration* de um alvo tem sucesso, inicia-se a fase de *reselection*. Isso acontece quando um alvo quer continuar uma operação interrompida por uma mensagem *disconnect*;

##*Command*: Durante esta fase, o iniciador lê os bytes de comando SCSI da memória do *host* (na grande maioria dos casos, o computador) e os envia ao dispositivo alvo através do barramento SCSI;

##*Data Transfer*: Nesta fase, bytes de dados são enviados através do barramento SCSI de e para os dispositivos iniciador e alvo. Na sub-fase *data-in*, bytes são transmitidos do alvo para o iniciador enquanto que na sub-fase *data-out*, a transmissão é do iniciador para o alvo;

##*Message Phase*: Neste momento, os dispositivos iniciador e alvo podem trocar mensagens entre si. As sub-fases são apresentadas a seguir;

##*Status Phase*: Logo após o envio da mensagem indicando que a *message phase* está completa, o dispositivo alvo envia ao iniciador informações sobre a situação do comando.

Na *message phase*, ocorre a troca de comandos entre os dispositivos. As mensagens podem ser de entrada (I) ou de saída (O), em relação ao iniciador, e são as seguintes:

##*Identify (I)*: Enviada pelo alvo depois da fase de *reselection* indica qual dispositivo vai continuar a operação;

##*Save Data Pointer (I)*: Esta mensagem enviada pelo alvo imediatamente antes de uma desconexão diz ao iniciador para salvar a posição corrente da operação de transferência de dados, de forma que a operação de transferência possa ser continuada sem a necessidade de retransmitir bytes de dados;

¹ A sequência geral de prioridade é: 7, 6, ..., 1, 0, 15, 14, ..., 9, 8.

##*Restore Data Pointer* (I): Ao enviar esta mensagem, o alvo informa ao iniciador para recuperar a posição de transmissão de bytes de dados, armazenada com a mensagem *save data pointer*;

##*Disconnect* (I): O alvo envia esta mensagem para informar que o barramento será liberado por ele. Essa liberação se dá no momento em que o alvo vai executar alguma tarefa interna;

##*Command Complete* (I): Mensagem enviada no momento em que o alvo quer informar ao iniciador que uma operação está concluída;

##*Identify* (O): Mensagem enviada pelo iniciador no início de uma operação que informa aos dispositivos no barramento a identidade do iniciador e do alvo desejado.

A mudança de estados no protocolo para uma operação de leitura típica podem ser observadas no *statechart* da Figura 2.7 (adaptada de Chervenak, 1990).

No momento em que um dispositivo consegue selecionar o barramento (fase de *selection*), o sinal ATTENTION (ATN) é injetado no mesmo. Este sinal tem por objetivo avisar ao dispositivo alvo que o dispositivo iniciador quer enviar uma mensagem. Isso é necessário pois todos os bytes de dados ou de comandos enviados pelo barramento são solicitados (*requested* – REQ) pelo alvo e confirmados (*acknowledged* – ACK) pelo iniciador, independente da direção da transferência.

No caso da operação de leitura, mostrada na Figura 2.7, o disco responde ao sinal ATN emitido pela controladora com o envio do sinal REQ para o iniciador, que responde com uma mensagem de saída *identify* e o sinal ACK. O protocolo REQ/ACK se repete para todas as outras fases.

Para o protocolo SCSI, uma transferência é assíncrona quando o barramento espera pelo *handshake* REQ-ACK para cada byte enviado. Isso reduz o desempenho para perto de 3Mb/s na melhor das hipóteses. Se, ao contrário, o dispositivo tiver permissão de enviar um certo número de pulsos REQ sem esperar pelos pulsos ACK, tem-se uma transferência síncrona. Esse número de pulsos REQ é chamado de *offset* e é o número máximo de pulsos REQ sem resposta permitido no barramento. O *offset* elimina a necessidade de espera por um ACK para cada REQ e, com isso, tem-se uma utilização mais eficiente da largura de banda do barramento.

A transferência de dados segue até que o comando seja completado ou surja a necessidade de desconexão. Em uma leitura, uma desconexão irá ocorrer toda vez em que o *buffer* do disco esvaziar (a taxa de transferência do disco pode ser inferior à suportada pelo barramento), ou haja uma troca de cilindro, gerando a necessidade de uma busca pelo próximo cilindro. Nas escritas, a desconexão ocorre quando o *buffer* do disco está cheio de dados enviados pelo iniciador que precisam ser gravados, e a operação é retomada assim que o *buffer* atingir um nível que permita o recebimento de mais dados.

Figura 2.7 Transição de fases do protocolo SCSI para operação de leitura

No que diz respeito ao sinal elétrico, os barramentos SCSI utilizados para troca de mensagens podem ser *single ended* ou *differential*. Nos barramentos *single ended*, os sinais são níveis TTL (0 e 5V) e são relativos à referência de terra comum. Em barramentos *differential*, o sinal é levado por dois fios, cada um com a imagem do outro. O sinal “1” é representado por uma tensão positiva em um fio, e uma igual tensão, só que negativa, no outro fio; e o “0” é representado pelo terra elétrico ou uma tensão de 0V nos dois fios. Os barramentos *differential* são mais resistentes a ruídos e podem ter comprimento de até 25 metros.

O protocolo evoluiu e gerou outras especificações e implementações, apresentadas de forma resumida na Tabela 2.4 (adaptada de Wehman et al., 1996 e Kozierok, 1998, e considerando

barramentos *single ended*). O padrão SCSI-2 trouxe algumas melhorias em relação ao SCSI-1, listadas a seguir (Wehman et al., 1996; Kozierok, 1998; Randall, 1998b):

- ≠ Adoção de barramentos mais largos, com 16 bits, podendo chegar a 32 bits com a utilização de um segundo cabo (chamado “*B-cable*”);
- ≠ Aumento do clock, de 5 para 10, 20 e, depois, 40Mhz no SCSI-3;
- ≠ Novos comandos foram adicionados aos protocolos para permitir a utilização de outros dispositivos, como *scanners* e leitoras/gravadoras de CD-ROM;
- ≠ Enfileiramento de comandos, que permite até 256 comandos por iniciador para cada LUN;
- ≠ A controladora do alvo é capaz de reordenar os comandos para obter a máxima eficiência possível na execução dos mesmos;
- ≠ Inclusão de comandos para permitir diagnóstico de dispositivos.

<i>Protocolo</i>	<i>Padrão (ano)</i>	<i>Vel. do Barramento (MHz)</i>	<i>Largura do Barramento (bits)</i>	<i>Taxa de Transferência (Mb/s)</i>	<i>Tamanho Máximo do Cabo (m)</i>	<i>Número de Dispositivos por Barramento</i>
SCSI “Regular”	SCSI-1 (1986)	5	8	4,77	6	8
Wide SCSI	SCSI-2 (1990)	5	16	9,54		16
Fast SCSI		10	8	9,54	3	8
Fast Wide SCSI			16	19,7		16
Ultra SCSI	SCSI-3 (1996)	20	8	19,7	1,5	8
Ultra Wide SCSI			16	38,15		16
Ultra-2 SCSI		40	8	38,15	N/D	8
Ultra-2 Wide SCSI			16	76,3		16
Serial SCSI ou IEEE1394 Firewire		100 a 400+	1	12 a 50	—	—

Tabela 2.4 Sumário dos protocolos e padrões SCSI disponíveis

A Adaptec lançou, no início de 1999, um controladora Ultra-2 SCSI com dois canais que, segundo a empresa, atinge velocidades de 160Mb/s. Segundo informações da empresa, a combinação da tecnologia Ultra2 SCSI, da arquitetura de 64 bits e de dois canais, conectando até 30 periféricos, reduz a possibilidade de congestionamento no sistema. Além disso, a interface PCI de 64 bits maximiza os recursos dos dois canais SCSI, aumentando a taxa de transferência no barramento PCI para até 266 MB/s.

2.5 Comparações entre IDE e SCSI

Dizer qual a melhor interface depende, basicamente, das necessidades do usuário. Ambientes multitarefa são especialmente beneficiados pela interface SCSI, uma vez que acessos simultâneos ocorrem com frequência e a interface é capaz de lidar bem com essa situação. Em ambientes

monotarefa, o desempenho da interface IDE é geralmente maior. Além disso, com poucos dispositivos (dois ou três), o desempenho da interface IDE é superior ao da interface SCSI, mesmo em ambientes multitarefa. Quando o número de dispositivos no barramento aumenta, o desempenho da interface SCSI é bem maior (Stam, 1996; Wehman et al., 1996; Kozierok, 1998).

Os discos internos SCSI geralmente aquecem mais que os discos EIDE/ATA, mas isso não é uma característica da interface: normalmente os discos SCSI possuem velocidade angular maior e, conseqüentemente, aquecem mais. Alguns dos discos mais novos precisam de um *cooler* para mantê-los em temperatura operacional adequada.

De uma maneira geral, a interface SCSI é tecnologicamente superior à IDE: ela suporta mais dispositivos, permite multitarefa sem degradação significativa de desempenho, facilita a expansão e permite a conexão de uma variedade maior de dispositivos.

O custo da interface IDE é inferior à SCSI em quase todas as situações. A única exceção seria a utilização de mais de oito dispositivos, o que torna necessário uma solução especial usando IDE, enquanto que na interface SCSI, até 15 dispositivos podem ser acomodados sem problemas. O custo das unidades de disco não é uma boa comparação, uma vez que são dispositivos com tecnologias completamente diferentes e, geralmente, tecnologias diferentes têm custos diferentes.

Comparar o desempenho de interfaces SCSI e IDE não é uma tarefa fácil, pois depende do desempenho dos dispositivos individualmente, da taxa de transferência máxima da interface, do ambiente ser mono ou multitarefa e da utilização ou não do *bus mastering* (disponível tanto na interface SCSI quanto na IDE). Também é importante observar o problema da mistura de dispositivos: enquanto que na IDE um CD-ROM pode tornar o canal utilizado por ele mais lento (CD-ROMs usam o protocolo ATA-PI), na SCSI este problema não existe, pois todos os dispositivos usam o mesmo protocolo. De maneira geral, a interface SCSI tem um desempenho melhor. Para aplicações muito simples (um disco rígido e um CD-ROM, por exemplo), a interface IDE tem uma vantagem pequena, enquanto que para aplicações mais complexas, SCSI tem uma vantagem significativa.

Em 1996, Nick Stam (Stam, 1996) aplicou uma série de testes de desempenho em sistemas SCSI e IDE. Nesses testes pôde-se observar, na prática, que para ambientes monotarefa apenas com discos conectados à interface, o desempenho da interface IDE foi ligeiramente superior ao da SCSI no sistema operacional Windows 95, enquanto que no Windows NT, o desempenho da SCSI foi ligeiramente maior. No ambiente de multitarefa, com sistema operacional Windows NT, o desempenho das interfaces SCSI foi superior ao das interfaces IDE. Quando se acrescenta um CD-ROM à interface, novamente a SCSI apresenta um desempenho melhor.

No quesito de configuração, dizer quem é melhor depende do número de dispositivos conectados à interface. Novamente, para poucos dispositivos, a interface IDE é muito mais simples de configurar que a SCSI e, para muitos dispositivos, a SCSI é melhor. Quanto à capacidade de crescimento, a interface SCSI tem uma grande vantagem sobre a IDE.

Quanto à variedade de dispositivos suportados, novamente a interface SCSI leva uma considerável vantagem sobre a IDE. Enquanto a IDE suporta apenas unidades de disco, CD-ROMs e gravadoras de CD, a interface SCSI suporta, além dos dispositivos citados para IDE, scanners, fitas DAT, *cd changers*, *jukeboxes*, entre outros.

As interfaces SCSI consomem menos UCP que as IDE, apesar que para sistemas operacionais mais antigos, o consumo de memória principal pode ser maior nas interfaces SCSI devido a necessidade de se carregar *drivers* para acesso, do sistema operacional, à controladora.

Ao sair do mundo PC, o suporte à interface SCSI é muito maior que à IDE. Enquanto IDE está limitado a PCs e aos novos iMac da Apple, SCSI pode ser encontrado em PCs, Apples e grande parte das estações de trabalho disponíveis no mercado atualmente.

Por fim, vale reafirmar que SCSI é um protocolo de muito mais alto nível que IDE. Na realidade, enquanto que IDE é apenas uma interface, SCSI é realmente um barramento de sistema, com controladores inteligentes em cada dispositivo SCSI, trabalhando em conjunto para gerenciar o fluxo de informações no canal.

Independente do barramento que conecta os discos, as informações armazenadas nos discos podem ser perdidas por alguma falha ou defeito dos componentes mecânicos que compõe os discos, ou por consequência de um erro de programa que corrompa uma tabela de acesso aos arquivos, por exemplo. Em 1988, Patterson et al. (Patterson et al., 1988) propôs uma forma de organizar os discos com a intenção de reduzir o tempo de acesso e, também, aumentar a confiabilidade dos dados. No capítulo seguinte são abordados aspectos relativos às matrizes de discos redundantes independentes, com suas características, desempenho e problemas.

Esta página foi deixada intencionalmente em branco

Capítulo 3

RAID – Redundant Array of Independent Disks

Este capítulo apresenta uma revisão bibliográfica sobre RAID. Apresenta-se a motivação para seu surgimento, os principais conceitos e seus níveis. Uma comparação básica sobre desempenho e confiabilidade dos diferentes níveis RAID e algumas possíveis soluções para problemas de desempenho de um nível RAID em especial.

O poder computacional dos processadores tem crescido de forma acentuada ao longo dos anos. O problema é que a velocidade de um sistema computacional não pode ser medido apenas pela velocidade da UCP, mas deve levar em consideração quantidade e velocidade de acesso à memória, barramentos, etc... Enquanto os microprocessadores apresentam um aumento de desempenho de 50% ou mais por ano, o tempo de acesso a disco – que depende de sistemas mecânicos – aumenta em menos de 10%. A taxa de transferência, dependente de aspectos mecânicos e de densidade da mídia magnética, tem aumentado cerca de 20% por ano (Patterson94 *apud* Chen et al., 1994).

No momento em que se compara o desempenho das unidades de discos magnéticos utilizadas como memória secundária com a tecnologia de memória principal, percebe-se que a melhora de desempenho dos discos cresce em uma taxa modesta. Esses dispositivos mecânicos são dominados pelo tempo de posicionamento das cabeças. O uso de parte da memória principal como *cache* ou *buffer* para aumentar o desempenho do acesso ao disco pode ser uma boa alternativa, quando os dados apresentam localidade espacial ou temporal. No entanto, este não é o caso de aplicações de processamento *on line* de transações (sem localidade temporal) ou de grandes simulações rodando em supercomputadores (sem localidade espacial).

O aumento de desempenho de apenas parte do sistema gera um aumento global de desempenho que pode ser medido com o auxílio da Lei de Amdahl (Amdahl67 *apud* Patterson et al., 1988), indicada abaixo:

$$S \leq \frac{1}{(1-f) + \frac{f}{k}}$$

Onde:
 S : *speed up* efetivo;
 f : fração de trabalho que ocorre na porção mais rápida;
 k : *speed up* da porção mais rápida
 Equação 3.1 Lei de Amdahl

A Lei de Amdahl diz que “se um componente seqüencial de um algoritmo é responsável por 1/s do tempo de execução do programa, então o *speed up* máximo possível é s” (Foster, 1995). Ela é

interessante quando processos ou componentes tem o seu desempenho aumentado de forma incremental e independente. A interpretação dada por Patterson et al. (Patterson et al., 1988) permite obter o aumento de desempenho global esperado (S) de um sistema computacional quando apenas uma fração do trabalho (f) do mesmo ocorre no subsistema que teve o aumento de desempenho (k). A visualização é facilitada com o auxílio dos gráficos na Figura 3.1, onde a Lei de Amdahl é apresentada para frações de trabalho de 20, 50, 80 e 95%.

Figura 3.1 Exemplos de speed up com a Lei de Amdahl

Para minimizar o problema do desempenho do subsistema de entrada e saída de memória secundária, é possível o agrupamento das unidades de disco em matrizes ou vetores (*arrays*) para que sejam executadas operações de entrada e saída em paralelo. Esse paralelismo é atingido com a fragmentação dos dados (Johnson84 *apud* Chen et al., 1990).

Nessas matrizes de disco, a confiabilidade é um ponto muito importante. Ao assumir uma taxa constante de erro – ou seja, um momento de falha distribuído exponencialmente – e falhas independentes, mesmas premissas adotadas pelos fabricantes para o cálculo do MTTF, a confiança de uma matriz de discos é dada por:

$$MTTF \text{ do Array} = \frac{MTTF \text{ de um disco}}{\text{Número de discos}}$$

Equação 3.2 Tempo médio para falha de matriz de disco

O tempo médio para falha pode ser calculado pela diferença entre o tempo médio entre falhas e o tempo médio para reparo ($MTTF = MTBF + MTTR$). Ao longo desta dissertação, considera-se que o MTTR é igual a zero e, portanto, $MTTF = MTBF$.

O disco Seagate Barracuda 4LP ST34371WD, por exemplo, tem um MTBF (= MTTF) declarado pelo fabricante de 1.000.000 de horas (Seagate, 1997). Em uma matriz com 100 desses discos, tem-se um MTTF de $1.000.000/100 = 10.000$ horas (aproximadamente 1 ano e 1 mês). Em uma matriz, agora, com 1.000 discos, o MTTF se reduz para 1.000 horas, ou seja, 41 dias e meio aproximadamente.

Um MTBF declarado, ou teórico, de 1.000.000 horas, por exemplo, quer dizer que numa grande população de discos o momento de falha médio deve ser em torno de 1.000.000 horas. O MTBF (ou MTTF) não é uma forma de prever a vida útil de um disco ou de um pequeno número de discos. Para atingir o MTBF previsto, um disco deve rodar até atingir o fim do seu período de vida útil ou falhar e então ser substituído por um novo disco de confiabilidade similar e assim por diante. Neste caso, é teoricamente possível que 1.000.000 horas tenham se passado até que ocorra falha em uma grande parte da população.

Os modelos de previsão teórica de MTBF são incapazes de levar em conta fatores como erros de projeto do dispositivo, drive, *firmware* ou equipamento, defeitos induzidos durante a produção, elementos humanos e o ambiente onde o drive será instalado (umidade e temperatura do local por exemplo).

Por outro lado o MTBF operacional é definido como uma falha no sistema, percebida pelo usuário, causada por uma falha no disco que torna a substituição do disco necessária. Algumas empresas (Quantum, 1996), consideram no cálculo do MTBF operacional todos os discos que retornaram de campo em decorrência de falhas excluindo falhas decorrentes de transporte. O MTBF operacional é muito menor do que MTBF teórico na maioria dos casos.

As matrizes de discos utilizam dois conceitos ortogonais: **fragmentação de dados** (*data striping*) para melhorar o desempenho, e **redundância** para melhorar a confiabilidade. A fragmentação de dados distribui dados, de forma transparente, ao longo de vários discos para fazer com que eles pareçam um disco único, grande e rápido. O conceito inicial de fragmentação foi proposto por Johnson, em 1984 (Johnson84 *apud* Chen et al., 1990). A fragmentação melhora o desempenho de E/S por permitir que múltiplas E/S sejam executadas em paralelo. Dois aspectos podem ser considerados nesse paralelismo. Primeiro, requisições múltiplas e independentes podem ser atendidas em paralelo por discos distintos. Isso reduz o tempo de fila das solicitações de E/S. Em 1986, Kim (Kim86 *apud* Chen et al., 1990) provou, utilizando teoria de filas, que uma unidade de fragmentação de 1 byte, para baixas cargas de trabalho, produz um *throughput* muito alto. Segundo,

requisições únicas de múltiplos blocos podem ser servidas por múltiplos discos atuando em coordenação. Isto aumenta a taxa de transferência efetiva vista por uma requisição única. Quanto mais discos na matriz, maiores os benefícios potenciais de desempenho. Infelizmente, um grande número de discos reduz a confiabilidade geral da matriz de discos para um valor abaixo do aceitável, como visto anteriormente.

A maioria das organizações redundantes de discos pode ser diferenciada com base na granulosidade do entrelaçamento dos dados e no método e padrão nos quais a informação redundante é computada e distribuída através da matriz de discos. O entrelaçamento de dados pode ser caracterizado como de granulosidade fina ou granulosidade grossa. Matrizes de discos de granulosidade fina intercalam dados agrupados em pequenas quantidades de bytes, de forma que todas as requisições de E/S, independente do tamanho, acessam todos os discos da matriz. Já as matrizes de disco de granulosidade grossa intercalam dados em quantidades relativamente grandes (muitos bytes), de forma que pequenas requisições de E/S acessam apenas um pequeno número de discos, mas requisições grandes acessam todos os discos da matriz. Isso permite que múltiplas requisições pequenas possam ser atendidas simultaneamente, mas ainda garante às requisições grandes os benefícios de utilizar todos os discos da matriz.

A incorporação de redundância em uma matriz de discos trás dois problemas de alguma forma ortogonais. O primeiro problema é a escolha do método de cálculo da redundância. A maior parte das matrizes de discos usa paridade, mas algumas usam códigos de Hamming ou de Reed-Solomon. O segundo problema é a escolha de um método para distribuir a redundância pelos discos. Apesar de existir uma infinidade de padrões para a distribuição dessa redundância, eles podem ser classificados em dois esquemas distintos: aqueles que concentram a informação redundante em um pequeno número de discos, e aqueles que distribuem a informação redundante de maneira uniforme ao longo dos discos. Esquemas que distribuem uniformemente a informação redundante através dos discos são preferidos, uma vez que evitam pontos de concentração e outros problemas de balanceamento de carga. Apesar dos conceitos básicos de fragmentação de dados e redundância serem conceitualmente simples, a escolha entre as possíveis fragmentações de dados e esquemas de redundância envolve prós e contras de confiabilidade, desempenho e custo.

3.1 Níveis RAID

As combinações de fragmentação e redundância dão origem aos *níveis RAID*. As primeiras combinações propostas deram origem aos cinco níveis originais de Patterson et al. (Patterson et al., 1988) e, com o passar do tempo, a comunidade propôs pelo menos mais cinco combinações. A seguir, serão abordadas as principais características de cada nível para uma posterior avaliação de desempenho, custo e confiabilidade dos mais utilizados.

A matriz de discos implementa uma abstração de espaço linear de endereçamento, aparecendo para o usuário como uma seqüência linear de unidades de dado, numeradas de 0 a $(N*B-1)$, onde N é o número de discos na matriz e B é o número de unidades de dados disponíveis para o usuário em cada disco. As unidades que armazenam redundância não aparecem no espaço de endereçamento, e não são endereçáveis pelo usuário. A controladora da matriz é responsável por mapear os endereços lógicos em posições físicas do disco e executar as operações de manutenção dos dados redundantes (Courtright II et al., 1996).

3.1.1 RAIDs originais

Os RAIDs originais foram os níveis de RAID originalmente propostos por Patterson et al. (Patterson et al., 1988) e revisados por Chen et al. (Chen et al., 1994). Existem as opções de espelhamento, de código de Hamming para correção de erros, de paridade entrelaçada em nível de bit, de paridade entrelaçada em nível de bloco e de paridade distribuída entrelaçada em nível de bloco, correspondendo aos RAIDs nível 1 a 5 respectivamente. Cada uma destas opções será detalhada a seguir. As considerações feitas por Patterson et al. e por Chen et al. são corroboradas por vários outros autores (McLean, 1991; ACN, 1998)

Figura 3.2 RAID 1

A solução tradicional (RAID 1), chamada de espelhamento ou *shadowing*, usa o dobro de discos de uma matriz não redundante. Toda vez que um dado é escrito em um disco, o mesmo dado é escrito em um disco redundante, de forma que sempre há duas cópias da informação, conforme ilustrado na Figura 3.2. Quando um dado é lido, ele pode ser recuperado do disco com a menor fila, menor tempo de busca e atraso rotacional. Se um disco falha, a segunda cópia é usada para atender às requisições. O espelhamento normalmente é usado em soluções onde a disponibilidade do dado é mais importante que a eficiência de armazenamento.

Os sistemas de memória têm oferecido recuperação de componentes falhos com um custo muito menor que a duplicação através de uso de códigos de Hamming. A utilização de códigos de Hamming para a correção de erros dá origem ao RAID 2. Os códigos de Hamming contêm paridades para subconjuntos sobrepostos de componentes distintos. Em uma versão deste esquema, são utilizados quatro discos de dados e três para redundância, um a menos que no espelhamento. O

número de discos redundantes é logaritmicamente proporcional ao número total de discos no sistema. A Figura 3.3 apresenta um RAID 2. Os dados são fragmentados ao longo dos quatro primeiros discos, e os três últimos armazenam os códigos de correção de erros.

Ki a Kj à Palavra K; ECC Kx a Kz à ECC Palavra K

Figura 3.3 RAID 2

No RAID 2, se um componente único falhar, vários dos componentes de paridade terão valores inconsistentes. A informação perdida é recuperada pela leitura dos outros componentes no subconjunto, incluindo os componentes de correção de erros, e pelo ajuste do bit que falta em 0 ou 1 para criar o valor correto de paridade para o subconjunto.

Uma melhora da detecção e correção de erros para os discos é que, ao contrário da memória (utilizando ECC), as controladoras de disco podem identificar facilmente qual disco falhou. Por isso, pode-se utilizar apenas um disco de paridade ao invés de um conjunto de discos para recuperar a informação perdida caracterizando, desta forma, o RAID 3, mostrado na Figura 3.4. Os quatro primeiros discos armazenam os fragmentos de arquivo e o último disco armazena a paridade, calculada pelo gerador de paridade com base nos fragmentos.

Em uma matriz de discos de paridade entrelaçada em nível de bit, o dado é conceitualmente entrelaçado bit-a-bit ou byte-a-byte ao longo dos discos de dados, e um único disco de paridade é adicionado para tolerar uma falha de disco única. Cada requisição de leitura acessa todos os discos de dados e cada requisição de escrita acessa todos os discos de dados e o disco de paridade. Por isso, apenas uma requisição pode ser atendida por vez. Pelo fato de o disco de paridade conter apenas paridade e não dados, o disco de paridade não pode participar em leituras, resultando em um desempenho de leitura levemente menor (Chen et al., 1994; Patterson et al., 1988) que nos esquemas de redundância nos quais a paridade é distribuída ao longo de todos os discos. As matrizes de discos de paridade entrelaçada em nível de bit são freqüentemente utilizadas em aplicações que necessitam de grande volume de dados em pouco tempo mas não de taxas de E/S elevadas. Elas também são simples de implementar.

O desempenho de um RAID 3, ou qualquer outro que faça a fragmentação em pequenas unidades de dados tem a propriedade de ser muito parecido, tanto no modo de operação sem falhas quanto no modo com falhas, onde o dado do disco falho deve ser recuperado com base nas informações de redundância e dos demais discos (Courtright II et al., 1996).

Figura 3.4 RAID 3

A matriz de discos com paridade entrelaçada em nível de bloco – RAID 4 – é similar à matriz de discos de paridade entrelaçada em nível de bit, exceto pelo fato de que o dado é entrelaçado ao longo dos discos em blocos de tamanho arbitrário (32Kb, por exemplo) ao invés de bits/bytes, como mostra a Figura 3.5. O tamanho desses blocos é chamado de “unidade de fragmentação”. Esse RAID é especialmente eficaz em ambientes onde as requisições são basicamente leituras de pequenas quantidades de dados contidas, preferencialmente, em uma unidade de fragmentação, permitindo um paralelismo máximo.

Figura 3.5 RAID 4

Requisições de leitura menores que uma unidade de fragmentação são atendidas com acesso a apenas um disco de dados. Requisições de escrita devem atualizar o bloco de dados que foi modificado e devem também calcular e atualizar o bloco de paridade. Para grandes escritas que atingem blocos em todos os discos, a paridade é facilmente computada pelo cálculo do ou-exclusivo (XOR) entre os dados novos e os antigos. Para pequenas requisições de escrita que atualizam apenas um disco de dados, a paridade é calculada e aplicada ao bloco de paridade. As pequenas requisições de escrita, por isso, necessitam de quatro E/Ss em disco: um para escrever o novo dado, dois para ler os

dados antigos e a paridade antiga para cálculo da nova paridade, e um para escrever a nova paridade. Isso é chamado de procedimento lê-modifica-escreve (*read-modify-write* ou *RMW*). Pelo fato de uma matriz de discos com paridade entrelaçada em nível de bloco ter apenas um disco de paridade, que deve ser atualizado em todas as operações de escrita, esse disco pode facilmente se tornar um gargalo.

O uso de uma matriz de discos com paridade distribuída entrelaçada em nível de bloco (RAID 5) elimina o gargalo do disco de paridade presente no RAID 4 através da distribuição uniforme da paridade ao longo de todos os discos. A Figura 3.6 mostra um arranjo RAID 5 com dados e paridades compartilhando o mesmo disco. Uma vantagem, geralmente não observada, de distribuir a paridade, é que isso também distribui os dados ao longo de todos os discos. Essa característica permite que todos os discos participem do atendimento de uma requisição de leitura em contra ponto aos esquemas de redundância com discos dedicados de paridade. Os RAID 5 têm uma dos melhores desempenhos para pequenas leituras, grandes leituras e grandes escritas em todos os esquemas de matrizes de discos redundantes, como será visto na seção 3.3.2.

As pequenas requisições de escrita no RAID 5 são, de certa forma, ineficientes quando comparadas com esquemas de redundância, como o espelhamento, devido à necessidade de executar operações *RMW* para atualizar a paridade. Essa é a principal desvantagem em termos de desempenho do RAID 5.

Figura 3.6 RAID 5

O método utilizado para distribuir a paridade nos RAID 5 pode ter impacto no desempenho. A Figura 3.6 mostra a melhor distribuição de paridade dentre as estudadas por Lee (Lee et al., 1991a), chamada de distribuição de paridade simétrica à esquerda (*left-symmetric parity distribution*). Uma propriedade útil da distribuição de paridade simétrica à esquerda é que quando se atravessa as unidades de fragmentação sequencialmente, cada disco será acessado uma vez antes de ser acessado pela segunda vez. Essa propriedade reduz os conflitos de disco quando o sistema está atendendo grandes requisições.

3.1.2 RAIDs derivados

Os RAIDs apresentados a seguir foram derivados dos cinco primeiros, apresentados na seção 3.1.1. Nesta seção serão discutidos o não redundante, o com redundância P+Q ou dupla paridade distribuída entrelaçada em nível de bloco, uma solução RAID proprietária, fragmentação com espelhamento, RAID 53 e TickerTAIP.

A matriz de discos sem tolerância a falha, RAID 0 é apresentado na Figura 3.7. O RAID 0 faz, apenas, a divisão do dado em blocos (por exemplo, um arquivo de 4Mb é dividido em 4 blocos de 1Mb) gravando os blocos em discos separados para aumentar o desempenho de escrita. Esse nível RAID também é chamado de conjunto de fragmentos sem paridade (*no-parity strip set*). Esse aumento de desempenho se deve à distribuição da carga de E/S ao longo de vários canais e discos. Obtêm-se o maior aumento de desempenho com o uso de várias controladoras e apenas um disco por controladora. Apesar de ter o melhor desempenho de escrita, pode não possuir o melhor de leitura. Esquemas de redundância como o espelhamento, que duplicam os dados, podem executar leituras melhores, já que podem escalonar as requisições para o disco com a menor previsão de tempo de busca e atraso rotacional.

Figura 3.7 RAID 0

O RAID 0 não oferece tolerância a falhas e, por isso, não tem a sobrecarga do cálculo de paridade. Pode ser usado em conjunto com outras técnicas RAID, como a 3, 4 e 5. Essa matriz tornou-se um nível RAID de fato, apesar de não o ser de direito, por não apresentar redundância. Se apenas um disco falhar, todo o dado é perdido e, por isso, não deve ser usado em ambientes de missão crítica. Seu projeto é muito simples, e sua implementação, fácil.

O RAID 6, também chamado de dupla paridade distribuída entrelaçada em nível de bloco ou redundância P+Q, é essencialmente uma extensão do RAID 5, apresentando tolerância a falha adicional com o uso de um esquema de paridade distribuída independente para suportar falhas múltiplas de disco.

No caso de uma falha de disco no RAID 5, a reconstrução da informação está condicionada ao sucesso da leitura de todos os outros discos não-falhos, e, apesar da probabilidade de isso ocorrer ser alta, é impossível garantir sucesso em 100% das vezes, como será discutido na seção 3.6.3. Se todos os

setores não falhos pertinentes a reconstrução não puderem ser lidos, é impossível reconstruir a informação, gerando perda de dados. No RAID 6, os dados são divididos da mesma forma que no RAID 5, e um segundo conjunto de paridade, utilizando geralmente códigos Reed-Solomon², é calculado e escrito ao longo de todos os discos, como mostrado na Figura 3.8, para suportar uma segunda falha simultânea de disco. No caso de uma falha que impossibilite a leitura do bloco C0 da Figura 3.8, por exemplo, a informação pode ser reconstruída com a leitura dos demais blocos X0 e a paridade 0. Se, na reconstrução, for impossível ler, por exemplo, o bloco B0, o valor do mesmo pode ser reconstruído com o auxílio da informação de paridade B e, depois da reconstrução de B0, o processo de cálculo do valor de C0 pode seguir normalmente. O nível de confiança oferecido pelo RAID 6 é muito alto e pode suportar múltiplas falhas simultâneas de discos sendo, portanto, a solução perfeita para aplicações de missão crítica. O projeto da controladora é bastante complexo e existe uma sobrecarga decorrente do cálculo da segunda paridade, que reduz o desempenho de escrita.

Figura 3.8 RAID 6

O RAID 7 é uma implementação de RAID proprietária, similar ao RAID 4, patenteada pela *Storage Computer Corporation* (MCP, 1998). O objetivo é oferecer altas taxas de E/S e altas taxas de transferência.

Figura 3.9 RAID 7

A Figura 3.9 mostra a estrutura de um RAID 7. Neste nível RAID, todas as transferências são assíncronas, controladas e armazenadas em cache independentemente, incluindo as transferências entre *host* e interface. Todas as escritas e leituras são armazenadas em cache centralmente através do *x-bus*, um barramento interno de alta velocidade usado para transferência de dados de cache. É

² A utilização de códigos Reed-Solomon em RAIDs é bem explorada por Plank em Plank, 1997.

utilizado um sistema operacional de tempo real orientado a processo, implementado e incorporado no microprocessador da controladora, para atingir alto desempenho. O gerador de paridade está incorporado ao cache da controladora. Múltiplos discos podem ser declarados como *hot spares*. Apesar de todas essas características, utiliza discos SCSI convencionais, barramento PCI e memória SIMM (*Single In-Line Memory Module*), além de ter suporte ao protocolo SNMP (*Simple Network Management Protocol*).

O desempenho geral de escrita é de 25 a 90% maior que um disco comum, e de 1,5 a 6 vezes maior que todos os outros níveis RAID. As interfaces no *host* são escaláveis para permitir a conectividade ou aumentar a largura de banda. Em um ambiente multi usuário, a taxa de acerto do cache é muito alta, gerando um tempo de acesso ao disco muito baixo. O desempenho de escrita aumenta com o aumento do número de discos na matriz. A grande desvantagem do RAID 7 é ser uma solução proprietária e patenteada, com um custo por Mbyte muito alto. Além disso, a unidade RAID deve estar conectada a um sistema de fornecimento ininterrupto de energia (UPS) para evitar a perda de dados nos caches no caso de interrupção da energia elétrica.

Os RAIDs se mostram, para o usuário, como um disco único de grande capacidade. É perfeitamente possível utilizar esses RAIDs como discos para montar outros RAIDs do mesmo ou de outro nível, criando uma estrutura em camadas, os chamados **RAIDs em camadas** (ISI, 1999). Desta forma, um RAID 10 é um RAID 0 que utiliza RAID 1 ao invés de discos convencionais, como mostrado na Figura 3.10. O RAID 10 apresenta o mesmo nível de tolerância a falhas do RAID 1, e a mesma sobrecarga do espelhamento.

Figura 3.10 RAID 10

Outro RAID em camadas é o RAID 53. Ao contrário do que o nome indica, este nível utiliza RAID 0, e não RAID 5, sobre RAID 3, como mostra a Figura 3.11. O RAID 53 tem a mesma tolerância a falhas e sobrecarga do RAID 3. É capaz de atingir altas taxas de transferências graças aos segmentos RAID 3, e as altas taxas de E/S para pequenas requisições são atingidas graças aos fragmentos do RAID 0. O conceito de camadas pode ser estendido para todos os níveis RAID e em qualquer número de camadas. O uso de camadas pode oferecer melhora de desempenho e disponibilidade.

Figura 3.11 RAID 53

Os RAID tradicionais apresentam uma arquitetura centralizada, onde todas as requisições passam por uma única controladora. Não estamos falando das controladoras dos discos SCSI convencionais, as quais podem ser alocadas uma para cada disco e a paridade calculada por *software*, mas de interfaces RAID dedicadas que incluem em *hardware* o cálculo da paridade e gerenciam todo o trabalho de alocação dos blocos nos discos e recuperação dos dados após uma falha.

Essa controladora se torna um ponto único de falha, o seu desempenho se torna limitante do desempenho global do RAID. Cao et al. (Cao et al., 1992) propôs uma arquitetura paralela para RAID que distribui as funções de controle ao longo de vários processadores fracamente acoplados, chamada de TickerTAIP, e a esses processadores estão conectadas as interfaces SCSI que contém os discos. O resultado é uma melhora na escalabilidade, tolerância a falhas e flexibilidade, conseguindo um desempenho melhor que um RAID centralizado.

Nesta seção foram apresentadas as estruturas possíveis para RAID, mas a forma como as matrizes de disco são conectadas ao computador foi ignorada. A próxima seção mostra as duas formas possíveis e faz algumas considerações sobre desempenho e confiabilidade.

3.2 Ortogonalidade

A forma como as matrizes de disco são conectadas ao computador afeta os aspectos de desempenho e de confiabilidade. A maior parte dos computadores conecta vários discos ao longo de poucas cadeias (barramentos SCSI, por exemplo). Uma falha na controladora da cadeia gera múltiplas falhas simultâneas de disco. Na maioria dos níveis RAID, a falha simultânea de múltiplos discos pode levar a perda de dados (Chen et al., 1994).

Sobre a matriz de 9 discos da Figura 3.12 é possível a organização dos grupos de paridade de duas formas distintas, identificadas por **Opção 1** e **Opção 2** na figura supra. A opção 1 combina todos os discos de uma cadeia em um mesmo grupo de paridade, enquanto a opção 2 utiliza um disco de cada cadeia para montar o grupo de paridade.

Figura 3.12 RAID Ortogonal

Na opção 1, a falha de uma controladora de cadeia torna inacessível os três discos do grupo, enquanto que na opção 2, a falha da controladora torna inacessível apenas um disco de cada grupo. A perda de um disco do grupo não torna o grupo todo inacessível sendo preferida em relação à opção 1. A opção 2 trás, ainda, o benefício de reduzir a disputa pela cadeia (ou barramento, no caso de SCSI) quando múltiplos discos de um mesmo grupo transferem dados ao mesmo tempo. A técnica de organizar grupos de correção de erro (grupos de paridade) de forma ortogonal em relação ao *hardware* comum (como as cadeias) é chamada **RAID ortogonal** (Schulze89 *apud* Chen et al., 1994).

Como foi possível observar, existe uma grande variedade de tipos de RAID e cada um com seus prós e contras. A próxima seção faz uma análise do desempenho e custo dos RAID mais utilizados.

3.3 Comparação de desempenho e custo

As três métricas principais na avaliação de uma matriz de discos são a confiabilidade, o desempenho e o custo. Os RAID de 0 a 6 possuem várias diferenças entre essas métricas. Para que seja possível compreender o valor e o custo de cada matriz, é necessário avaliar as três métricas. Nessa seção, serão comparados o desempenho e o custo. A confiabilidade será avaliada na próxima seção.

3.3.1 Observações sobre a avaliação

Apesar de haver apenas três métricas, existe uma vasta gama de combinações das mesmas. Por exemplo, o desempenho deve ser medido em E/Ss por segundo, bytes/s ou tempo de resposta? Uma métrica híbrida, como E/Ss por segundo por dólar seria mais apropriada?

Em sistemas orientados a *throughput*, o desempenho pode ser potencialmente aumentado de forma linear com a adição de componentes, ou seja, se um disco fornece trinta E/Ss por segundo, dois discos deveriam fornecer sessenta. Por isso, na comparação de desempenho de matrizes de discos, o desempenho do sistema é normalizada pelo seu custo. Assim, nas métricas de desempenho serão usadas unidades como E/Ss por segundo por dólar ao invés de simplesmente E/Ss por segundo.

3.3.2 Comparações

A Tabela 3.1 (adaptada de Chen et al., 1994) mostra o *throughput* máximo por dólar em relação ao RAID nível 0 para os RAIDs nível 0, 1, 3, 5 e 6. Assume-se que o custo de cada sistema é proporcional ao número de discos na matriz. Assim, a tabela mostra que, entre um RAID nível 0 e um RAID nível 1 com o mesmo custo, o RAID nível 1 pode sustentar metade do número de pequenas escritas por segundo. Da mesma forma, pode-se dizer que o custo de um pequena escrita em um RAID nível 1 é o dobro em relação ao RAID 0.

	<i>Pequenas Leituras</i>	<i>Pequenas Escritas</i>	<i>Grandes Leituras</i>	<i>Grandes Escritas</i>
RAID 0	1	1	1	1
RAID 1	1	1/2	1	1/2
RAID 3	1/G	1/G	(G-1)/G	(G-1)/G
RAID 5	1	Max (1/G, 1/4)	1	(G-1)/G
RAID 6	1	Max (1/G, 1/6)	1	(G-2)/G

Tabela 3.1 *Throughput* máximo por dólar em relação ao RAID 0

Na Tabela 3.1, “pequeno” se refere a uma unidade de fragmentação; “grande” se refere a um fragmento completo (uma unidade de fragmentação de cada disco em um grupo). G significa o número de disco em um grupo, incluindo os de paridade. Em todos os casos, quanto maior o valor, melhor. Os valores dessa tabela consideram apenas os aspectos principais do desempenho, deixando de lado alguns efeitos de segunda ordem. Por exemplo, já que o RAID 1 armazena duas cópias do dado, uma otimização comum é efetuar a leitura a partir do disco que levará menos tempo para chegar até o dado.

Outros dois aspectos que podem ser considerados como fator de comparação são a sobrecarga de armazenamento, ou seja, o espaço adicional necessário para armazenar as informações redundantes, e o grau de concorrência máximo, expressos na Tabela 3.2, onde *N* indica o número de discos de dados da matriz. O grau de concorrência máximo para os níveis 1 e 5 é *N* pois essas matrizes podem suportar até *N* leituras concorrentes, mas as escritas envolvem múltiplas operações de E/S, e isso reduz a concorrência máxima suportada.

	<i>Sobrecarga de Armazenamento (%)</i>	<i>Máximo grau de concorrência</i>
RAID 0	0	N
RAID 1	100	N
RAID 3	100/G	1
RAID 5	100/G	N

Tabela 3.2 Sobrecarga de armazenamento e grau máximo de concorrência

A Figura 3.13 mostra os valores da Tabela 3.1 variando-se o tamanho do grupo de paridade. O desempenho/custo de um RAID 1 é equivalente ao desempenho/custo de um RAID 5 quando o tamanho do grupo de paridade é igual a 2. O desempenho/custo de um RAID 3 é sempre menor ou igual ao desempenho/custo de um RAID 5. Isso é esperado, uma vez que o RAID 3 é uma subclasse do RAID 5 gerada a partir da restrição do tamanho da unidade de fragmentação de forma que todas as requisições acessem exatamente um fragmento de paridade do dado. Uma vez que o arranjo de um RAID 5 não está sujeita a tal restrição, o desempenho/custo de um RAID 5 nunca pode ser inferior ao desempenho/custo de um RAID 3. É importante frisar que essas observações de desempenho/custo aplicam-se apenas a modelos abstratos de matrizes de discos. Na realidade, uma implementação específica de RAID 3 pode ter um desempenho/custo melhor que uma implementação específica de RAID 5.

Figura 3.13 *Throughput* máximo por dólar versus tamanho do grupo

A questão de qual nível RAID escolher é geralmente expressa em função de questões sobre o tamanho do grupo de paridade e a unidade de fragmentação. Se um grupo de paridade de tamanho dois é indicado, então a melhor escolha é o espelhamento (RAID 1). Se for necessária uma unidade de

fragmentação bem pequena, então um RAID 3 é suficiente. A Figura 3.14 mostra os mesmos valores da Tabela 3.1, agora agrupados por nível RAID, mostrando os valores para grandes leituras (GL), grandes escritas (GE), pequenas leituras (PL) e pequenas escritas (PE), sempre em relação ao RAID 0.

Figura 3.14 Comparação de desempenho entre RAIDs 3, 5 e 6

Com base nos gráficos e na utilização de um grupo de paridade grande, é possível concluir que RAID 3 é indicado para ambientes onde as escritas e as leituras são efetuadas em grandes volumes de dados de cada vez. Ambientes onde ocorrem pequenas leituras ou escritas, o seu desempenho cai com o aumento do grupo de paridade.

Tanto RAID 5 quanto RAID 6 são extremamente versáteis, adaptando-se a quase todas as situações, exceto aquelas geradas por ambientes onde ocorrem, com frequência, pequenas escritas. Nesses ambientes, o desempenho relativo ao RAID 0 cai com o aumento do tamanho do grupo de paridade, mas estabiliza em 25% e 20% a partir do 4º e 6º discos nos RAIDs 5 e 6 respectivamente (Chen et al., 1990). A seção seguinte faz algumas considerações sobre os impactos das operações *RMW* que ocorrem durante as gravações em RAID 5 e algumas técnicas utilizadas para minimizá-los.

Como pôde ser observado através da Figura 3.14, o RAID 5 seria a melhor alternativa, tanto para ambientes que possuem escritas e leituras de grandes volumes de dados, como os ambientes que apresentam escritas/leituras de pequenos volumes de dados. A única ressalva é o desempenho para pequenas escritas no RAID 5 reduz com o aumento do tamanho do grupo de paridade. O problema e algumas possíveis soluções são tratados na seção seguinte.

3.4 Aspectos de desempenho do RAID 5 (Chen et al., 1994)

O RAID 5 apresenta o melhor desempenho em quase todas as situações. Sua maior desvantagem é a execução de pequenas escritas. Esse baixo desempenho é causado pela forma como essas escritas são executadas.

No momento em que se torna necessária a gravação de uma pequena quantidade de informações no RAID 5, ocorrem quatro operações: leitura da paridade antiga, leitura dos dados antigos, escrita do novo dado e da nova paridade. Isto dobra o tempo de resposta de escritas e divide por quatro o *throughput*. Nas opções de RAID que usam espelhamento, como é o caso do RAID 1, as pequenas escritas geram apenas duas operações de E/S, o que causa um aumento muito pequeno no tempo de resposta e uma redução de 50% no *throughput*.

Operações *RMW* não são a única forma de alterar dados em RAID 5, mas ela é usada em situações onde o dado a ser alterado atinge até $N/2$ discos da matriz. Se for necessário alterar entre $(N/2)$ e $(N-1)$ discos, a solução que apresenta melhor desempenho é a conhecida por **escrita de reconstrução** (*reconstruct write*), onde o dado novo é escrito, os dados que não foram alterados são lidos, a nova paridade é calculada e escrita no local apropriado. Em situações onde uma alteração atinge todos os discos da matriz, a solução ideal é escrever os dados novos e, simplesmente, sobre-escrever a paridade antiga com o novo valor, calculado a partir dos dados que acabaram de ser escritos.

Nos tópicos seguintes serão descritas três técnicas para melhorar o desempenho de pequenas escritas que utilizam *RMW* em RAID 5: *buffering* e *cache*, *floating parity* e *parity logging*.

3.4.1 Buffering e Cache

A utilização das técnicas de *buffering* e *cache*, comuns em sistemas de E/S, podem ser particularmente eficazes em matrizes de disco.

O *buffering* de escrita, também chamado de escrita assíncrona, confirma a escrita do usuário antes de a escrita ser efetivada no disco. Esta técnica reduz o tempo de resposta experimentado pelo usuário sob cargas de trabalho baixas ou moderadas. Dessa forma, o tempo de resposta se torna independente do sistema de disco, e o RAID 5 pode oferecer o mesmo tempo de resposta dos outros. O problema surge numa possível falha do sistema com os dados no *buffer* (em memória). Isso pode ser minimizado com o uso de memória não-volátil para o *buffer*.

O uso da técnica de escrita assíncrona permite ainda que novas escritas sejam efetuadas em dados ainda no *buffer*, eliminando a necessidade de escrever um dado e, logo em seguida, sobre-

escrevê-lo. Essa técnica não é muito eficiente para cargas altas, pois o espaço do *buffer* de escrita se encherá mais rapidamente do que se esvaziará, e os tempos de resposta do RAID 5 voltarão a ficar altos.

Essa técnica pode ser estendida para agrupar escritas sequenciais. Escrevendo grandes volumes de dados de uma vez, pequenas escritas podem se tornar escritas de fragmentos completos, reduzindo sensivelmente o problema de pequenas escritas do RAID 5.

O *cache* de leitura é normalmente utilizado em sistemas de disco para melhorar o tempo de resposta e o *throughput* para leitura. Esse *cache* pode ser explorado, no RAID 5, para armazenar o dado antigo necessário ao cálculo da paridade, reduzindo assim, o número de operações de E/S de quatro para três. Isso é muito comum, por exemplo, em ambientes de processamento de transações, onde os registros são atualizados freqüentemente através da leitura dos valores antigos, alteração, e escrita dos novos dados no mesmo local.

Se for possível armazenar em *cache* a paridade que acabou de ser escrita, a leitura da paridade antiga pode ser eliminada, reduzindo-se o número de operações de E/S de três para duas. Uma vez que a paridade é calculada ao longo de vários setores de disco logicamente consecutivos, o armazenamento em *cache* da paridade explora tanto a localidade temporal quanto a espacial.

3.4.2 Floating Parity

A forma de organização de paridade no RAID 5 proposta por Menon e Kasson (Menon93b *apud* Chen et al., 1994) reduz o número de acessos *RMW* de atualização da paridade para pouco mais de um acesso em média. A idéia é agrupar as paridades em cilindros, cada um contendo uma trilha com blocos livres. Toda vez que um bloco de paridade precisa ser atualizado, o novo bloco de paridade pode ser escrito no bloco livre rotacionalmente mais próximo do bloco antigo. Menon e Kasson mostram que para discos com 16 trilhas por cilindro, o próximo bloco livre é o bloco imediatamente seguinte ao que acabou de ser lido com a paridade em 65% dos casos. Além disso, o número médio de blocos que devem ser pulados para chegar ao bloco livre mais próximo é pequeno, entre 0,7 e 0,8. Dessa forma, a escrita do novo bloco de paridade pode ocorrer imediatamente após a leitura do bloco de paridade antigo, fazendo com que todo o acesso *RMW* seja apenas alguns poucos milissegundos mais longo que um acesso de leitura.

Para eficiência da implementação de *floating parity*, é necessário que o diretório de localização dos blocos livres e dos blocos de paridade estejam em memória principal. Menon e Kasson estimam em 1Mb de memória principal para alocar o diretório de cada matriz de quatro a dez discos de 500Mb. Para que seja possível explorar o bloco não alocado logo após o de paridade que está sendo

lido, o dado deve ser modificado e a cabeça do disco movida para a trilha que contém o bloco não alocado antes de o disco rotacionar para um *gap* inter-setores. Devido a essas limitações, e por apenas a controladora de disco possuir o conhecimento exato de sua própria geometria, a técnica de *floating parity* é mais factível de implementação na própria controladora

3.4.3 Parity Logging

Stodolsky et al. (Stodolsky et al., 1993) propõe uma abordagem chamada *parity logging* para reduzir as penalidades de pequenas escritas em RAID 5. A idéia é atrasar a leitura da paridade antiga e a escrita da paridade nova. Ao invés de atualizar imediatamente a paridade, uma imagem atualizada – a diferença entre as paridades nova e antiga – é temporariamente escrita em um log. O atraso da atualização permite que as paridades sejam agrupadas em blocos contíguos longos que podem ser atualizados de forma mais eficientemente.

Este atraso ocorre em dois lugares. Primeiro, a imagem atualizada é armazenada temporariamente em memória não volátil. Quando essa memória, que pode ser de alguns poucos kilobytes, enche, a imagem de atualização da paridade é escrita no log. Quando o log enche, a imagem de atualização da paridade é lida na memória junto com a paridade antiga e é aplicada sobre a paridade antiga. A paridade nova resultante é então escrita no disco. As unidades de dados transferidas de e para o disco são maiores neste esquema, e por isso, conseguem ser mais eficientes, como visto na seção 2.1. *Parity logging* consegue reduzir o número de acessos de quatro para dois, na média. O custo do uso de *parity logging* repousa sobre a memória principal adicional utilizada para o armazenamento temporário das imagens de atualização, sobre o espaço extra de disco necessário para o log das imagens de atualização, e a memória adicional utilizada para gerar a nova paridade com o auxílio da paridade antiga e das imagens de atualização.

3.5 Tamanho da unidade de fragmentação

A escolha correta do tamanho para a unidade de fragmentação tem um impacto considerável no desempenho geral de um RAID. Chen e Lee (Chen et al., 1995) afirmam que o tamanho da unidade de fragmentação ideal (aquela que conduz ao melhor desempenho) para um RAID 5 varia com o número de discos da matriz. Para ambientes onde ocorrem predominantemente leituras, o tamanho da unidade fragmentação (*SU*) varia inversamente com o número de discos, enquanto em ambientes onde as escritas predominam, varia diretamente como número de discos. Eles afirmam, também, que pode ser tomado como tamanho de unidade de fragmentação ideal, para ambientes onde não há como prever a carga de trabalho:

$$SU \propto \frac{avSettle \Delta dtr}{2}$$

Equação 3.3 Unidade de fragmentação (Chen & Lee)

Onde $avSettle$ é o tempo médio de posicionamento (busca e atraso rotacional médio), e dtr é a taxa de transferência de dados do disco. Chen e Patterson (Chen et al., 1990) fizeram um estudo para a definição do tamanho da unidade de fragmentação ideal para qualquer esquema de fragmentação. Através do estudo de Chen e Patterson é possível escolher um tamanho de unidade de fragmentação que garante, pelo menos, 95% do *throughput* máximo para requisições de qualquer tamanho, desde que se conheça o nível de concorrência (número de requisições simultâneas sendo manipuladas pela matriz). O valor é obtido através da equação:

$$SU \mid 9,8Kb \Delta (DC \ 4 \ 1) \ 2 \ seSize$$

Equação 3.4 Unidade de fragmentação (Chen & Patterson - 1)

Onde SU é dado em Kb, DC representa o nível de concorrência, e $seSize$ indica o tamanho do setor no disco, em Kb. A constante 9,8 – chamada de fator de desvio (*slope factor*) – é um valor que pode ser ajustado com base nos parâmetros do disco através da equação:

$$slope \mid S \Delta avSettle \Delta dtr$$

Equação 3.5 Fator de desvio

Onde S é o coeficiente de desvio de concorrência (*concurrency-slope coefficient*) e, nos estudos de Chen e Patterson, variam de 0,22 a 0,26. Assim, substituindo-se a Equação 3.5 na Equação 3.4, tem-se uma equação que permite escolher o melhor tamanho de unidade de fragmentação com base nas características do disco e o nível de concorrência na matriz:

$$SU \mid S \Delta avSettle \Delta dtr \Delta (DC \ 4 \ 1) \ 2 \ seSize$$

Equação 3.6 Unidade de fragmentação (Chen & Patterson - 2)

Com o auxílio da Equação 3.6 é possível perceber que, se o nível de concorrência for 1, o tamanho ideal da unidade de fragmentação é o tamanho do setor, de forma que o processo que está emitindo requisições seja capaz de utilizar todos os discos. O tamanho da unidade de fragmentação cresce com o aumento do nível de concorrência para reduzir gradualmente a probabilidade de que um acesso qualquer em particular utilize mais do que um disco, permitindo que o maior número possível de processos tenha acesso ao disco.

Tal como Chen e Lee (Chen et al., 1995), Chen e Patterson também estudaram o melhor tamanho de unidade de fragmentação quando não há informação alguma disponível sobre a carga de trabalho, estando representado pela equação seguinte:

$$SU \mid Z \Delta avSettle \Delta dtr$$

Equação 3.7 Unidade de fragmentação (Chen & Patterson - 3)

Onde Z representa o coeficiente de conhecimento zero (*zero-knowledge coefficient*) e varia de 0,63 a 0,72, podendo ser assumido, para a maioria dos casos, o valor de 0,66.

Merchant e Yu (Merchant92 *apud* Courtright II et al., 1996) propõe uma abordagem de fragmentação dupla. Em ambiente de bancos de dados, há basicamente dois padrões de acesso aos dados (carga de trabalho): transações e consultas *ad hoc*. As transações geram pequenas requisições aleatórias ao longo dos discos, enquanto que as consultas *ad hoc*, geralmente, acessam grandes quantidades de dados. Para resolver o problema da escolha da unidade de fragmentação, eles propõe a utilização de uma matriz espelhada e fragmentada (RAID 10) onde, uma das cópias atenderia apenas as requisições das transações e teria uma unidade de fragmentação menor (4Kb no estudo feito por eles), e a outra cópia, responsável por atender as consultas *ad hoc*, teria uma unidade de fragmentação maior (32Kb). A organização proposta foi avaliada por simulação e soluções analíticas dos modelos, e os resultados provaram que essa abordagem gera ganhos significativos de desempenho.

3.6 Comparação de confiabilidade

A confiabilidade é uma métrica tão importante para alguns sistemas quanto o desempenho e o custo, e ela é, talvez, a razão mais importante para o sucesso dos RAIDs. Ao longo desta sessão, a abordagem de confiabilidade adotada será a teórica onde, para o cálculo do MTBF, são considerados apenas os fatores intrínsecos dos equipamentos, desprezando-se os fatores utilizados no cálculo do MTBF Operacional.

3.6.1 Confiabilidade básica

A redundância nas matrizes de discos é motivada pela necessidade de superar as falhas de disco. Quando apenas falhas independentes de disco são consideradas, um esquema de paridade simples funciona muito bem. Patterson (Patterson et al., 1988) mostra que o MTBF de um RAID 5 com N discos e um grupo de paridade de tamanho G pode ser expressa pela fórmula:

$$MTBF_{RAID5} \approx \frac{MTTF(disco)^2}{N \Delta(G-1) \Delta MTTR(disco)}$$

Equação 3.8 Tempo médio entre falhas para RAID 5

Onde $MTTF(disco)$ é o tempo médio para falha de um único disco, $MTTR(disco)$ é o tempo médio para reparar um disco, N é o número total de discos na matriz, e G é o tamanho do grupo de paridade. Se forem tomados como exemplo 100 discos Seagate Barracuda 4LP ST34371WD (Seagate, 1997), cada um com MTTF de 1.000.000 horas, e MTTR de 1 hora, e eles forem organizados em uma matriz com tamanho do grupo de paridade médio de 16, então o MTBF do sistema chega a 76.103

anos e 6 meses aproximadamente. Tempos médios de falha desta magnitude reduzem as chances de falha em qualquer período de tempo informado.

Para RAIDs com dois discos redundantes por grupo de paridade, como o RAID 6, o tempo médio de falha é dado por:

$$MTBF_{RAID6} \approx \frac{MTTF(disco)^3}{N \Delta(G 4 1) \Delta(G 4 2) \Delta MTTR^2(disco)}$$

Equação 3.9 Tempo médio entre falhas para RAID 6

Se forem tomados os mesmos valores como parâmetros de confiabilidade do exemplo anterior, tem-se um MTBF astronômico de mais de 5,5 bilhões de anos.

Essa é uma imagem ideal, é claro, mas mostra a confiabilidade potencial de uma matriz de discos. As próximas seções ilustram de forma mais realista a confiabilidade de uma matriz de discos entrelaçada em nível de bloco considerando fatores como falhas do sistema, erros de bit incorrigíveis e falhas de disco correlatas que podem afetar drasticamente a confiabilidade das matrizes.

3.6.2 Falha de sistema e inconsistência de paridade

Falha de sistema pode ser considerado, nesta seção, qualquer evento do tipo falta de luz, erro de operador e falha de *hardware* ou de *software* que possa interromper uma operação de E/S em uma matriz de discos. Tais falhas podem interromper operações de escrita, resultando em estados onde o dado é atualizado e a paridade não, ou vice-versa. Em ambos os casos, a paridade está inconsistente e não pode ser usada no caso de uma falha de disco. Técnicas como *hardware* e fornecimento de energia redundante podem ser usados para reduzir a incidência dessas falhas, mas nenhuma técnica pode evitar a falha em 100% dos casos.

Na realidade, falhas de sistema podem ser muito piores que falhas de disco por duas razões. Primeiro, elas podem ocorrer com uma frequência muito maior que falhas de disco. Segundo, uma falha de sistema em matriz de discos usando redundância P+Q é equivalente a uma falha dupla de disco já que as informações “P” e “Q” estão inconsistentes. Para evitar a perda de paridade em falhas de sistema, informações suficientes para reconstruir a paridade devem estar armazenadas em memória não-volátil antes da execução de toda e qualquer operação de escrita. Implementações de RAID por *hardware* podem implementar essas soluções com o uso de memória RAM não-volátil. Implementações em *software* geralmente não podem implementar essas proteções sem um significativo sacrifício de desempenho.

3.6.3 Erros de bit incorrigíveis

Apesar dos discos rígidos apresentarem uma confiabilidade razoável, eles ocasionalmente falham na leitura ou escrita de poucos bits do dado. Em 1993, a maioria dos fabricantes reportava BER (*Bit Error Rate*) de um em 10^{14} bits lidos. Infelizmente, a interpretação correta de taxa de erro de bit não é clara. É o ato de ler o disco que realmente gera um erro, ou o erro ocorre na escrita mas é detectado na leitura? Nesta monografia será adotada a interpretação de Chen (Chen et al., 1994), onde os erros são gerados na escrita e detectados na leitura. A ocorrência desses erros pode ser atribuída, por exemplo, ao envelhecimento da mídia magnética.

A ramificação primária de um erro de bit incorrigível é sentida no momento em que é necessária a reconstrução de um disco falho a partir da leitura dos discos que não falharam. Na reconstrução de uma matriz de 100Gb, por exemplo, é necessário sucesso na leitura de aproximadamente 200 milhões de setores de disco (Chen et al., 1994). Uma taxa de erro de um em 10^{14} bits implica que um setor de 512 bytes em 24 bilhões pode não ser lido corretamente. Assim, se for assumido que as probabilidades de sucesso de leitura de setor são independentes, a probabilidade

de leitura com sucesso dos 200 milhões de setores é $\left(1 - \frac{1}{2,4 \times 10^{10}}\right)^{200.000.000} \approx 99,2\%$. Isto quer

dizer que, em média, 0,8% das falhas de disco pode resultar na perda de dados em decorrência de um erro de bit incorrigível. Assim, percebe-se que os erros de bit incorrigíveis podem ser um fator significativo no projeto de uma matriz de disco grande e que pretende ser altamente confiável. Os erros de bit incorrigíveis podem ser chamados também de falhas latentes de setor (*latent sector failure*) (Courtright II et al., 1996)

3.6.4 Falhas de disco correlatas

O modelo mais simples de confiabilidade de matriz de discos, proposto por Patterson (Patterson et al., 1988), assume que todas as falhas de disco são independentes para o cálculo do tempo médio para perda de dados. Isso gera valores estimados muito altos, chegando a casa dos milhões de anos. Na realidade, fatores ambientais e de fabricação irão causar, geralmente, falhas de disco correlatas. Sobrecargas de energia, faltas de luz e o simples ato de ligar e desligar os discos pode aplicar um desgaste simultâneo nos componentes elétricos de todos os discos. Discos também compartilham *hardware* de suporte em comum. Se esse *hardware* falhar, falhas simultâneas em múltiplos disco podem acontecer.

Deixando de lado os fatores ambientais, os próprios discos possuem certas falhas correlatas. Por exemplo, os discos tendem a falhar mais quando muito novos ou muito velhos. As falhas enquanto novos normalmente são causadas por defeitos que não foram detectados durante os testes de

fabricação. As falhas nos discos mais velhos são causadas pelo desgaste natural. Uma falha na linha de produção também pode gerar um lote de discos que pode falhar mais ou menos ao mesmo tempo. As falhas de disco correlatas reduzem a confiabilidade da matriz de discos em muito, pois podem introduzir uma segunda falha, antes que o sistema se recupere da primeira falha.

3.6.5 Confiabilidade revista

Nesta seção, serão apresentadas as estatísticas para o tempo médio para perda de dados (MTTDL), através da incorporação dos fatores apresentados nas seções anteriores. Os novos modos de falha implicam que há agora três formas, relativamente comuns, de se perder dados em uma matriz de discos entrelaçada em nível de bloco e protegido por paridade: falha dupla de disco, falha de sistema seguida por uma falha de disco, e uma falha de disco seguida de um erro de bit incorrigível durante a reconstrução.

Como já foi discutido na seção 3.6.2, a proteção de uma falha de sistema seguida por uma falha de disco pode ser implementada com a ajuda de memória não-volátil na maioria das implementações de matrizes de discos por *hardware*, mas essa proteção não é tão simples em soluções por *software*. Para construir um modelo simples de falhas de disco correlatas, assumir-se-á que cada falha de disco sucessiva é dez vezes mais provável que a anterior, até que o disco falho seja reconstruído. A Tabela 3.3 mostra os parâmetros de confiabilidade que serão usados para os cálculos estimativos de confiabilidade nesta seção. As estimativas de confiabilidade serão computadas com uma capacidade constante de 100 discos, consistindo de grupos de paridade independentes de 16 discos.

<i>Característica</i>	<i>Valor</i>
Capacidade total	100 discos (500Gb)
Tamanho do disco	5Gb
Tamanho do setor	512 bytes
BER	1 em 10^{14} bits
	1 em $2,4 \times 10^{10}$ setores
p(disco)	99,96%
A probabilidade de ler todos os setores de um disco (derivada do tamanho do disco, tamanho do setor e BER)	
Tamanho do grupo de paridade	16
MTTF(disco)	200.000 horas
MTTF(disco2)	20.000 horas
MTTF(disco3)	2.000 horas
MTTR(disco)	1 hora
MTTF(sistema)	1 mês
MTTR(sistema)	1 hora

Tabela 3.3 Parâmetros de confiabilidade para cálculos estimativos

A Tabela 3.3, elaborada por Chen (Chen et al., 1994), mostra as métricas de confiabilidade para uma matriz RAID 5. Pode-se observar pela tabela que a frequência das três combinações de falhas diferem em, no máximo, uma razão de 10 e, por isso, não podem ser ignoradas na determinação da confiabilidade da matriz. Assim, percebe-se que é difícil aumentar a confiabilidade geral do sistema

sem melhorar a confiabilidade de vários componentes do mesmo: um disco mais confiável reduziria muito a frequência de falhas duplas de disco, mas sua proteção contra as outras duas combinações de falhas seria menos pronunciada. Frequências, tanto de falha do sistema como de erro de bit, devem ser reduzidas antes de uma melhora significativa da confiabilidade geral do sistema. Mesmo com um MTDDL de 285 anos, existe 3,4% de chance de uma perda de dados nos primeiros 10 anos.

<i>Falha</i>	<i>MTDDL</i>	<i>Probabilidade de perda de dados num período de 10 anos</i>
Falha dupla de disco	$\frac{MTTF(disco) \Delta MTTF(disco2)}{N \Delta (G 4 1) \Delta MTTR(disco)}$ 285 anos	3,4%
Falha de sistema + Falha de disco	$\frac{MTTF(sistema) \Delta MTTF(disco)}{N \Delta MTTR(sistema)}$ 154 anos	6,3%
Falha de disco + Erro de bit	$\frac{MTTF(disco)}{N \Delta (1 4 p(disco))^{G41}}$ 36 anos	24,4%
RAID por <i>software</i>	Soma harmônica dos valores 26 anos	31,6%
RAID por <i>hardware</i> (NVRAM)	Soma harmônica, excluindo a falha de sistema + falha de disco 32 anos	26,8%

Tabela 3.4 Métricas de confiabilidade para RAID 5

<i>Falha</i>	<i>MTDDL</i>	<i>Probabilidade de perda de dados num período de 10 anos</i>
Falha tripla de disco	$\frac{MTTF(disco) \Delta MTTF(disco2) \Delta MTTF(disco3)}{N \Delta (G 4 1) \Delta (G 4 2) \Delta MTTR(disco)^2}$ 38.052 anos	0,03%
Falha de sistema + Falha de disco	$\frac{MTTF(sistema) \Delta MTTF(disco)}{N \Delta MTTR(sistema)}$ 144 anos	7,7%
Falha dupla de disco + Erro de bit	$\frac{MTTF(disco) \Delta MTTF(disco2)}{N \Delta (G 4 1) \Delta (1 4 (1 4 p(disco))^{G42}) \Delta MTTR(disco)}$ 47.697 anos	0,02%
RAID por <i>software</i>	Soma harmônica dos valores 143 anos	6,8%
RAID por <i>hardware</i> (NVRAM)	Soma harmônica, excluindo a falha de sistema + falha de disco 21.166 anos	0,05%

Tabela 3.5 Métricas de confiabilidade para RAID 6

A Tabela 3.5 (Chen et al., 1994) mostra as métricas de confiabilidade para uma matriz de discos com redundância P+Q, como o RAID 6. Como é possível ver, falhas de sistema são o calcanhar de Aquiles de esquemas de redundância P+Q.

Uma vez que uma falha de sistema invalida tanto a informação P quanto a informação Q, seu efeito é similar a uma falha dupla de disco. Por isso, a menos que o sistema forneça proteção contra falhas do sistema, como assumido no cálculo da confiabilidade para RAIDs em *hardware*, a redundância P+Q não oferece uma vantagem significativa sobre as matrizes de disco protegidas por paridade. Em geral, esquemas de redundância P+Q são mais úteis contra erros de bit incorrigíveis que ocorrem durante a reconstrução e contra falhas de disco correlatas múltiplas.

3.7 Considerações finais

Nesse ponto, pode-se observar que falhas no sistema e erros de bit incorrigíveis podem reduzir significativamente a confiabilidade de uma matriz. Observou-se, também, que matrizes com redundância P+Q são muito efetivas na proteção contra falha dupla de disco e erros de bit incorrigíveis, mas são suscetíveis a falhas de sistema. Com o intuito de obter todas as vantagens da redundância P+Q, deve ser utilizado armazenamento não-volátil para a proteção contra falhas de sistema.

Os RAIDs oferecem uma opção atrativa em relação ao custo para satisfazer o crescimento exponencial na velocidade dos processadores e memória. Com vantagens no desempenho/custo, confiabilidade, consumo de energia e crescimento modular, os RAIDs vêm substituindo os SLEDs (Discos Únicos, Grandes e Caros – *Single Large Expensive Disk*) nos sistemas de E/S que precisam de alta desempenho.

Uma das melhores formas de se estimar o desempenho de um RAID em uma determinada situação de carga de trabalho e características de discos é simulá-lo. O capítulo seguinte apresenta os principais conceitos de simulação.

Capítulo 4 Simulação

“Atualmente, a simulação é uma poderosa técnica para solução de problemas. Sua origem remonta à teoria de amostragem estatística e análise de sistemas físico-probabilísticos complexos. A abordagem comum nos dois casos é o uso de número aleatórios e amostragem aleatória para aproximar um resultado ou solução. Uma das mais famosas aplicações de amostragem aleatória ocorreu durante a Segunda Guerra Mundial, quando simulação foi utilizada para estudar difusão aleatória de neutrons no processo de desenvolvimento da bomba atômica. Já que a pesquisa era confidencial, lhe foi dado um nome-código próprio para processos que tratam de eventos aleatórios: Monte Carlo, em referência ao mais famoso cassino do mundo. O nome persistiu e chegou a ser utilizado para fazer referência a qualquer tentativa de simulação. Mas o termo métodos de Monte Carlo se refere, hoje, apenas a um ramo da matemática experimental que trata de experimentos com número aleatórios” (Hoover et al., 1990)

A resolução de qualquer problema, ou sistema, independente da forma de resolução, só é possível se o problema estiver bem definido. Para que a definição do problema seja bem feita, é necessário o bom conhecimento do problema. Um sistema pode ser descrito (abstraído) de várias formas, dependendo da parte do problema a ser resolvida.

Figura 4.1 Processo de abstração e inferência

Desta forma, **modelo** é uma descrição do sistema (Soares, 1990). O modelo é gerado a partir de uma abstração do problema, onde são representadas as partes pertinentes no momento. O nível de complexidade do modelo é diretamente proporcional à abstração efetuada, ou seja, ao número e a quais características foram utilizadas para a abstração. Se o modelo estiver muito complexo, é possível simplificá-lo, desde que o modelo simplificado continue representando bem o problema. Assim, “a modelagem de sistemas pode ser vista como a simplificação que descarta as características julgadas

irrelevantes” (Orlandi, 1995). A Figura 4.1 (adaptada de Orlandi, 1995) mostra esse processo de abstração.

Os modelos podem ser construídos de várias formas: na forma de ícones, por analogia ou de forma simbólica (Hoover et al., 1990). Modelos na forma de ícones se parecem fisicamente com o sistema real, da mesma forma que uma maquete se parece com um prédio ou uma miniatura se parece com um carro. Modelos análogos se parecem com o sistema real apenas no seu comportamento, como o comportamento de um protótipo no túnel de vento modela o comportamento de um carro na estrada. Por fim, os modelos simbólicos não tem relação nem física, nem análoga, mas sim uma relação lógica com o sistema real.

4.1 Classificação dos modelos

Há vários tipos de modelos simbólicos, que podem ser classificados com **discretos** ou **contínuos** (Hoover et al., 1990; Soares, 1990). Essa classificação diz respeito às variáveis do modelo. As variáveis contínuas podem assumir qualquer valor real, enquanto que as variáveis discretas só podem assumir determinados valores específicos. Se as alterações no modelo ocorrem de forma contínua com a variação do tempo, diz-se que o modelo é contínuo, caso contrário, se as alterações ocorrem em pontos discretos de tempo, diz-se que é discreto. É importante notar que modelos discretos podem tratar o tempo como uma variável contínua, entretanto, as mudanças no sistema não ocorrem continuamente a uma determinada taxa, mas em tempos de eventos particulares e discretos. O processo de resolver modelos desse tipo por simulação tem o nome de **simulação discreta**.

Os modelos simbólicos elaborados para representar uma situação podem ser resolvidos, também, por **métodos analíticos**.

Os modelos simbólicos podem ser classificados, também, em **probabilístico** ou **determinístico** (Hoover et al., 1990). Se houver variáveis aleatórias definidas por funções de probabilidade apropriadas, o modelo é probabilístico, caso contrário, se só houverem variáveis cujos valores podem ser definidos com certeza, o modelo é determinístico. Todos os sistemas envolvendo seres humanos e muitos sistemas envolvendo apenas máquinas necessitam de variáveis aleatórias. Uma linha de montagem totalmente automatizada por robôs pode ser descrita de forma adequada através de um modelo determinístico.

Hoover (Hoover et al., 1990) diz, ainda, que “**simulação** é o processo de projetar um modelo lógico ou matemático de um sistema real e conduzir experimentos em computador com o modelo para descrever, explicar e prever o comportamento do sistema real”. Pelo fato de ser de custo relativamente

baixo, flexível e de fácil utilização, a simulação tem atuado nas mais diversas áreas do conhecimento humano.

4.2 Simulação Discreta

Nesta dissertação serão abordados, apenas, os aspectos pertinentes aos modelos discretos, uma vez que os sistemas computacionais, que são objeto de interesse, são melhor representados por modelos e consequentemente simulação discreta.

Segundo Soares (Soares, 1990), o objetivo de um modelo para simulação discreta é reproduzir as atividades dos objetos envolvidos e, a partir daí, conhecer algo sobre o comportamento e desempenho do sistema. Para que isso seja possível, é necessário definir os estados (valores dos atributos dos objetos) do sistema e elaborar atividades que façam a transição de estados.

Na simulação discreta, a alteração de estados só ocorre em tempos discretos. Já que o estado do sistema permanece inalterado entre os tempos de eventos, é necessária avançar o **tempo simulado** para obter a descrição completa do estado do sistema.

O processo de solução de um modelo por simulação discreta pode ser formulado de três formas:

- ##Pela descrição das mudanças nos estados, que podem ocorrer a cada tempo de evento;
- ##Pela descrição das atividades nas quais os objetos estão envolvidos;
- ##Pela descrição do processo através do qual os objetos do processo se relacionam.

A Figura 4.2 mostra relação entre evento, atividade e processo. No exemplo, o início e o término da carga do caminhão são alguns dos eventos. As atividades são carregar o caminhão, viajar e descarregar. O processo é transportar uma determinada carga.

Figura 4.2 Relação entre evento, atividade e processo

Na simulação orientada a evento, a idéia básica é avançar na escala de tempo até a ocorrência do próximo evento e então, dependendo do evento, modificar o estado do sistema e, possivelmente,

escalonar novos eventos. O escalonamento dos novos eventos pode ser probabilístico ou determinístico, dependendo do modelo adotado.

Se for adotada a orientação por atividade, é necessário descrever as atividades que envolvem os objetos do sistema e declarar as condições de início e término de uma atividade. Os eventos indicadores de início e fim de atividade não são escalonados, mas sim indicados a partir das condições definidas para cada atividade. A grande vantagem desta abordagem é que ela é muito mais simples de imaginar e de formalizar (Trick, 1996). Na maioria das vezes em que as pessoas são solicitadas a gerar eventos para simulação, elas escolhem eventos que necessitam de busca por atividade. A maior desvantagem é sua ineficiência, uma vez que é difícil fazer o programa de simulação sempre procurar por atividades sem reduzir a velocidade da simulação.

A simulação orientada a processo permite a modelagem do sistema através da definição de processos interativos como abstrações das entidades ativas do sistemas (SC86 *apud* Orlandi, 1995), dessa forma, é possível a definição de uma lista de eventos que devem ocorrer para que um determinado processo ocorra. Do ponto de vista do modelador, o sistema é um pouco diferente: os eventos não tem sentido. Do ponto de vista do programa de simulação, entretanto, o sistema é o mesmo: eventos devem ser escalonados e o sistema atualizado. (Trick, 1996)

4.3 Ferramentas para simulação

Uma vez que o modelo tenha sido definido e a simulação tenha sido escolhida como forma de solução do modelo, resta decidir como simular o modelo. Algumas ferramentas são utilizadas para a solução de modelos através de simulação, entre elas, a utilização de linguagens de propósito geral, linguagens específicas para simulação, pacotes de simulação, extensões funcionais e os sistemas automáticos (Spolon, 1994; Oliveira, 1998). Os parágrafos seguintes fazem considerações sobre cada uma das ferramentas.

Em teoria, qualquer linguagem de programação poderia ser utilizada para o desenvolvimento de programas que resolvam modelos por simulação. Assim, a escolha da linguagem depende unicamente da familiaridade do modelador com a mesma. A principal vantagem dessa abordagem, é que o modelador não precisa aprender nenhuma linguagem nova. Uma vantagem adicional é a portabilidade (se houver um compilador disponível para a plataforma, o programa pode rodar na plataforma em questão). A principal desvantagem é a ausência total de estruturas de dados e procedimentos que dêem apoio à simulação, cabendo ao modelador criá-los e manipulá-los diretamente através da linguagem.

Outra alternativa é utilizar linguagens específicas para simulação. Essas linguagens são, intrinsicamente, orientadas a eventos, atividades ou processos, cabendo ao modelador escolher a linguagem com base na orientação dada ao modelo. A principal vantagem de utilizá-las como ferramenta de simulação é que as mesmas já possuem todas as estruturas e funções e/ou procedimentos necessários à execução da simulação. A principal desvantagem desta abordagem é que o modelador deverá aprender uma nova linguagem de programação que só será utilizada para resolver modelos por simulação. Como exemplo de linguagens, têm-se SLAM II, GASP e SIMAN orientadas a eventos; GPSS, Simula e RESQ orientadas a processos, e; CSL orientada a atividades.

Quando o modelo a ser resolvido estiver dentro do domínio de aplicação de algum pacote de simulação existente no mercado, esse é outra alternativa à solução do modelo. Os resultados obtidos com esses pacotes são extremamente satisfatórios. O problema dos pacotes de simulação está na falta de flexibilidade do *software* (mudanças no modelo simulado podem extrapolar o escopo da ferramenta), aliado ao fato de que se o modelo sair do domínio de aplicação do pacote, será necessário trocar de pacote ou, até mesmo, partir para a utilização de outra ferramenta. São exemplos de pacotes de simulação ARENA, CMF, SNAP e OPNET Modeler.

Outra forma de tentar solucionar o modelo através da simulação é a utilização de uma extensão funcional. As extensões funcionais são bibliotecas que podem ser incorporadas a uma linguagem de programação convencional (como C/C++, Pascal e Fortran, chamadas de linguagem hospedeira) e que incorporam primitivas (funções, procedimentos e estruturas de dados) que auxiliam na elaboração de um programa de simulação orientado a eventos, procedimentos ou atividades, conforme a extensão escolhida. A vantagem mais fácil de se observar é que o modelador continua a utilizar a linguagem de programação que já conhece, sendo necessário aprender, apenas, as primitivas da biblioteca. Uma desvantagem é a necessidade de ter conhecimento *a priori* sobre a orientação a ser dada à simulação (eventos, atividades ou processos). São exemplos de extensões funcionais o SMPL (MacDougall, 1987) e o SSS (Pollatschek, 1995), ambos para a linguagem C/C++.

Por fim, o último tipo de ferramenta considerado aqui são os sistemas automáticos, que afastam o modelador da tarefa de transformar um modelo em um programa de simulação. Nesses sistemas, o modelador representa graficamente o modelo através de alguma técnica conhecida pelo sistema como, por exemplo, redes de fila e *statecharts*, e parametriza-o através de uma interface gráfica. Uma vez gerado o modelo, e estando o mesmo parametrizado, o sistema gera o programa de simulação através de um gerador de aplicações. Tendo o programa de simulação em mãos, cabe ao modelador (ou ao sistema, se esse for capaz) executá-lo, coletar os resultados e analisá-los. O Laboratório de Sistemas Distribuídos e Programação Concorrente do ICMC-USP desenvolve um sistema automático de simulação conhecido como ASiA – Ambiente de Simulação Automático

(Spolon, 1994; Santana et al., 1996; Oliveira, 1998). Nele, os modelos são representados em redes de filas, e o programa de simulação é gerado em C com a utilização da extensão funcional SMPL.

SMPL e SSS são exemplos de extensões funcionais para a linguagem C/C++ e foram utilizadas na simulação desenvolvida neste trabalho. A próxima seção apresenta uma breve descrição e uma comparação entre elas.

4.4 SMPL x SSS

A extensão funcional SMPL foi proposta por M. H. MacDougall em 1987 (MacDougall, 1987) com o objetivo de ser uma forma de resolver modelos através de simulação. No Laboratório de Sistemas Distribuídos e Programação Concorrente do ICMC-USP, SMPL tem sido usado pelo ASiA para a geração de programas de simulação. A SSS (*Simulation Subroutine Set library*) foi proposta em 1995 por M. A. Pollatschek (Pollatschek, 1995).

SMPL possui três elementos básicos que são usadas para representar o modelo: facilidades, *tokens* e eventos. Cada **facilidade** representa um recurso (centro de serviço) do sistema que está sendo modelado e o conjunto das facilidades interconectadas representa um sistema. O SMPL oferece primitivas para definir, reservar e liberar as facilidades, bem como verificar seu estado. Os *tokens* representam as entidades ativas do sistema, como por exemplo uma requisição de disco. Através do SMPL é possível executar duas operações sobre os *tokens*: reserva de uma facilidade e o escalonamento de eventos. Se um *token* reserva uma facilidade que não tem servidores disponíveis, ou seja, o centro de serviço não pode atender o *token* no momento, esse *token* é enfileirado até que algum servidor fique disponível e possa atendê-lo. O único atributo que SMPL disponibiliza aos *tokens* é a prioridade, o que permite a preempção das facilidades. Os **eventos** são a representação da mudança de estado de qualquer entidade do sistema. Por exemplo, quando uma requisição de disco termina de ser atendida, são gerados dois eventos: o final da requisição e a liberação do disco. Um evento é individualizado pelo seu número, o tempo de ocorrência e a identificação do *token* relacionado com esse evento.

Em SSS, os elementos básicos são filas, entidades e eventos. As **filas** são similares às facilidades do SMPL. Ao contrário do SMPL, uma fila está associada a somente um servidor, não sendo possível definir centros de serviço com múltiplos servidores ou múltiplas filas. As **entidades** são equivalentes aos *tokens*, com uma diferença: elas podem ter qualquer número de atributos e esses atributos podem ser utilizados durante a execução da simulação para tomar decisões e para individualizar as entidades dentro do sistema. Por exemplo, se um determinado atributo diz que uma entidade é do tipo A, ela pode ser colocada em uma fila; se for do tipo B, vai para outra fila; se for de

outro tipo, vai para uma terceira fila. Os **eventos** são equivalentes aos eventos do SMPL e têm a mesma função: indicar mudanças de estado no sistema.

As duas extensões funcionais possuem rotinas para submissão e seleção de eventos, além de ferramentas para coleta de dados e análises estatísticas. Comparando as versões atuais de SMPL e SSS, as saídas oferecidas pela SSS são mais completas e incluem, além de valores médios, desvio padrão, valor máximo, mínimo e um histograma de distribuição. SMPL oferece estatísticas apenas para centros de serviço, enquanto SSS permite que sejam coletados dados para análises estatísticas sobre virtualmente qualquer coisa de interesse dentro do sistema: atributo das entidades, filas e eventos, por exemplo. É possível saber, com as facilidades oferecidas por SSS, tamanho mínimo e máximo de fila, por exemplo, aspecto que o SMPL não fornece diretamente através de seu relatório de simulação. SSS foi desenvolvida em C, para permitir portabilidade, tal como o SMPL. Ao contrário de SMPL, não se tem notícia de uma versão de SSS para ambientes UNIX ou, mesmo, uma versão de SSS para simulação distribuída, tal como o ParSMPL (Ulson, 1999).

Tanto os programas desenvolvidos com SMPL, como os desenvolvidos com SSS para a solução de modelos por simulação são caracterizados pela presença de um laço responsável pelo controle da simulação. Dentro desse laço, faz-se o processamento dos eventos e o escalonamento de *tokens*/entidades. Além disso, são coletados dados para análise estatística ao fim da simulação. A Figura 4.3 (adaptada de Ulson, 1999) mostra a estrutura de um programa de simulação escrito com SMPL e com SSS onde é possível ver a similaridade entre eles.

<pre>#include <smpl.h> main() { prepara variáveis de estado inicia variáveis smpl() facility() schedule() cause(&evento, &token) { evento 1: evento 2: evento n: } report() }</pre>	<pre>#include <sss.h> main() { prepara variáveis de estado inicia variáveis INIQUE() INISTA() CREATE() entidade = IDIC() evento = NEXTEV() { evento 1: evento 2: evento n: } SUMRY() }</pre>
--	---

Figura 4.3 Estrutura de programa de simulação em SMPL e em SSS

A transformação de um programa de simulação com SMPL em um com SSS é uma tarefa simples na qual o maior trabalho é trocar nomes de funções.

4.5 Considerações finais

A simulação tem sido uma ferramenta de grande valia no desenvolvimento de estudos de análise e avaliação de desempenho de sistemas computacionais. A flexibilidade oferecida pela simulação, aliada ao custo relativamente baixo dessa técnica, constituem o principal atrativo. O uso de sistemas de geração automática de programas de simulação e de pacotes de simulação (quando disponíveis para o domínio em questão) é uma realidade nos dias atuais, uma vez que esses sistemas afastam o modelador da tarefa de escrever o programa de simulação, permitindo que eles se concentrem em sua atividade fim: modelar os problemas.

Além das aplicações convencionais de simulação (solução de problemas), ela pode ser utilizada como ferramenta de auxílio ao aprendizado, como será discutido no próximo capítulo.

Capítulo 5

Ambientes de Aprendizado por Simulação

Neste capítulo abordam-se aspectos de inteligência computacional – os Intelligent Simulation and Learning Environments – pertinentes ao objetivo do projeto aqui discutido, como suas principais características e módulos componentes.

No Capítulo 3, sobre RAIDs, pôde-se observar que a área em questão apresenta uma taxonomia vasta, e que pequenas alterações de parâmetros podem gerar grandes variações de desempenho ou confiabilidade. Nem sempre o usuário domina a taxonomia e consegue entender porque certos resultados foram obtidos com determinados parâmetros.

Para que o usuário seja capaz de experimentar combinações de valores para os parâmetros, pode-se utilizar um simulador, mas quanto a taxonomia e o porquê de determinada alteração de parâmetro modificar o desempenho ou a confiabilidade, a alternativa é a presença de um tutor humano com conhecimento da área. O problema é que nem sempre esse tutor está disponível no momento em que o usuário precisa, restando, então, a alternativa de incluir o conhecimento sobre RAIDs no ambiente de simulação, e oferecê-lo ao usuário conforme a necessidade.

A forma escolhida para a oferta de conhecimento ao usuário deve considerar que o mesmo é ativo no processo, ou seja, está interagindo com o ambiente, alterando valores e suas necessidades de informação não seguem uma linha lógica: num momento, pode ser necessário saber por que uma unidade de disco é tão lenta, em outro momento, a informação necessária pode ser qual o impacto da escolha do interface no desempenho geral de uma unidade RAID e, em um terceiro momento, qual o impacto do valor escolhido como unidade de fragmentação no desempenho de uma unidade RAID 0.

Durante a simulação, o usuário pode ainda, formular algumas hipóteses e alterar valores e configurações. Os resultados obtidos a partir de uma hipótese podem ser analisados pelo usuário, fazendo com que o mesmo aprenda de forma construtivista. Aceita-se a abordagem construtivista como um caminho eficiente para adquirir conhecimento (Jong, 1991 *apud* Freire, 1998).

Desta forma, torna-se necessário um ambiente de simulação que ofereça instrução assistida por computador e tenha capacidade de lidar com o comportamento ativo do usuário. A computação, na área de Inteligência Computacional, oferece um ambiente de simulação exatamente com essas características, conhecido por *Intelligent Simulation and Learning Environment* (ISLE), que será apresentado ao longo dos tópicos deste capítulo.

5.1 *Intelligent Simulation and Learning Environment*

As ferramentas de CAI (*Computer Assisted Instruction*) e CAL (*Computer Assisted Learning*) vêm sendo desenvolvidas desde os anos 60, com a finalidade de oferecer instrução e aprendizado através dos computadores. O problema é que os sistemas CAI e CAL possuem uma programação linear, ou seja, são eles que decidem o que apresentar para o usuário, sem considerar o aproveitamento do mesmo. Assim, o usuário assume uma posição passiva.

Para suprir essa deficiência, a área de Inteligência Computacional interagiu com a psicologia (psicologia cognitiva³) e com a pedagogia (metodologias de ensino), gerando um novo conjunto de ferramentas para auxílio ao aprendizado chamado de *Intelligent Computer Assisted Instruction* ou ICAI. Um ICAI incorpora técnicas de IA que lhe permitem saber o quê deve ser ensinado, a quem ensinar e como isso deve ser feito (Freire, 1998). Essa última capacidade é obtida a partir das áreas de psicologia e pedagogia.

A principal diferença entre os CAIs e os ICAIs é que os últimos tratam o usuário como um elemento ativo no processo de aprendizado.

Como já citado, um ISLE (*Intelligent Simulation and Learning Environment*) é um ambiente de simulação e aprendizado inteligente, portanto um ICAI, que contém, normalmente, um simulador, esquema de ajuda avançada, sugestões, explicações e facilidades de orientação, podendo oferecer ao usuário um rota orientada (*guided tour*) que o conduza, ao longo da simulação, de acordo com suas necessidades (Freire, 1998).

O uso de um ISLE ajuda a resolver algumas limitações da simulação pura, a saber:

- *o usuário pode vagar a esmo*: se não tiver algo que o oriente, o usuário pode se perder ou se distrair com a simulação, deixando de gerar conhecimento;
- *inaptidão do usuário gera pouco benefício*: se o usuário não tiver conhecimento prévio do que está sendo simulado, também não produzirá conhecimento ou será incapaz de aprender com os resultados, por ser incapaz de interpretá-los;
- *acomodamento do usuário*: por não ter um objetivo específico, o usuário pode se acomodar e não aproveitar a possibilidade de gerar e testar hipóteses.

³ “O estudo de todos os processos envolvidos através dos quais entradas sensoriais são transformadas, reduzidas, elaboradas e usadas” (Neisser, 1967 *apud* Pinker, 1998)

Essas limitações também poderiam ser superadas com a presença de um tutor humano, mas esse aspecto já foi discutido no início do capítulo.

Existem no mercado alguns ISLEs para as mais diversas áreas, indo de detecção de problemas em circuitos eletrônicos (QUEST) a microeconomia (Smithtown) e chegando ao campo de manufatura industrial (Enxuto).

Figura 5.1 Estrutura de um ISLE

Os componentes básicos de ISLE são o módulo de domínio, o módulo de estudante, o módulo de tutor e o módulo de interface, além do próprio usuário/estudante do ambiente, mostrados na Figura 5.1. Cada um desses componentes é discutido brevemente a seguir.

##*Módulo de Domínio*: tem por responsabilidade armazenar os modelos e seus parâmetros.

Essa base de conhecimento pode ser usada, tanto para a própria simulação, como para o suporte didático oferecido pelo ISLE;

##*Módulo de Estudante*: oferece meios para a avaliação da assimilação do conteúdo pelo usuário do sistema. É neste módulo que é feita a adaptação do conhecimento do ISLE ao usuário do sistema no momento, por exemplo, o que ele já sabe e que pontos precisam de reforço;

##*Módulo de Tutor*: é aqui que é decidido como transferir o conhecimento para o usuário. As formas de transferência podem ser, entre outras, o diálogo Socrático clássico^{4/5}, a implementação do modelo progressivo, no qual o avanço no conteúdo está condicionado ao desempenho do usuário na compreensão e desenvolvimento das tarefas propostas pelo

⁴ De Sócrates (470-399 a.C.), filósofo grego nascido em Atenas. Uma característica peculiar de sua filosofia está em que, ao conversar com um interlocutor, dava a impressão de que era ele quem desejava aprender, e fazia isso através de diálogos que funcionavam como um estímulo para que o interlocutor formulasse suas próprias idéias e pensamentos. Era capaz de se fingir de ignorante, ou mostrar-se mais tolo do que realmente era, dando origem ao termo *Ironia Socrática*. (Marvin, 1997)

⁵ Perguntas e respostas com o objetivo de examinar o raciocínio e descobrir e ajustar falhas no mesmo

ISLE, e o raciocínio baseado em casos, onde as experiências passadas são utilizadas para a resolução de casos parecidos;

• *Módulo de Interface*: este módulo possui dupla função (Norman, 1986). Deve ser projetado de forma que seja fácil para o sistema transmitir informações para o usuário e, ao mesmo tempo, a transformação de **intenções** em **ações e informações** por parte do usuário deve ser facilitada.

O Módulo de Interface pode ser considerado como o ponto mais importante no projeto de um ISLE. É ideal que a interface seja o mais próximo possível do que o usuário está simulando. Alguns aspectos sobre interfaces são apresentados na seção seguinte.

5.2 Considerações sobre interfaces com o usuário

A utilização de um dispositivo pelo usuário nem sempre é simples como se deseja. Em alguns casos, o dispositivo é tão complicado que o usuário se sente frustrado e culpado por não saber como utilizá-lo. Na maior parte dos casos, a culpa não é do usuário, mas do projetista do dispositivo (Norman, 1990) que não fez um bom trabalho. Os controles não estão suficientemente visíveis, não existem pistas de como utilizar os controles e não há *feedback* das ações executadas sobre o dispositivo.

As pessoas normalmente estão cercadas de coisas que devem saber como manipular e utilizar. Além disso, geralmente deparam-se com novos objetos que precisam ser aprendidos enquanto estão executando outras atividades. Se esse novo objeto não tiver sido bem projetado, de forma que seu uso seja natural, a atenção é desviada da ação principal e, dependendo do que estava sendo executado, as consequências dessa distração podem ser indesejáveis.

Como as pessoas gerenciam essa necessidade de aprendizado "em tempo real"? Segundo Norman (Norman, 1990), parte da resposta está na forma como a mente funciona (na psicologia do pensamento e cognição humana), parte está na informação disponível na aparência do objeto (a psicologia das coisas do dia-a-dia) ou, mais especificamente no caso da computação, na interface com o usuário. A parte final vem da habilidade do projetista de tornar a operação clara, de projetar uma boa imagem da operação, e de aproveitar os conhecimentos que as pessoas já possuem. É aqui que a combinação do conhecimento do projetista sobre psicologia humana e sobre como as coisas funcionam se torna crucial.

Novos estudos de psicologia e sociologia indicam que as pessoas tendem a interagir com computadores e outros dispositivos de forma natural e social (Turk, 1998). Ou seja, as pessoas trazem para seus relacionamentos com computadores e dispositivos, atitudes e comportamentos semelhantes àqueles utilizados com outras pessoas. O problema é que as interfaces de computador utilizadas

atualmente não são desenvolvidas visando a utilização de forma natural, mas sim forçando uma "adaptação" às mesmas.

A Tabela 5.1 (Turk, 1998) mostra a evolução dos paradigmas de interação homem-máquina (IHM) com o passar do tempo.

<i>Era</i>	<i>Paradigma</i>	<i>Implementação</i>
1950s	Nenhum	chaves, fios, cartões perfurados
1970s	Máquina de escrever	interface de linha de comando
1980s	Área de trabalho	GUI / WIMP
—	Interação natural	PUI / Pós-WIMP

Tabela 5.1 Paradigmas de interação homem-máquina

Na primeira era (décadas de 50 e 60), a preocupação em elaborar uma camada de abstração entre o computador e o usuário era praticamente nula. As pessoas interagiam com os computadores através do liga/desliga de chaves, mudança de fios, alimentando-os com uma pilha de cartões perfurados e lendo os resultados através de um painel de LEDs ou outra pilha de cartões gerada como resultado do processamento.

Na era seguinte (década de 70), a metáfora de uma máquina de escrever – a linha de comando – foi a dominante no projeto de interfaces para sistemas interativos.

Nos últimos 10 ou 15 anos, o paradigma dominante foi o da mesa de trabalho (*desktop*), onde quase toda a interação com computadores se dá através de interfaces gráficas com base em WIMP (*Windows, Icons, Menus and Pointing Devices*), que englobam as GUIs (*Graphical User Interface*).

Hoje em dia, discute-se um novo paradigma de interface entre homem e máquina: PUI (*perceptual user interfaces* – interfaces perceptuais com o usuário) (van Dam, 1997 *apud* Turk, 1998). Essas interfaces teriam entradas e saídas em multimeios. Na era de interfaces PUIs, técnicas como reconhecimento/sintetização de voz e realidade virtual ou aumentada, além de interfaces com base na visão (tem alguém aí? quem está aí? ele está falando ou está calado?), reconhecimento de gestos com base na aparência (que gesto está fazendo?) e acompanhamento de corpo todo (que partes do corpo estão se movendo? o que isso quer dizer para o sistema?) serão essenciais para o desenvolvimento das interfaces.

Nesse momento, não é interessante discutir as interfaces PUI, uma vez que as técnicas necessárias para a implementação das mesmas não se encontram perfeitamente desenvolvidas (Turk, 1998; Shaffer, 1998). Adotando-se as interfaces GUI/WIMP, espera-se que seus projetos tenham como principais objetivos os seguintes pontos (Norman, 1986; Norman, 1990; Nielsen, 1993; Norman, 1999):

- ⌘ *Velocidade de aprendizado*: em quanto tempo um usuário inexperiente "aprende" a interface? Quanto menor esse tempo, melhor;
- ⌘ *Velocidade de utilização*: um usuário familiarizado com a interface é capaz de executar quantas tarefas em um intervalo de tempo? Quanto maior, melhor;
- ⌘ *Taxa de erros*: quantos erros o usuário cometeu até atingir o seu objetivo? Quanto menor, melhor. Os principais erros cometidos pelos usuário são discutidos no Anexo A;
- ⌘ *Recordação da interface*: depois de algum tempo sem utilizar a interface, o usuário deve ser capaz de rapidamente "reaprender" a utilizá-la;
- ⌘ *Ser consistente*: permitir que o usuário generalize conhecimento sobre um aspecto do sistema para outros aspectos. Adotar a consistência para passar ou receber informações do usuário, ou seja, passá-las ou obtê-las sempre da mesma forma;
- ⌘ *Fornecer um bom modelo conceitual*: facilitar a montagem do modelo mental do usuário (modelo do usuário) de funcionamento do sistema, elaborando um bom modelo conceitual (modelo de projeto) do sistema. Ou seja, se o usuário achar que agindo de tal forma o sistema vai gerar tal resultado, é isso que deve acontecer. Se o modelo conceitual não for bem elaborado, a utilização do sistema será prejudicada. A Figura 5.2, adaptada de Norman, 1990, mostra a relação entre os vários modelos;
- ⌘ *Tornar as coisas visíveis (feedback)*: o sistema deve informar o que está fazendo, em que fase se encontra, que problemas teve e o que está necessitando. Se as coisas estão visíveis, é possível simular o funcionamento do sistema a partir do modelo mental montado (desde que ele corresponda ao modelo conceitual correto) e saber, *a priori*, se a ação pretendida vai gerar o resultado desejado;
- ⌘ *Mapeamento natural*: ao ver uma impressora desenhada em um botão o usuário deve imaginar que, ao pressionar o botão, alguma coisa deve ser impressa, dessa forma, espera-se que o projetista tenha pensado o mesmo no momento do projeto da interface;

Outros dois aspectos que devem ser levados em conta no momento do desenvolvimento de uma interface com o usuário são as capacidades (*affordance*) e as limitações (*constraint*) dos elementos componentes da interface (Norman, 1990; Norman, 1999).

O aspecto de capacidade diz respeito às propriedades de um determinado elemento da interface. Um botão, por exemplo, tem a propriedade de ser pressionado, e é isso que um usuário deve tentar fazer ao ver um botão. O usuário não vai esperar que uma mensagem de erro surja em um botão ou que ele deva digitar algum valor no mesmo, por exemplo. Isso se deve às limitações naturais do elemento botão.

Figura 5.2 Relação entre modelo de projeto, de usuário e do sistema

Já as limitações dizem respeito às incapacidades intrínsecas de um elemento. Um botão desabilitado, por exemplo, indica claramente ao usuário que ele não pode ser utilizado e, por isso, o usuário não deve pressioná-lo. Uma janela que não apresenta campos para entrada de dados automaticamente limita o usuário, ou seja, ele é capaz de perceber que sua intervenção não é necessária nesse momento.

5.3 Considerações finais

A simulação, apesar de permitir o aprendizado construtivista através da formulação e testes de hipóteses, não consegue garantir que o usuário esteja gerando conhecimento adicional. Isso é decorrência do fato de a simulação não ter como guiar ou auxiliar o usuário. Os ISLEs surgiram como forma de auxílio na resolução desse problema. No próximo capítulo é apresentada a especificação de um Ambiente de Simulação e Aprendizado Inteligente para matrizes de discos.

Esta página foi deixada intencionalmente em branco

Capítulo 6

Especificação do Ambiente para Simulação e Aprendizado de RAID

Apresenta-se, a seguir, uma descrição geral do Ambiente proposto e, nas seções seguintes, descrição dos módulos componentes e dos depósitos de dados utilizados pelo mesmo.

O principal objetivo do Ambiente aqui proposto é permitir ao Estudante experimentar arranjos e arquiteturas RAID definidos por um Tutor Humano, ou por ele mesmo, e com base nos resultados dos experimentos, obtidos através de um simulador, aprender de forma construtivista o funcionamento de um RAID, podendo observar seu desempenho em função de cargas de trabalho e níveis de concorrência diferentes.

Os usuários do Ambiente podem ser classificados em dois grupos distintos: **Tutores Humanos** e **Estudantes**. Os Tutores Humanos são responsáveis por incorporar conhecimento ao Ambiente, criando modelos básicos de discos, interfaces e diferentes arranjos de RAID, e gerenciar o desempenho dos Estudantes. Os Estudantes, por sua vez, são aqueles que utilizam o Ambiente para aprender sobre o funcionamento de RAID e simular seu desempenho. Os Estudantes são também avaliados pelo Ambiente através de testes aplicados em determinados momentos. Os resultados desses testes servirão ao Tutor Humano como parâmetro para medir o aproveitamento do Estudante.

Em uma primeira análise, o Ambiente possui algumas características de um ISLE (seção 5.1), sendo composto pelo **meta-módulo de inicialização do ambiente**, responsável por armazenar os modelos iniciais e seus parâmetros, pelo **meta-módulo de avaliação do conhecimento**, que permite que o Tutor Humano avalie continuamente a assimilação do conteúdo pelo Estudante e que o próprio Estudante seja avaliado pelo Ambiente, pelo **meta-módulo de transmissão do conhecimento**, que transfere conhecimento para o Estudante, e pela **interface**, que permite que o Estudante e o Tutor Humano interajam com o Ambiente. A especificação de uma interface para o Ambiente requer conhecimentos adicionais de IHM que estão fora do escopo desta Dissertação.

Para a representação do ambiente proposto e de seus módulos componentes, optou-se pela utilização de Diagramas de Fluxo de Dados – DFDs (Yourdon, 1990). A utilização de uma ferramenta de análise estruturada clássica/moderna não impede o projeto e a implementação do sistema em ferramentas orientadas a objetos.

O DFD é uma ferramenta de modelagem gráfica, utilizada pela análise estruturada clássica e moderna de sistemas, que permite imaginar um sistema como uma rede de processos funcionais (o que deve ser feito), interligados por fluxos de comunicação e depósitos de dados. Ele descreve o fluxo de informação sem uma representação explícita da lógica das funções (como fazer). O DFD foi escolhido por permitir que o modelo possa ser visualizado de forma subdividida, ao mesmo tempo em que não se perde a noção do todo. Como o interesse principal desta dissertação é mostrar *o que* deve acontecer, e não *como* deve acontecer, DFD se mostra uma ferramenta adequada para a especificação do Ambiente.

6.1 Visão Geral

A Figura 6.1 apresenta uma visão geral do Ambiente proposto, seus meta-módulos e depósitos de dados, e a relação existente entre eles. As atividades desenvolvidas pelo Ambiente podem ser agrupadas em três meta-módulos responsáveis pela inicialização do Ambiente, transmissão do conhecimento, e avaliação do conhecimento assimilado pelo Estudante.

Figura 6.1 Visão geral do Ambiente através de um diagrama de blocos

O Tutor Humano interage, basicamente, com o meta-módulo de inicialização do Ambiente e de avaliação do conhecimento. Durante a interação com o meta-módulo de inicialização, os **depósitos de discos** são populados com informações básicas sobre discos, interfaces e RAID. Essas informações serão utilizadas pelo Ambiente durante as simulações executadas pelos Estudantes. Devem ser populados, também, os **depósitos básicos**, que armazenam o conhecimento que será transmitido ao

Estudante, e o conhecimento gerado automaticamente pelo Ambiente (características de desempenho de alguns modelos RAID definidos pelo Tutor Humano). Junto com o conhecimento básico, devem ser especificados testes que serão utilizados pelo Ambiente para avaliar o aproveitamento do Estudante, ou seja, se o Estudante está compreendendo o que está acontecendo.

É necessário também, durante a inicialização, “criar os Estudantes”, ou seja, incluir no Ambiente os **dados dos Estudantes**, como as informações básicas para individualização do Estudante (nome e *email*, por exemplo) e informações sobre conhecimento prévio acumulado pelo mesmo sobre o que lhe será transmitido (“o Estudante *Y* já sabe como calcular unidade de fragmentação e tempo médio de busca, mas precisa lembrar como são calculados os códigos de Hamming”, por exemplo). Essas informações prévias sobre o conhecimento são chamadas, no Ambiente, de **Pontos de Parada**. Os Pontos de Parada também podem ser definidos pelo próprio Estudante (“é preciso rever o impacto do tempo de acesso ao disco responsável pela paridade no RAID 3 no tempo de acesso global”, por exemplo).

Através do meta-módulo de avaliação do conhecimento, o Tutor Humano tem acesso à **avaliação dos Estudantes**. As avaliações são feitas pelo Ambiente com base nas respostas dadas, pelos Estudantes, aos testes aplicados. O Tutor Humano tem condições, através do meta-módulo avaliação de conhecimento, de acompanhar um Estudante, sabendo que informações já foram absorvidas e/ou assimiladas pelo Estudante, e quais ainda precisam de reforço.

O Simulador foi considerado como um elemento externo por ser um componente que pode ser substituído sem prejuízo à funcionalidade global do Ambiente. É possível, inclusive, substituir o Simulador em tempo de execução do Ambiente. O Simulador deve, uma vez recebido o modelo RAID e os parâmetros do mesmo, solucioná-lo e obter medidas de desempenho que serão devolvidas ao Ambiente para tratamento adequado.

O Estudante, por sua vez, interage com o Ambiente através dos meta-módulos de transmissão e avaliação de conhecimento. Através do meta-módulo de transmissão de conhecimento, o Estudante tem acesso ao conhecimento disponível no Ambiente. Esse conhecimento pode ser transferido de várias formas, inclusive as citadas como exemplo na seção 5.1 e cabe ao meta-módulo, com base nas informações disponíveis sobre o Estudante, decidir qual das formas utilizar. O Estudante pode ainda, criar e simular seus próprios modelos RAID, utilizando os discos e interfaces disponíveis no Ambiente, e observar o comportamento dos mesmos. O relacionamento entre o Estudante e o meta-módulo de avaliação de conhecimento se dá no momento de avaliar quanto do conhecimento transmitido foi assimilado. Essa avaliação acontece em momentos definidos como, por exemplo, depois de o Estudante ter tido acesso à alguma informação nova ou após o resultado de uma simulação estar disponível. No último caso, o Estudante deve ser capaz de explicar, com base no seu

conhecimento, a razão de determinado resultado (um tempo médio de resposta de um RAID, por exemplo).

Nas seções seguintes serão detalhados os meta-módulos. Ao longo das seções em questão serão feitos comentários sobre os depósitos de dados necessários, informando algumas características básicas sobre os elementos componentes dos mesmos, mas os detalhes sobre os depósitos são apresentados a partir da seção 6.2.

6.1.1 Meta-módulo de inicialização do Ambiente

Esse meta-módulo engloba 5 módulos, representados na Figura 6.2, responsáveis pela inicialização do Ambiente e o armazenamento das informações básicas para a utilização do mesmo.

Figura 6.2 Diagrama de blocos do meta-módulo de inicialização do Ambiente

O módulo A.1 (Alterar Conhecimento) é responsável por incluir, no Ambiente, as informações básicas sobre os discos (como tempo de busca, velocidade angular e número de cilindros, por exemplo), interfaces (IDE ou SCSI, por exemplo) e regras (cálculo de redundância e relacionamento entre os discos, por exemplo) que serão utilizadas para a construção das arquiteturas e arranjos RAID. Essas informações são armazenadas, respectivamente, nos depósitos de dados D1, D2 e D3. O

detalhamento do conteúdo dos depósitos utilizados por todos os módulos do Ambiente é feito na seção 6.2

Através dos módulos componentes do meta-módulo de avaliação de conhecimento, o Estudante é avaliado. Para isso, é necessário informar ao Ambiente quais testes podem ser aplicados ao Estudante. Isso é feito, também, através do módulo A.1, onde o Tutor Humano deve informar os testes que podem ser aplicados (múltipla escolha ou dissertativos, por exemplo), e a resposta correta. Essas informações são adicionadas ao depósito D10.

É necessário ao Ambiente ter informações bibliográficas – conhecimento sobre a área em que ele atuará como Tutor – para que este possa ser transferido ao Estudante. O Ambiente permite que o Tutor Humano inclua essas informações bibliográficas também através do módulo A.1. Essas informações não podem ser, simplesmente, as referências bibliográficas para que o Estudante as obtenha e as leia para, depois, interagir com o Ambiente. Assim, o Tutor Humano deve elaborar um resumo da referência, que pode conter recursos multimeios (imagens e animações, por exemplo), e é esse resumo que é armazenado em D5, com o conhecimento bibliográfico, da Figura 6.2.

O módulo A.1 deve permitir também que, uma vez incluídas as informações em qualquer um dos depósitos de dados manipulados por ele, essas possam ser visualizadas e alteradas pelo Tutor Humano quando lhe for conveniente.

Depois de o Tutor Humano popular os depósitos com informações sobre discos, interfaces e regras, ele deve interagir com o módulo A.3 (Montar Arranjo) e, através deste, montar arranjos básicos RAID que utilizem os discos, interfaces e regras disponíveis. Esses arranjos dizem, por exemplo, que deve ser montado um RAID 3 (informação disponível através das regras armazenadas no depósito D3) utilizando controladoras SCSI Ultra-2 (informações sobre a controladora disponível no depósito de interface – D2) e utilizando discos Quantum Tornado (dados sobre os discos armazenados no depósito D1). Uma vez montado o arranjo, ele é armazenado no depósito de arranjos RAID (depósito D4) que serão utilizados (experimentados) pelo Estudante.

No conhecimento bibliográfico que o Tutor Humano inclui no Ambiente, através do módulo A.1, há alguns casos onde arranjos de RAID são descritos. Esses arranjos podem ser incluídos no Ambiente através do módulo A.3. Uma vez definido esse arranjo, ele é parametrizado pelo Tutor Humano com os parâmetros da literatura, e o Ambiente o experimenta através do módulo A.4 (Gerar Medidas) para obter as medidas de desempenho. Essas medidas, que serão armazenadas no depósito D6, servirão de referência para o Estudante pois, quando ele alterar os parâmetros desse arranjo, gerando um novo modelo, será possível observar as diferenças de desempenho entre o modelo original, e o parametrizado ou alterado por ele. O modelo é solucionado pelo módulo A.4 com o auxílio de um Simulador externo, como foi dito anteriormente.

Como já foi citado, o Ambiente é independente de simulador, ou seja, qualquer simulador que possa ser configurado externamente através de arquivos pode ser usado. Quando for necessário interagir com o Simulador, o Ambiente deve recuperar o arranjo desejado, parametrizá-lo, e convertê-lo para um arquivo texto compatível com o Simulador escolhido. Para isso, os módulos A.4, e B.5 (discutido posteriormente), devem possuir uma camada de abstração (quando houverem mais características no modelo que as suportadas pelo Simulador) e inferência (quando o modelo do Ambiente não for suficientemente completo para o Simulador), onde os arranjos parametrizados do Ambiente sejam convertidos em arquivos de configuração para o Simulador escolhido. A abstração pode ser totalmente automática, mas a inferência necessitará de interação, ou do Tutor Humano ou do próprio Estudante, para informar os valores de parâmetros que não sejam possíveis de obter a partir do modelo de RAID do Ambiente. Uma vez disponível o arquivo de configuração, o Simulador é iniciado e, quando os resultados estiverem disponíveis, os mesmos módulos A.4 e B.5 irão processá-los adequadamente, armazenando-os no depósito D6, ou exibindo-os para o Tutor Humano ou Estudante. No Capítulo 7 são apresentados dois simuladores disponíveis na literatura e sua aplicabilidade ao Ambiente aqui proposto.

Uma vez que o Tutor Humano tenha provido o Ambiente de conhecimentos bibliográficos e/ou de simulação, o mesmo Tutor Humano deve elaborar as informações (Aulas) que serão transmitidas aos Estudantes. Isso é feito através do módulo A.5 (Gerar Informações), onde o Tutor Humano seleciona determinadas referências e resultados de experimentos e os arranja em um documento multimeios que será armazenado no depósito de aulas (D7) e apresentado ao Estudante no momento oportuno. As informações que compõe cada uma das aulas serão apresentadas com mais detalhes na descrição do depósito D7 na seção 6.2.4.

Durante o processo de inicialização do Ambiente, o Tutor Humano deve definir, também, os Estudantes que utilizarão o mesmo. Essa definição se dá através do módulo A.2. Nesse módulo são fornecidas informações básicas sobre o Estudante, como conhecimento prévio e melhor forma de transmissão de conteúdo para aquele Estudante, e alguns pontos que, de antemão, o Tutor Humano sabe que o Estudante tem interesse e/ou necessidade de assimilar (Pontos de Parada) e uma “Rota Orientada” (*guided tour*). As informações básicas ficam armazenadas no depósito D8, e os chamados Pontos de Parada, no depósito D9.

6.1.2 Meta-módulo de transmissão do conhecimento

Os 5 módulos englobados por este meta-módulo, apresentados na Figura 6.3, são responsáveis por interagir com o Estudante, transferindo-lhe conhecimento e permitindo-lhe simular novos modelos

RAID. A interação do Estudante com o Ambiente se dá, principalmente, através dos módulos B.1 (Pedir Conhecimento) e B.2 (Apresentar Conhecimento).

Figura 6.3 Diagrama de blocos do meta-módulo de transmissão do conhecimento

Quando o Estudante pede algum conhecimento ao Ambiente, isto é feito através do módulo B.1. A estrutura do depósito de dados de Aulas (D7) permite que o Estudante solicite conhecimento através de uma requisição por aulas semelhante (raciocínio baseado em casos), por aulas que sejam logicamente seqüentes a atual (modelo progressivo), ou mesmo através de aulas que possuam determinada palavra chave. O Estudante pode solicitar uma nova aula, também, através da seleção de algum Ponto de Parada definido anteriormente no depósito de dados apropriado (D9).

Uma vez tendo recebido de B.1 a solicitação de transmissão de determinado conhecimento ao Estudante, o módulo B.2 recupera a aula correta do depósito de dados D7 para que a mesma seja apresentada ao Estudante através da interface gráfica do Ambiente. Como as aulas são documentos multimeios, a interface deve ser capaz de apresentar e permitir que o Estudante manipule informações textuais, gráficas e/ou animadas. A forma como esse conhecimento será transmitido ao Estudante (diálogo socrático ou modelo progressivo, por exemplo) deve ser decidida com base nas preferências pessoais do Estudante, armazenadas também em D8. Além das informações solicitadas, o Ambiente exibe para o Estudante os Pontos de Parada definidos até então, recuperados de D9, e as aulas já apresentadas, recuperadas de D8, para que o Estudante tenha condições de saber seu andamento, o que já foi visto, e o que falta ser revisto.

Durante a interação com o módulo B.2, o Estudante pode achar necessário definir um Ponto de Parada para essa aula, de forma que ela possa ser revista ou revisitada em um momento futuro. Isso é feito através do módulo B.3 (Definir Ponto de Parada), que armazena essa informação no depósito de dados D9. Os Pontos de Parada podem ser definidos, também, pelo próprio Ambiente, com base no desempenho do Estudante nos testes. Isso é feito através de uma mensagem recebida pelo módulo B.3 e originária do módulo C.2

Antes que o Estudante solicite novo conhecimento, o próprio Ambiente se encarrega de selecionar e aplicar pequenos testes para avaliar a assimilação do conhecimento transferido. Isso é feito através do módulo C.2, descrito mais à frente, que é ativado pelo módulo B.2.

No momento em que o Estudante desejar criar seu próprio modelo RAID e experimentá-lo para observar seu comportamento e confrontá-lo com a literatura ou, simplesmente, para observar seu comportamento frente a determinada carga de trabalho ou com determinado tipo de interface ou disco, isso pode ser feito através do módulo B.4 (Criar Arranjos de Estudante). Através desse módulo, tal como o Tutor Humano faz através do módulo A.3, o Estudante pode criar um arranjo RAID utilizando os discos, interfaces e regras disponíveis no Ambiente. Esses arranjos são armazenados no depósito D12.

Tendo o seu arranjo armazenado, ou tendo escolhido um arranjo definido pelo Tutor Humano, o Estudante pode experimentá-lo. Isso é feito através do módulo B.5 (Experimentar Modelos), onde o Estudante seleciona o arranjo, parametriza-o, e submete-o ao Simulador do Ambiente. Tal como acontece no módulo A.4, o Ambiente deve fazer abstração ou inferência no modelo do Ambiente e transformá-lo em um modelo que possa ser submetido ao Simulador. O Estudante pode, também, parametrizar de forma diferente algum dos arranjos de RAID definidos pelo Tutor Humano e armazenados em D4. Quando os resultados da simulação estiverem disponíveis, o módulo B.5 os exibe ao Estudante, para que este os analise e tire suas conclusões a cerca do desempenho. Para tirar

essas conclusões, o Estudante pode necessitar de conhecimento adicional, e esse é obtido através de uma solicitação de conhecimento ao módulo B.1.

6.1.3 Módulo de avaliação de conhecimento

Através dos 2 módulos componentes deste meta-módulo, apresentados na Figura 6.4, o Ambiente pode avaliar o conhecimento adquirido pelo Estudante, e manter o Tutor Humano a par do progresso de cada Estudante.

O modulo de validação do conhecimento (módulo C.2) é ativado pelo próprio Ambiente, através do módulo B.2, depois que algum conhecimento é transmitido ao Estudante. Este módulo aplica pequenos testes ao Estudante. A seleção de qual teste aplicar é feita com base no conhecimento que já foi transmitido ao Estudante (informação disponível no depósito de dados D8) e no conhecimento que o Estudante deve ou quer rever – os Pontos de Parada definidos para o Estudante (depósito D9).

Figura 6.4 Diagrama de blocos do meta-módulo de avaliação de conhecimento

Os testes, que foram inseridos no Ambiente pelo Tutor Humano através do módulo A.1, estão armazenados no depósito D10. Se o teste for respondido corretamente, o Ambiente assume que o Estudante assimilou aquele conhecimento, e as informações sobre o Estudante são atualizadas para refletir esse progresso. Se a resposta fornecida pelo Estudante não for considerada correta pelo

Ambiente, um Ponto de Parada para aquele conhecimento é definido através de uma mensagem para o módulo, e a resposta correta é dada ao Estudante.

Os resultados dos testes aplicados a um Estudante, assim como os próprios testes, as respostas dadas e o tempo gasto em cada um dos testes, são armazenados no depósito de avaliações de Estudantes (depósito de dados D11). Essas informações serão utilizadas pelo Ambiente para gerar as informações necessária para manter o Tutor Humano a par do desempenho do Estudante.

De tempos em tempos, o Tutor Humano pode querer saber o desempenho de todos os Estudantes, ou de um em particular. Isso é feito através do módulo C.1 (Gerar Situação de Estudante), onde o Tutor Humano seleciona um Estudante, e o Ambiente coleta suas informações de desempenho. Essas informações de desempenho contém os Pontos de Parada definidos, disponíveis em D9, os conhecimentos já transmitidos, armazenados em D8, e o desempenho do Estudante nos testes aplicados pelo Ambiente, armazenados pelo módulo C.2 no depósito D11. Com base nessas informações, o Tutor Humano pode alterar a situação de um Estudante como, por exemplo, definir novos Pontos de Parada através do módulo A.2. Quando o Tutor Humano solicita as informações de um Estudante, o próprio Estudante recebe uma cópia dessas informações para que ele possa auto-avaliar seu desempenho.

Nessa seção, e nas duas anteriores, foram apresentados os meta-módulos e os módulos componentes do Ambiente, mas não foram detalhados os depósitos de dados utilizados. Isso é feito na próxima seção e suas sub-seções.

6.2 Detalhamento de Depósitos

As informações armazenadas nos depósitos são a alma do Ambiente. Como base nas informações dos depósitos D8 e D9, por exemplo, o Ambiente é capaz de selecionar o conhecimento apropriado a ser transmitido a cada Estudante. Ao longo das subseções, serão apresentados as características mais importantes que devem estar presentes em cada depósito utilizado pelo Ambiente.

6.2.1 Discos, Interfaces e Regras (D1, D2 e D3)

Os depósitos D1, D2 e D3 são responsáveis por armazenar as características dos discos e interfaces que serão utilizados no modelos e a forma de relacionamento entre eles.

Para a parametrização de discos, a bibliografia (Thekkath et al., 1992; Ruemmler et al., 1994; Ganger, 1995; Worthington et al., 1996) enumera algumas características que são julgadas importantes pelos seus autores. Os simuladores estudados, e apresentados no Capítulo 7, utilizam algumas características coincidentes com a literatura, e outras que não estão presentes nas referências citadas.

Pela característica do Ambiente, de ser independente de simulador, é necessário que o depósito D1 contenha informação suficiente para que seja gerado um arquivo de configuração utilizável pelos simuladores estudados. O modelo de disco adotado pelo DiskSim (que será apresentado na seção 7.2.2) é o mais completo entre os analisados e, por isso, foi escolhido, com simplificações, como modelo para o Ambiente. Desta forma definiu-se que D1 deve armazenar as seguintes características:

- ≠# Número de superfícies de gravação no disco;
- ≠# Número de zonas no disco;
- ≠# Número de setores por trilha dentro de cada zona;
- ≠# Tamanho do setor;
- ≠# Limite (cilindro inicial e final) de cada zona;
- ≠# *Track skew* e *cylinder skew* de cada zona;
- ≠# Velocidade angular, em RPM;
- ≠# Tempo de troca de trilha;
- ≠# Tempo de troca de cabeça;
- ≠# Tempos de busca por 1 a 10 cilindros, região em que a curva de tempo de busca obtida com a equação de Lee (Equação 2.1) não tem uma boa aproximação em relação ao tempo real de busca;
- ≠# Tabela com tempos de busca para algumas distâncias, que serão usados para obter o tempo de busca por uma determinada distâncias através de interpolação;
- ≠# Tempo de *full stroke*;
- ≠# Tempo médio de busca no disco, obtido por informação declarada do fabricante, ou por experimentos de campo onde são coletados tempos de várias buscas e, depois, calculado o tempo médio utilizando uma das técnicas descritas na seção 2.1;
- ≠# Tempo adicional de assentamento para escrita;
- ≠# Tempo de transferência de um setor de 512 bytes (usado na determinação da largura de banda máxima do disco).

Com essas informações é possível construir um modelo relativamente preciso de um disco que pode ser utilizado por praticamente qualquer simulador. Para qualquer informação não armazenada e que seja necessária a um simulador específico, o Ambiente deve solicitar a informação ao Estudante antes de simulá-lo.

No tocante às informações armazenadas em D2 sobre as interfaces, pouca ou nenhuma referência à parametrização de interfaces para simulação é feita na bibliografia. Tempo de manipulação de bloco de 512 bytes, tamanho máximo da fila, tempo de arbitragem, tipo de arbitragem,

velocidade de transferência do disco para a controladora e da controladora para o disco, política de escalonamento das requisições e características de cache da controladora são alguns parâmetros que podem ser considerados (Ganger et al., 1999). Apenas o DiskSim, dos simuladores estudados, considera as características das interfaces no desempenho das matrizes de disco. Definiu-se, com base nos parâmetros apresentados em (Ganger et al., 1999) que, para o Ambiente, os seguintes parâmetros são necessários:

- ≠# Tempo de processamento de um bloco de 512 bytes antes de enviá-lo ou após recebê-lo (atraso imposto pela controladora);
- ≠# Tempo de transferência de um bloco de 512 bytes de/para os discos através de um canal da controladora (largura de banda do canal);
- ≠# Política de escalonamento das requisições de leitura/escrita (FIFO, LIFO, SSF, com ou sem preempção);
- ≠# Número de canais da controladora;
- ≠# Número máximo de discos por canal da controladora;
- ≠# Tipo de canal (exclusivo, como a IDE, ou de banda compartilhada, como o SCSI)

A largura de banda de dados entre o disco e a controladora é dado pela menor das larguras de banda. Por exemplo, se o disco tiver um tempo de transferência de setor igual a 0,106ms (largura de banda aproximada de 4,5Mbytes/s), e a controladora tiver o tempo de transferência igual a 0,053 (largura de banda aproximada de 9Mbytes/s), a largura de banda do conjunto disco-controladora será de 4,5Mbytes/s.

No tocante ao depósito D3, responsável pelas informações sobre redundância e fragmentação, as seguintes características foram definidas como necessárias:

- ≠# Algoritmo utilizado no cálculo da informação redundante (ou-exclusivo, Códigos de Hamming, Códigos de Reed-Solomon, etc.);
- ≠# Local de armazenagem (um disco especial ou ao longo de vários discos);
- ≠# Número de discos na organização;
- ≠# Tamanho da unidade de fragmentação (em bytes);
- ≠# Regra de distribuição dos fragmentos (como os fragmentos são distribuídos ao longo dos vários discos);
- ≠# Detalhes topológicos da organização (discos por canal de controladora e ortogonalidade dos discos em relação à controladora)

De posse das informações armazenadas em D1, D2 e D3, o Tutor Humano e o Estudante são capazes de montar diversos arranjos RAID ou, mesmo, apenas discos simples, que são armazenadas nos Depósitos D4 e D12, apresentados na seção seguinte.

6.2.2 Arranjos RAID do Ambiente e do Estudante (D4 e D12)

O depósito D4 é responsável por armazenar os arranjos RAID utilizados pelo Ambiente para a transmissão do conhecimento, e pelos simuladores para as simulações. A função desse depósito é armazenar o relacionamento entre discos, interfaces e regras, por exemplo, discos Seagate Barracuda 4LP ST34371WD (informações em D1) ligados a uma controladora Adaptec AHA-2940UW (informações em D2) implementando um *strip set* sem paridade de dois discos (informações de como os dados são divididos e a redundância é calculada em D3). O depósito D12 tem a mesma função, mas as relações entre os elementos são definidas pelo Estudante, durante a utilização do Ambiente. Assim, D4 e D12 devem conter as seguintes informações:

- ## Regra que será utilizada (armazenado em D3);
- ## Tipo de controladora a ser utilizada (armazenado em D2);
- ## Tipos de discos que serão utilizados. Dependendo da matriz, pode haver vários discos, e cada um pode ser de um tipo, por exemplo, em um RAID 3, o disco de paridade pode ser um mais rápido que os demais.

Esses arranjos serão utilizados pelos módulos A.4, onde é gerado o Conhecimento de Simulação, e B.5, no qual o Estudante experimenta os arranjos. Nos dois casos, os arranjos são parametrizados e, então, se tornam modelos que podem ser abstraídos ou inferidos para se transformarem em modelos que podem ser submetidos aos Simuladores.

6.2.3 Conhecimento Bibliográfico e de Simulação (D5 e D6)

No depósito D5 ficam as informações (textos, tabelas e ilustrações) sobre RAID disponíveis na bibliografia. São informações, armazenadas em uma base de dados, de forma que possam ser recuperadas posteriormente pelo Estudante através de palavras-chave. Sugere-se como estrutura básica desta base de dados, as seguintes informações:

- ## Identificador do documento;
- ## Autores;
- ## Palavras-chave;
- ## Referência bibliográfica (para permitir a recuperação do texto original pelo Estudante);
- ## Resumo do documento.

No resumo do documento, o Tutor Humano pode incluir textos, imagens ou qualquer outra forma de representação de informação que ele considerar como útil ao Estudante antes de o mesmo ter acesso ao texto original referenciado. Quando o documento apresentar resultados experimentais, o Tutor Humano pode criar um arranjo RAID equivalente ao proposto no documento e submetê-lo ao simulador. Os resultados obtidos na simulação devem ser armazenado no depósito D6. Com base nesses modelos e no conhecimento armazenado em D5, o Estudante pode estudar os arranjos propostos na bibliografia, sem precisar construí-los, parametrizá-los a seu gosto, e observar as alterações nos resultados aprendendo de forma construtivista.

Pela presença de textos e gráficos, o depósito D5 deve ser um depósito multimeios, que permita o armazenamento de textos, imagens, animações e áudio, enquanto o depósito D6 será uma simples tabela onde serão armazenadas referências ao arranjo utilizado (armazenado em D4), ao conhecimento bibliográfico utilizado para construir o modelo simulado (armazenado em D5) e os resultados obtidos pela experimentação do modelo.

6.2.4 Aulas (D7)

A função do depósito D7 é agregar às informações disponíveis na bibliografia (D5) e às obtidas através de simulação (D6), observações do Tutor Humano sobre o Conhecimento que será transmitido aos Estudantes. Tal como D5, este depósito é uma base de dados multimeios, capaz de armazenar textos e ilustrações que serão utilizados pelo módulo B.2 para apresentar o conteúdo ao Estudante.

Como o Ambiente deve ser capaz de transmitir conhecimento de várias formas, o depósito D7 deve estar estruturado de forma que o módulo B.2 seja capaz de selecionar a forma apropriada para transmissão de conhecimento ao Estudante como, por exemplo, Diálogo Socrático, modelo progressivo e raciocínio baseado em casos, entre outros. Se o Estudante optar pelo diálogo Socrático, a base deve ter as perguntas e comentários que serão feitas ao usuário de forma a estimular seu raciocínio e provocá-lo a experimentar os modelos e observar os resultados; na escolha pelo modelo progressivo, o depósito deve ser capaz de oferecer, ao módulo B.2, texto estruturado de forma mais ou menos encadeada, de forma a criar uma dependência entre informações; na opção pelo raciocínio baseado em casos, as informações devem estar agrupadas em situações semelhantes e os casos devem estar bem definidos.

Como o mesmo Conhecimento pode ser transmitido de várias formas (por exemplo, as três citadas no parágrafo anterior), o depósito fatalmente conterá informações redundantes, ou seja, uma

mesma informação estará armazenada de formas diferentes, de forma que o módulo B.2 recupere a informação na forma correta para o Estudante.

Desta forma, a seguinte estrutura básica é sugerida para o armazenamento das aulas:

- ≠# Identificador de aula;
- ≠# Identificador do documento;
- ≠# Próximos documentos na sequência lógica;
- ≠# Pré-requisitos (aulas que devem ser apresentadas antes desta);
- ≠# Perguntas que podem ser feitas;
- ≠# Respostas às perguntas;
- ≠# Aulas semelhantes;

6.2.5 Estudantes (D8)

Além das informações básicas (nome, *email*, etc.), o depósito D8 deve ser capaz de informar ao Ambiente quais aulas já foram transmitidas ao Estudante e qual a melhor forma, ou a forma preferida, de passar conhecimento para esse Estudante. Nas informações sobre o que já foi passado ao Estudante, o depósito deve ser capaz de responder a ordem de passagem das informações, e quais perguntas já foram feitas ao Estudante sobre aquele assunto para, no momento em que for necessário avaliar a assimilação do conhecimento, o Ambiente não faça a mesma pergunta mais de uma vez. Assim, sugere-se a seguinte estrutura para o depósito de estudantes:

- ≠# Identificador de estudante;
- ≠# Informações básicas sobre o estudante (nome, *email*, endereço, etc...);
- ≠# Melhor forma de transmitir conhecimento;
- ≠# Lista de testes já aplicados;
- ≠# Lista de aulas já apresentadas;

6.2.6 Pontos de Parada (D9)

O Ambiente deve ser capaz de reconhecer os pontos que o Estudante pediu para rever, ou os que o módulo de avaliação de conhecimento indicou que devem ser reforçados. Essas informações devem ficar armazenadas no depósito D9. Esse depósito contém tuplas que identificam o estudante, e o conhecimento (aula ou aulas) que precisa ser transferido para ele, juntamente com o momento em que o Ambiente (ou o próprio Estudante) decidiu por essa necessidade e a ordem com que esses pontos de parada devem ser visitados. As seguintes informações foram definidas como prioritárias para o depósito D9:

- ≠# Identificador de estudante;
- ≠# Identificador do ponto de parada;
- ≠# Identificador de aula;
- ≠# Instante de definição do ponto de parada;
- ≠# Ordem de apresentação;
- ≠# Quem definiu o ponto de parada (Estudante, Tutor Humano ou Ambiente);

6.2.7 Avaliação de Estudantes e Testes (D11 e D10)

Para que o Tutor Humano saiba da situação de cada Estudante, o Ambiente deve ser capaz de municiá-lo com essas informações. Para isso, o resultados das avaliações de conhecimento aplicadas à cada Estudante devem estar armazenadas no depósito D11, que deve ter as seguintes informações:

- ≠# Identificador de estudante;
- ≠# Identificador do teste;
- ≠# Resposta dada pelo Estudante;
- ≠# Tempo para resposta;
- ≠# Avaliação da resposta (correta, incorreta, parcialmente correta, etc.);

O depósito D10, por sua vez, armazena os possíveis testes que podem ser aplicados aos Estudantes. Os testes podem ser de múltipla escolha, subjetivos ou qualquer outra forma de avaliação de conhecimento que o Tutor Humano julgar interessante. Dessa forma, a seguinte estrutura é proposta para o depósito em questão:

- ≠# Identificador do teste;
- ≠# Texto do teste;
- ≠# Alternativas;
- ≠# Resposta correta;
- ≠# Ação a ser tomada no caso de resposta errada (definir ponto de parada, notificar Estudante, notificar Tutor Humano, etc...).

6.3 Considerações finais

Este capítulo apresentou o Ambiente de Simulação e Aprendizado Inteligente para RAID proposto nesta dissertação e fez considerações sobre os módulos componentes e os depósitos de dados necessários. Como foi afirmado em vários pontos do capítulo, o Ambiente deve ser independente de

Simulador, ou seja, cabe ao Tutor Humano e, possivelmente ao Estudante, selecionar o Simulador que ele deseja utilizar para experimentar seus modelos.

Para atender a essa característica do Ambiente foram estudados alguns Simuladores de discos magnéticos e RAID disponíveis na literatura e a sua aplicabilidade ou não ao Ambiente aqui proposto. Os resultados desse estudo são apresentados no capítulo seguinte.

Esta página foi deixada intencionalmente em branco

Capítulo 7

Simuladores de discos e RAID

Ao longo deste capítulo são apresentadas informações sobre os simuladores de discos magnéticos e RAID estudados e sua aplicabilidade ao Ambiente proposto no capítulo anterior. São apresentados aspectos como elementos modelados e/ou simulados, nível de detalhamento da configuração, resultados oferecidos e observações gerais.

Durante o período inicial deste trabalho, fez-se uma longa busca por simuladores de discos magnéticos e, preferencialmente, RAID que pudessem ser utilizados pelo Ambiente. Foram encontrados dois simuladores: RAIDframe (Gibson et al., 1995; Courtright II et al., 1996), e DiskSim (Ganger, 1995; Ganger et al., 1999).

Nas seções seguintes serão feitas considerações sobre como esses simuladores representam e manipulam os diferentes RAID e como eles podem ser alterados para incluir uma arquitetura nova. Discute-se como os discos magnéticos foram modelados no simulador em questão, e as principais características do mesmo. Por fim, na seção 7.3 são feitas considerações sobre a utilização de cada um dos simuladores no Ambiente proposto nesta dissertação.

7.1 RAIDframe – A Prototyping Tool for RAID Systems

O RAIDframe é, na realidade, um *framework* para prototipação e avaliação rápida de matrizes de discos. Os protótipos construídos podem ser avaliados em três ambientes distintos: como um *software* controlador de dispositivos rodando sobre discos reais, processos de usuário rodando sobre discos reais, ou um simulador orientado a evento.

Courtright II et al. resolveram, em 1994, criar um sistema de avaliação de matrizes que reduzisse os custos e a complexidade de experimentar novas arquiteturas. Esse ambiente, o RAIDframe, foi criado a partir do RaidSim, um simulador de matrizes de disco criado na Universidade de Berkeley. O RaidSim, por sua vez, foi criado a partir de um controlador de dispositivo para o sistema operacional Sprite. A esse simulador foram incorporadas novas funções de controle da matriz de discos, aumentada a coleta de dados para estatísticas da simulação e funções de depuração e geração de carga de trabalho sintética. Durante os ajustes, cerca de 36% do código fonte do RaidSim foi alterado.

O foco principal do RAIDframe não é simulação, mas prototipação. Apesar disso, o seu simulador permite parametrizar os protótipos e experimentá-los. A carga de trabalho utilizada pode ser gerada sinteticamente, através do gerador incorporado, ou pode ser usado um arquivo de *trace* externo.

O pacote foi escrito em C e, na sua versão atual (1.1.1), é capaz de simular RAID 0, 1, 4, 5 e 6 além de outras forma de redundância, como o *parity declustering* proposto por Mark Holland (Holland, 1994). O código fonte pode ser compilado numa vasta gama de plataformas com pouco ou nenhuma alteração, incluindo processadores Intel rodando *Linux* com *kernel 2.2.x*.

7.1.1 Representação de RAIDs

Para os autores do RAIDframe, as operações sobre um RAID podem ser representadas por seqüências de operações primitivas (por exemplo, escrita e leitura) e essas operações devem estar disponíveis a um projetista de arquitetura, que deve arranjá-las de uma forma que represente a arquitetura que está sendo projetada. A forma escolhida para representar essas primitivas e a ordem em que seriam utilizadas foram os Grafos Orientados Acíclicos (DAG – *Directed Acyclic Graphs*).

A escolha dos DAGs para a representação das primitivas foi natural tendo em vista os objetivos estabelecidos: representação clara, dependências claras entre os nós do DAG e facilidade de submeter o DAG a um motor (*engine*) para execução e/ou interpretação.

Figura 7.1 DAG de pequena escrita bem sucedida em RAID 6

A Figura 7.1 mostra um DAG utilizado para a modelagem de RAID 6 no RAIDframe. Os nós representam operações primitivas e os arcos representam dependências (controle ou dado) que limitam a execução. As primitivas representadas na figura são: H (cabeçalho), T (terminador), R (leitura de disco), W (escrita em disco), XOR (ou-exclusivo) e Q (cálculo de Reed-Solomon). Os índices p, d e q

indicam, respectivamente, paridade, dados e símbolos Reed-Solomon. A função das primitivas H e T é apenas definir o início e o fim do DAG. O caminho H-Rq-Q-Wq-T representa o cálculo da paridade Q (códigos de Reed-Solomon), assim como o caminho H-Rp-XOR-Wp-T representa o cálculo do primeiro conjunto de paridade. Os outros caminhos (H-Rd-Wd-T) representam a leitura do dado antigo (para ser utilizado no cálculo das paridades) e a escrita do dado novo na matriz.

A criação de uma nova arquitetura RAID no RAIDframe necessita de duas fases. Na primeira são definidos os DAGs necessários à arquitetura (leitura, escrita, leitura com falha, escrita com falha, por exemplo) e, na segunda, é necessário acrescentar algumas linhas de código em arquivos específicos do *framework* (principalmente o arquivo com as arquiteturas RAID – `rf_layout.c` e `rf_layout.h`). Uma vez tendo sido feitas as alterações, é só compilar novamente o *framework* e utilizar a nova arquitetura nos arquivos de configuração, na seção de LAYOUT, como será apresentado na seção 7.1.3.

O RAIDframe está muito bem documentado, tanto em seu manual de usuário (Courtright II et al., 1996), quanto em relatório técnico que o descreve (Gibson et al., 1995) e no próprio código fonte. A inclusão ou alteração de arquiteturas é uma tarefa apenas de consulta à documentação e programação em C.

7.1.2 Modelagem de discos magnéticos

Durante o processo de configuração do RAIDframe é necessário especificar, entre outras coisas, os discos que serão utilizados durante a prototipação ou simulação. Se o RAIDframe for utilizado como ferramenta de prototipação, não é necessário informar características dos discos, pois o mesmo executará as operações de entrada e saída diretamente no disco para coletar tempos de acesso e serviço que serão utilizados para a avaliação de desempenho da arquitetura. Decidindo-se por utilizar o RAIDframe como simulador, aí sim torna-se necessário modelar os discos pois, nesse modo de operação, os dados não são escritos ou lidos dos discos reais, e os tempos de acesso e serviço são estimados por simulação.

O RAIDframe considera, durante a modelagem de discos magnéticos, alguns poucos parâmetros, a saber:

- ≠# Trilhas por cilindro (fica implícito o número de superfícies de dado e de cabeças de leitura/gravação);
- ≠# Número de zonas;
- ≠# Para cada zona: cilindros na zona, *track skew* e número de setores por trilha;
- ≠# Tempo de revolução (RPM^{-1});

- ⌘# Tempo de troca de trilha;
- ⌘# Tempo de *full stroke*;
- ⌘# Tempo médio de busca;
- ⌘# Tempo sem atividade antes do disco parar de girar, e tempo de *spin up*;

Todos os tempos em segundos. Esses parâmetros são distribuídos ao longo de poucas dezenas de linhas no arquivo de configuração de discos, chamado *disk.db*. Na Figura 7.2 têm-se o exemplo de linhas de configuração para o disco HP 2247.

name HP2247 tracks per cylinder 13 number of disk zones 8 number of cylinders in zone 533 sectors per track in zone 96 track skew in zone 5 number of cylinders in zone 192 sectors per track in zone 92 track skew in zone 5 number of cylinders in zone 136 sectors per track in zone 88 track skew in zone 5 number of cylinders in zone 144 sectors per track in zone 84 track skew in zone 5 number of cylinders in zone 138	sectors per track in zone 80 track skew in zone 5 number of cylinders in zone 318 sectors per track in zone 72 track skew in zone 5 number of cylinders in zone 268 sectors per track in zone 64 track skew in zone 5 number of cylinders in zone 252 sectors per track in zone 56 track skew in zone 5 revolution time 11.12e-3 1 cylinder seek time 2.5e-3 max stroke seek time 22e-3 average seek time 10.0e-3 time to sleep 1.0e6 time to spinup 0.0
--	--

Figura 7.2 Configuração de disco no RAIDframe

De posse dessas informações, duas possíveis formas de calcular o tempo de busca por um determinado cilindro são a interpolação linear entre o tempo de troca de trilha e o tempo de *full stroke*, discutido na seção 2.1, ou através da equação de Lee (Equação 2.1). O RAIDframe calcula o tempo de busca através de Lee. A determinação dos coeficientes da equação é feito através da função *rf_DiskParam*, no arquivo *rf_geometry.c*, que é chamada no momento em que são lidos os arquivos de configuração que serão utilizados durante a simulação. O atraso rotacional é calculado com base na posição anterior do braço e o *track skew* de cada zona.

7.1.3 Aspectos do simulador

No modo de simulação do RAIDframe são considerados, apenas, aspectos do disco (política de fila, tempos de acesso e de serviço, por exemplo), não sendo contemplados os aspectos de conexão dos discos, como barramentos e interfaces utilizados. Enquanto isso, no modo de prototipação, esses tempos acabam sendo considerados indiretamente, uma vez que os tempos de acesso e serviço do disco são medidos nos discos reais do sistema.

A carga de trabalho utilizada pelo simulador pode ser gerada sinteticamente, ou através da leitura de um arquivo de *trace*. O gerador de carga sintética permite definir várias frações da carga de trabalho total, cada uma delas podendo ser de leitura ou de escrita na matriz, de um determinado tamanho (determinístico ou exponencialmente distribuído com média do tamanho informado) e em uma determinada região da matriz. Essa carga de trabalho é definida através de um arquivo, exemplificado na Figura 7.3

```
Arqui vo: /opt/rai dframe/user/syn. wl
20 r 8 8 e
20 w 8 8 e
30 r 4 8 e 100 10 45
30 w 4 8 e 100 10 45
```

Figura 7.3 Exemplo de carga de trabalho no RAIDframe

A carga de trabalho especificada na Figura 7.3 indica que 20% dos acessos são leituras de blocos com tamanho médio de 8Kb exponencialmente distribuído, e alinhamento⁶ em 8Kb; outros 20% são de escritas com as mesmas características; 30% das operações são leituras por blocos com tamanho médio de 4Kb exponencialmente distribuído, alinhamento em 8Kb, e 100% dessas leituras ocorre numa fração de 10% do disco, situado na região central da matriz (a 45% do início do disco, indo até 55%); por fim, os últimos 30% são escritas com as mesmas características da fração anterior.

Para executar uma simulação utiliza-se o programa `rf_genplot`, disponível no pacote do RAIDframe, e com a seguinte sintaxe:

```
rf_genplot [-p] [-o] [-n] <confi gli st> <workl ist> <output>
```

Onde *configlist*, *worklist* e *output* indicam, respectivamente, os arquivos de configuração da simulação, de carga de trabalho e de saída. A simulação consiste em aplicar cada uma das cargas de trabalho definidas a todas as configurações de matriz especificadas em *configlist*. O resultado é apresentado em um gráfico gerado pelo programa e armazenado em um arquivo texto.

```
Arqui vo: /opt/rai dframe/user/confi gli st
Carga de Trabal ho #1
RAID 5 x RAID 0
0 900 10 60
200 100 5 2.5
/opt/rai dframe/user/confi g5. user
/opt/rai dframe/user/confi g1. user
```

Figura 7.4 Exemplo de arquivo de configuração de simulação no RAIDframe

⁶ O ponto de início da requisição deve ser um múltiplo do valor de alinhamento.

O arquivo de configuração, exemplificado na Figura 7.4, indica o título e subtítulo que serão utilizados no gráfico com os resultados da simulação, os limites dos eixos do gráfico, as distâncias entre as marcações nos eixos x e y, e os arquivos com as matrizes a serem submetidas às cargas de trabalho.

O arquivo especificado nas últimas linhas do exemplo acima são responsáveis pela configuração das matrizes, e segue o formato especificado na Figura 7.5.

```
Arquivo: /opt/raidframe/user/config1.user

# Identifica, na matriz, o número de linhas, colunas, e discos
# de reserva, que são utilizados.
START array
1 4 1

# Especifica o tipo de disco que será utilizado na matriz. Uma
# linha para cada um dos discos especificados. As características
# do disco ficam armazenadas no arquivo disk.db
# Se o arquivo fosse utilizado para prototipação, poderia ser
# utilizado algo como /dev/decz26a no lugar do nome do disco
START disks
HP2247
HP2247
HP2247
HP2247

# Especifica o tipo de disco que será utilizado como reserva na
# matriz.
START spare
HP2247

# A configuração da matriz, com os seguintes parâmetros:
# Setores por unidade de fragmentação
# Unidades de fragmentação por unidade de paridade
# Unidades de fragmentação por unidade de reconstrução
# Identificador do tipo de reconstrução que será usado
# 0 – RAID 0
# 1 – RAID 1
# 4 – RAID 4
# 5 – RAID 5
START layout
64 1 1 1

# Especifica a ordem na qual as requisições serão atendidas, e o
# número de requisições que podem ser emitidas ao mesmo tempo ao
# disco. Os tipos podem ser fifo (first in, first out), cvscan,
# sstf (shortest seek time first), scan (Two-way elevator), cscan
# (One-way elevator)
START queue
fifo 1

# Variáveis de debug
START debug
accessDebug 1
mapDebug 1
dagDebug 1
testDebug 1
```

Figura 7.5 Exemplo de arquivo de configuração de matriz no RAIDframe

O arquivo da Figura 7.5 representa um arranjo de RAID 1 utilizando 4 discos e um de reserva (*spare*), estando todos os discos em uma mesma controladora (número de linhas na seção *array*). O

arquivo *worklist* contém o nome dos arquivos com as cargas de trabalho que serão utilizadas durante a simulação, um por linha.

```
Arquivo: /opt/raidframe/user/worklist
/opt/raidframe/user/syn.wl
/opt/raidframe/user/trace.wl
```

Figura 7.6 Exemplo de arquivo de configuração de cargas de trabalho no RAIDframe

Como resultado, o RaidFRAME oferece, no módulo de simulação, o *throughput* (em entradas e saídas por segundo) e o tempo médio de resposta da organização (em milisegundos), conforme o modelo da Figura 7.7. O mesmo resultado é apresentado através de um gráfico cartesiano armazenado em um arquivo *post script*.

```
Arquivo: /opt/raidframe/user/output
32.83      304.01
110.00     90.39
```

Figura 7.7 Exemplo de arquivo de saída no RAIDframe

7.2 DiskSim – The DiskSim Simulation Environment

DiskSim é simulador de sistemas de disco configurável e eficiente. Foi escrito em C, sem bibliotecas especiais de *software*, durante o doutorado de Gregory Ganger (Ganger, 1995). Ele foi desenvolvido com dois propósitos básicos: avaliar novas arquiteturas e compreender o desempenho de unidades de armazenamento.

Ao contrário do RAIDframe, o DiskSim não possui programas controladores de dispositivos para serem incluídos no sistema operacional, sendo apenas um simulador, capaz de simular, tanto discos magnéticos, quanto RAID 0, 1, 3 e 5 e *parity declustering* (Holland, 1994). A carga de trabalho utilizada pelo simulador pode ser gerada sinteticamente ou lida a partir de um arquivo de *trace*.

Um conceito introduzido pelo DiskSim não presente no RAIDframe é o de organização lógica. Um conjunto de discos arranjados de uma determinada forma é uma organização lógica. Um controlador de dispositivos pode controlar mais de uma organização lógica. Assim, é possível simular o funcionamento de mais de um RAID, ou de um RAID e um disco isolado, ao mesmo tempo e observar o comportamento do sistema computacional como um todo. É possível saber, por exemplo, se o barramento que liga as controladoras tem largura de banda suficiente para todas as organizações lógicas que estão rodando no sistema, bastando para isso, analisar a utilização do barramento em questão.

A implementação do conceito de controlador de dispositivo no DiskSim é diferente do RAIDframe. Enquanto no RAIDframe esse controlador de dispositivo é uma peça de *software* real, que é incorporada ao sistema operacional, no DiskSim ele é um elemento lógico no modelo onde as requisições de leitura e escrita geradas pelos geradores de carga sintética ou arquivo de *trace* chegam. Uma vez lá, são encaminhadas pelos barramentos para a organização lógica correta.

O DiskSim não simula apenas os RAID ou discos, mas sim todo o sistema computacional, ou seja, ele permite avaliar o desempenho dos RAIDs ou discos do ponto de vista dos processos de usuário que estariam executando e os acessando, levando em conta o controlador de dispositivo e as controladoras dos discos.

Enquanto o RAIDframe soluciona os modelos e gera um gráfico comparativo de desempenho (*throughput* e tempo de resposta) entre dois ou mais arranjos, o DiskSim oferece um arquivo texto (cerca de 1.000 linhas para um modelo simples de 3 discos) onde estão as estatísticas dos discos, das organizações lógicas e do controlador de dispositivo para cada um dos arranjos. Cabe ao usuário manipular e coletar os dados de interesse dentro desses arquivos.

A documentação sobre os detalhes de implementação do DiskSim é escassa, quase inexistente e o código fonte não está bem documentado, o que torna a alteração de seu código para a inclusão ou alteração das arquiteturas uma tarefa não trivial. Em compensação, o manual do usuário é suficientemente completo para permitir a criação de arquivos de configuração. No entanto, esse mesmo manual apresenta alguns erros na definição de tipos de dados dos parâmetros (valores que deveriam ser ponto flutuante definidos, no manual, como sendo inteiro).

7.2.1 Representação de RAIDs

No DiskSim, os diversos RAID são representados por linhas de código em C, não havendo nenhuma ferramenta de representação, tal como os DAGs do RAIDframe. A documentação também não faz referência sobre a forma de incluir ou alterar novos arranjos RAID. A parametrização dos RAID é feita em arquivos e configuração que são utilizados como configuração para o simulador.

Além de representar características dos discos, o DiskSim considera as controladoras de disco e os barramentos entre disco e controladora e entre controladora e sistema. As controladoras são configuráveis, sendo possível definir, entre outras, características o tamanho máximo da fila da controladora, tamanho e política de cache e política da fila. Os barramentos também permitem configuração, sendo possível definir, por exemplo, tipo do barramento (exclusivo – IDE – ou banda compartilhada – SCSI), velocidade e tipo de arbitragem (FCFS ou com prioridade).

Figura 7.8 Exemplo de organização física dos elementos no DiskSim

É possível, com o DiskSim, fazer qualquer arranjo físico entre os elementos disco, controladora e barramento, o que o torna bastante flexível. É possível definir até 15 discos ou controladoras por barramento. O elemento de mais alto nível da organização física é o controlador de dispositivo. A definição das especificações de conexão devem ser feitas na ordem controlador de dispositivo, controladora de disco e barramento. A Figura 7.8 mostra um exemplo de organização válida e as linhas necessárias no arquivo de configuração do DiskSim para representar tal organização. Essa organização pode conter uma ou mais organizações lógicas, que são definidas separadamente (discos 1 e 2 formam um RAID 0, disco 3 e 4 trabalham como discos independentes, por exemplo)

7.2.2 Modelagem de discos magnéticos

Os discos magnéticos são os componentes mais detalhados do DiskSim. O número de características que podem ser configuradas passa de 100, distribuídas ao longo de aproximadamente 260 linhas para um disco simples com 9 zonas, estando incluídas aí, características específicas do mecanismo e cache interno do disco, além da conexão entre disco e barramento da controladora. Um exemplo de arquivo de configuração está presente no Anexo C.

O modelo de disco do DiskSim é bastante completo, adicionando os seguintes parâmetros aos especificados pelo RAIDframe:

- ## Tempo adicional de posicionamento para escrita;
- ## Arquivo com a curva do tempo de busca;
- ## Tempos de busca por 1 a 10 cilindros;
- ## Tempo de troca de cabeça;
- ## *Cylinder skew* de cada zona;
- ## Se o disco exige ou não exclusividade do barramento;
- ## Aspectos de cache interno do disco;

- ≠# Definição de áreas com defeito no disco;
- ≠# Realocação de setores/trilhas defeituosos em outras áreas do disco;

Esses parâmetros permitem modelar os tempos de busca e serviço dos discos com alta precisão, superior a 98% no pior caso segundo experimentos conduzidos pelos autores do mesmo.

Enquanto o tempo de busca por um cilindro no RAIDframe é estimado através da Equação 2.1, de Lee, no DiskSim ele pode ser estimado de 5 formas diferentes, definidas no arquivo de configuração de cada disco: interpolação linear entre os tempos de busca pelo próximo cilindro, tempo médio de busca, e tempo de *full stroke*; através da equação de Lee; através da equação Equação 2.4, de Ruemmler, ou; através de interpolação linear no arquivo com a curva do tempo de busca.

Se a Equação 2.4 for escolhida para estimar o tempo de busca, devem ser fornecidos os coeficientes. Para buscas até 10 cilindros são utilizados os valores de tempo de busca informados e, para buscas maiores que 10 cilindros, utiliza-se a Equação 2.4.

O atraso rotacional pode ser determinado com base na posição anterior do braço e dos valores de *track skew* e *cylinder skew* de cada zona, tal como no RAIDframe, ou pode ser definido como metade do tempo de revolução do disco.

7.2.3 Aspectos do simulador

Ao contrário do RAIDframe, o DiskSim considera, além das características intrínsecas do disco, as características dos elementos de conexão entre disco e sistema computacional: controladoras e barramentos.

A carga de trabalho para o simulador pode ser gerada sinteticamente ou através de arquivos de *trace*. No caso da geração sintética de carga de trabalho, podem ser definidos vários elementos geradores de carga de trabalho. Cada elemento se comporta como um pequeno processo em nível de usuário emitindo requisições após um “tempo de pensar” (*think time*) e, quando for o caso, esperando que elas se completem. As requisições podem ser configuradas quanto à localidade espacial no disco e quanto ao tempo entre chegadas.

Cada elemento pode ser configurado nos seguintes aspectos:

- ≠# Maior número de setor que pode ser requisitado;
- ≠# Probabilidade de acesso seqüencial, local e, implicitamente, aleatório;
- ≠# Probabilidade de acesso de leitura/escrita;

- ≠# Probabilidade de requisições de tempo crítico (bloqueiam o processo gerador até que sejam completadas) ou tempo limitado (o processo espera por um determinado tempo e depois continua, desde que a requisição já tenha sido atendida; caso contrário, espera pela requisição) e, implicitamente, independentes;
- ≠# Tamanho da requisição

Cada uma dessas probabilidades de tempo (crítico, limitado ou independente), além da definição da área considerada “local” no disco e do tamanho da requisição é definida através de uma distribuição de probabilidade uniforme, normal, exponencial ou poisson.

Um exemplo de arquivo de configuração do DiskSim representando um arranjo RAID 0 de 3 discos é apresentado no Anexo C.

Os resultados obtidos pelo DiskSim, também listados no Anexo C, quando simulando discos magnéticos se comportaram dentro do previsto na literatura mas, a simulação de RAID apresentou algumas discrepâncias em relação à literatura. Algo que merece destaque é o resultado de simulações de RAID 0 e RAID 5, com a mesma carga de trabalho predominantemente de pequenas escritas e o mesmo custo (número de discos). Segundo a literatura, o comportamento deve ser o apresentado na Figura 7.9a.

Figura 7.9 *Throughput* máximo por dólar versus tamanho do grupo

Como pode ser observado, ao aumentar o tamanho do grupo de paridade (número de discos na matriz), o desempenho do RAID 5 deve reduzir, em relação ao desempenho do RAID 0. O comportamento encontrado no DiskSim foi exatamente o oposto, como pode ser visto na Figura 7.9b.

Pode-se perceber que o desempenho de uma matriz RAID 5 no DiskSim não está se comportando como esperado. Os autores do DiskSim foram contatados várias vezes durante o desenvolvimento deste trabalho com o intuito de compreender o comportamento apresentado mas, em

todas as vezes, a resposta foi de que, *infelizmente*, eles não podiam ajudar, conforme mensagem apresentada no Apêndice A.

7.3 Aplicabilidade ao Ambiente proposto

Como descrito anteriormente, o objetivo principal do Ambiente é permitir ao Estudante experimentar modelos RAID definidos pelo Tutor Humano, definir seus próprios modelos, e aprender com os resultados obtidos pela simulação desses modelos. Para que isso seja possível, o simulador utilizado pelo Ambiente deve permitir essa configuração da forma mais transparente possível ao Estudante. Além disso, essas alterações não devem atingir o código do simulador, permanecendo na esfera de arquivos de configuração ou, no máximo, parâmetros de execução do simulador.

Nos dois simuladores estudados, a inclusão ou alteração de arquiteturas provoca a alteração do código fonte do simulador, ocasionando a necessidade de uma recompilação. Isso é uma característica que depõe contra a aplicabilidade dos dois simuladores ao Ambiente. É importante considerar, também, a documentação disponível relativa a implementação do simulador. A documentação do RAIDframe é farta e o código bem documentado, enquanto que a documentação do DiskSim é escassa, e o código mal documentado.

A flexibilidade e completude dos modelos necessária ao Ambiente, uma vez definida a arquitetura, foi encontrada no DiskSim. Através dele é possível ajustar desde pequenos detalhes, como política de substituição dos elementos no *cache* interno do disco, até aspectos mais macros, como tipo e velocidade dos barramentos que conectam os discos às controladoras. Para alterar essas características, basta ajustar o arquivo de configuração, sem necessidade de alteração de código fonte. No RAIDframe, a alteração dessas características também é feita através de arquivos de configuração, mas o número de opções disponíveis é bem menor que o DiskSim.

A facilidade de geração da carga sintética é outro ponto que deve ser considerado na escolha do simulador que será utilizado no Ambiente. Os dois simuladores estudados podem trabalhar com arquivos de *trace*, mas isso limita as alterações que o Estudante pode fazer na carga de trabalho, provocando a necessidade de vários arquivos de *trace* com características diferentes. O problema é que, para ter um conjunto de requisições de entrada e saída suficientemente grande para o simulador, é necessário ter um arquivo de *trace* também grande, que ocupa espaço em disco. A alternativa aos arquivos de *trace* é a geração de carga de trabalho sintética e, nesse aspecto, o DiskSim é mais flexível que o RAIDframe.

No DiskSim é possível definir sistemas abertos (requisições independentes umas das outras), fechados (requisições totalmente dependentes umas das outras), ou sistemas mistos (uma fração das

requisições é independente e outra fração é dependente). Outro ponto é o número de geradores de carga que podem ser definidos. No RAIDframe só é possível definir um gerador de carga por simulação, enquanto que no DiskSim esse número está limitado a 128. Assim, é possível gerar cargas de trabalho complexas que podem representar bem as mais variadas situações.

Outro ponto que deve ser considerado na escolha do simulador é a precisão dos resultados oferecidos. O DiskSim apresenta uma excelente precisão na simulação de discos isolados, mas o comportamento das simulações de RAID executadas durante o período de testes é um ponto negativo contra ele. Os resultados do RAIDframe não apresentaram grandes variações. As percebidas podem ser atribuídas ao modelo de disco muito simples adotado por ele e que pode afetar os resultados da matriz como um todo.

Deve-se considerar, ainda, a quantidade de resultados oferecidos. Enquanto o RAIDframe informa, apenas o número de requisições por segundo atendidas pela matriz de discos e o tempo médio de resposta da mesma, o DiskSim oferece medidas detalhadas de todos os componentes da matriz, desde os discos, até o controlador de dispositivos de nível mais alto e são dadas informações como tamanho e tempo médio de fila, tempo de resposta e utilização de cada um dos elementos presentes no modelo. Com essas informações, o Estudante é capaz de, por exemplo, descobrir pontos de estrangulamento no desempenho da matriz. A desvantagem direta da quantidade de informações oferecidas pelo DiskSim é o problema de localizar a informação desejada entre as centenas de linhas de texto oferecidas como resultado. O RAIDframe tem uma abordagem para a apresentação dos resultados mais próxima do desejado, oferecendo um gráfico. A Tabela 7.1 apresenta um quadro comparativo entre os dois simuladores, atribuindo conceitos, de fraco a excelente, às características de cada um.

<i>Característica</i>	<i>RAIDframe</i>	<i>DiskSim</i>
Documentação externa	Excelente	Boa
Documentação interna	Excelente	Razoável
Facilidade de inclusão da arquiteturas	Boa	Razoável
Flexibilidade de configuração	Razoável	Excelente
Compleitude	Razoável	Excelente
Caracterização de carga de trabalho	Boa	Excelente
Precisão dos resultados	Boa	Boa a Excelente
Quantidade de resultados	Fraca	Excelente
Facilidade de extração dos resultados	Excelente	Razoável

Tabela 7.1 Comparação entre dos simuladores RAIDframe e DiskSim

O Ambiente proposto nesta dissertação não exige exclusividade de simulador, como já foi discutido. É possível, inclusive, substituir o simulador durante a execução do Ambiente conforme a necessidade do Estudante. Quando for necessário ter uma informação detalhada sobre um disco, por exemplo, ele pode utilizar o DiskSim e quando estiver interessado apenas em comparar os tempos de resposta de duas matrizes, pode utilizar o RAIDframe.

No Simulador ideal para o Ambiente de Simulação e Aprendizado para RAID deve ser fácil incluir uma arquitetura, preferencialmente sem necessidade de recompilar o simulador. Deve haver várias opções de configuração do arranjo, permitindo configurar detalhadamente discos e interfaces. As opções de geração de carga de trabalho devem ser fartas, para permitir a representação e estudo de comportamento do arranjo em várias situações. Deve oferecer resultados precisos e detalhados sobre os vários componentes, mas a obtenção e extração desses resultados deve ser uma tarefa fácil. Além disso, a documentação do Simulador deve ser farta e precisa, para que ele possa ser estendido e alterado com facilidade quando se fizer necessário.

Apesar de os simuladores estudados poderem ser utilizados eventualmente no Ambiente proposto, nenhum deles apresenta todas as características desejáveis, principalmente no tocante à inclusão de arquiteturas novas e precisão dos resultados. Desta forma, uma abordagem alternativa à seleção de um Simulador disponível é o desenvolvimento de um novo Simulador de RAID, com o melhor de cada um dos simuladores estudados. Neste caso, o novo simulador seria considerado o simulador padrão do Ambiente proposto e, se o usuário (Estudante ou Tutor Humano) desejar, pode trocá-lo por outro.

O capítulo seguinte faz algumas considerações sobre a construção de um simulador de discos magnéticos que pode ser expandido e estendido para se tornar o Simulador de RAID padrão para o Ambiente de Simulação e Aprendizado Inteligente para RAID descrito no Capítulo 6.

Capítulo 8

Protótipo de simulador de discos magnéticos

Com base nos parâmetros sugeridos como suficiente para modelar um disco com precisão suficiente, apresentados na seção 6.2.1, apresenta-se neste capítulo um protótipo de simulador de discos magnéticos que, quando expandido e estendido para simular RAID, pode ser adotado como simulador padrão para o Ambiente proposto nesta dissertação.

Como foi observado na seção 7.3, os simuladores estudados possuem algumas características que são desejáveis em um simulador que será utilizado pelo Ambiente. Entretanto, a inclusão ou alteração de uma arquitetura, nos dois simuladores, envolve alteração de código fonte e recompilação do simulador, tornando a tarefa não-trivial, e afastando o simulador do usuário típico do Ambiente. Além disso, em um dos simuladores os resultados disponíveis são escassos, enquanto em outro os resultados apresentaram algumas discrepâncias em relação ao previsto na literatura.

Como já afirmado, uma possibilidade é a implementação de um simulador próprio para o Ambiente, onde seja possível alterar as arquiteturas através de arquivos de configuração, sem necessidade de recompilação do simulador, ou mesmo geração de código em C/C++. Além disso, o Simulador deve oferecer resultados precisos e estar bem documentado, para permitir a expansão e extensão do mesmo.

A base para um simulador de RAID é um simulador de discos eficaz e preciso. Sobre este simulador, acrescenta-se uma camada de *software* para representar, e simular, a controladora de discos, como ilustrado na Figura 8.1, e outra responsável por implementar a arquitetura em questão, ou seja, decidir quais fragmentos vão para qual disco da camada inferior, como calcular a redundância, onde armazená-la, e como proceder no caso de falha de um dos discos.

Figura 8.1 Camadas de um simulador de RAID

O restante deste capítulo apresenta um protótipo de simulador de discos magnéticos que pode ser utilizado como base para o desenvolvimento de um Simulador de RAID para o Ambiente.

8.1 Modelagem e implementação

De acordo com o apresentado na seção 6.2.1, definiu-se os parâmetros necessários para modelar satisfatoriamente um disco magnético. Decidiu-se implementar um programa de simulação em C++ utilizando a biblioteca de simulação discreta orientada a eventos SSS (Pollatschek, 1995) para simular o comportamento de um disco magnético. O simulador aqui proposto pode ser expandido e estendido para simular RAID e se tornar o simulador padrão do Ambiente. A escolha pelo SSS se deu devido a presença de um maior número de ferramentas para coleta e análise estatísticas dos dados da simulação. O SSS pode coletar e analisar praticamente qualquer variável. Isso permitiu coletar dados separados sobre tempo de busca, tempo de serviço e tempo de resposta das requisições sem necessidade de escrever linhas de código que efetuassem a coleta e análise dos dados.

O modelo inicial foi concebido utilizando a notação de Rede de Filas de acordo com a Figura 8.2. Requisições chegam no centro de serviço com uma determinada taxa e são (1) atendidas, consumindo um determinado tempo de serviço ou (2) enfileiradas para atendimento posterior. A disciplina da fila é FIFO (*First In, First Out*). O tempo de serviço no servidor é dado pelo tempo de busca acrescido do atraso rotacional e do tempo necessário para transferir os dados que foram solicitados. São coletados dados para geração de estatísticas sobre tempo de busca, tempo de serviço e tempo de resposta (tempo de serviço mais tempo de fila).

Figura 8.2 Modelo de rede de filas para um disco magnético

Para a geração do arcabouço do programa de simulação, decidiu-se utilizar o Ambiente de Simulação Automático, proposto pelo Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP (Spolon, 1994; Santana et al., 1996; Oliveira, 1998). O programa obtido é apresentado no Anexo B.1.

O programa foi alterado para refletir as características de tempo de serviço do disco, para incluir a coleta de dados para a elaboração das estatísticas e para utilizar a biblioteca SSS em lugar da SMPL. Optou-se por fazer um programa de simulação que fosse capaz de ler arquivos de configuração gerados em ASCII, onde são informados:

Tempo de simulação e número máximo de requisições (condições de parada);

- ## Parâmetros da distribuição de chegada de clientes, setor a ser lido, e quantos setores a partir do setor inicial;
- ## Parâmetros do disco, como tempo de troca de trilha, *full stroke*, troca de cabeça e zonas, entre outros.

Nos eventos de chegada de cliente, a distribuição exponencial utilizada no programa gerado pelo ASiA foi substituída pela seleção da distribuição com base no arquivo de configuração lido pelo simulador. A opção pela geração de arquivos de configuração separados foi decorrência da necessidade de um maior grau de parametrização sem necessidade de alteração do código fonte do simulador. Isso permitiu, ainda, que as simulações fossem executadas em lote em momentos de ociosidade das máquinas do Laboratório de Sistemas Distribuídos e Programação Concorrente do ICMC-USP. O tempo de busca, nessa primeira versão, pode ser calculado pela Equação de Lee (Equação 2.1), ou através de interpolação linear em uma tabela de tempos de busca conhecidos para o disco.

Os setores foram mapeados nas superfícies de forma vertical (Figura 2.4), que é a forma mais comum nos discos atuais. A troca da forma de mapeamento dos setores no disco envolve pequenas alterações do código fonte na seção em que um determinado setor é convertido (ou “quebrado”) em zona, cilindro, superfície e setor. Dessa forma, é perfeitamente possível implementar as várias formas de mapeamento de setor no programa de simulação e, através de um parâmetro no arquivo de configuração do disco, selecionar qual deve ser utilizada.

O atraso rotacional foi calculado através de uma distribuição aleatória uniforme entre 0 e o tempo de rotação do disco, o que corresponde a um atraso rotacional médio igual a metade do tempo de rotação. Por isso não foram considerados o deslocamento de trilha e de cilindro, que só têm sentido quando se modela o atraso rotacional. A inclusão do atraso rotacional no programa de simulação é discutida, entre outros, em Ruemmler et al., 1994. O programa de simulação resultante, apresentado no Anexo B.2, é executado com a seguinte linha de comando:

```
C: \>simulador <input> <seed> <output> <seek?> <debug> [seekfile]
```

Onde:

- ## *Input*: arquivo com as configurações do disco;
- ## *Seed*: semente do gerador de números aleatórios;
- ## *Output*: arquivo onde o resultado da simulação será armazenado, exemplificado no Anexo B.3;

- ## *Seek?*: Determina como será calculado o tempo de busca. (1) para Lee e (2) para valores tabelados;
- ## *Debug*: arquivo onde serão gravadas informações sobre cada uma das requisições emitidas pelo simulador, conforme exemplo apresentado no Anexo B.4;
- ## *Seekfile*: parâmetro obrigatório se <seek?> igual a 2. Contém os valores tabelados de tempo de busca. Tempos para distâncias ausentes serão calculados através de interpolação linear.

8.2 Testes efetuados e resultados obtidos

Para testar o programa de simulação, foram criados arquivos de configuração para representar as características dos discos **Quantum QM39100TD SW** (chamado ao longo desta e das próximas seções de *Disco 1*) e **DEC RZ26** (*Disco 2*). Enquanto o Disco 1 foi criado no fim de 1998, o Disco 2 é do início dos anos 90, o que faz com que tenham a curva de tempo de busca bem diferentes. Além disso, o Disco 1 possui 14 zonas, com 256 a 168 setores por trilha, enquanto o Disco 2 possui apenas uma zona com 58 setores por trilha. Os parâmetros dos discos foram obtidos por Schindler & Ganger através da ferramenta de extração automática de parâmetros *DIXtrac* (Schindler et al., 1999).

Durante a parametrização do Disco 1, um parâmetro importante não estava disponível entre os coletados por Schindler & Ganger: o tempo médio de busca (*avSeek*). A falta deste parâmetro pode ser justificada pela coleta de valores tabelados de tempo de busca. Esses valores também podem ser utilizados para determinar o tempo de busca através de interpolação, como já foi discutido

O protótipo de simulador proposto pode calcular o tempo de busca através da Equação 2.1 e, para isso, o tempo *avSeek* é necessário. Durante a revisão bibliográfica efetuada para esta Dissertação, algumas formas de cálculo de *avSeek* foram encontradas, entre elas, a média simples dos tempos conhecidos para várias distâncias (Equação 2.2) e a média dos tempos de busca ponderada pelo número de buscas possíveis para uma determinada distância (Equação 2.3). Além disso, *avSeek* pode ser definido como um terço do tempo de *full stroke* ou, ainda, o tempo de busca por um terço do número de cilindros do disco. A Tabela 8.1 apresenta os valores encontrados para *avSeek* utilizando-se cada uma das formas acima.

Média simples	10,148904ms
Média ponderada	5,859658ms
1/3 tempo de <i>full stroke</i>	5,120000ms
Tempo de busca por (<i>numCyl</i> /3)	8,701500ms

Tabela 8.1 Tempo médio de busca para Disco 1

Devido à diferença superior a 69% entre o menor e o maior tempo encontrado, optou-se por calcular as três constantes para a Equação de Lee, um conjunto para cada valor de *avSeek*, e verificar qual conjunto gerava uma curva mais próxima da curva obtida com os tempos medidos. Os valores obtidos para cada constante (*a*, *b* e *c* da Equação 2.1) são apresentados na Tabela 8.2.

A Figura 8.3 mostra as curvas obtidas com cada conjunto de constantes e, também, a curva obtida com os valores tabelados.

	<i>a</i>	<i>b</i>	<i>c</i>
Média simples	0,216273	-0,000709	1,663000
Média ponderada	-0,020555	0,001929	1,663000
1/3 tempo de <i>full stroke</i>	-0,061757	0,002388	1,663000
Tempo de busca por (<i>numCyl</i> /3)	0,137746	0,000165	1,663000

Tabela 8.2 Constantes de Lee variando-se o tempo médio de busca do Disco 1

Considerando-se que os valores tabelados para o Disco 1 apresentam uma boa precisão, uma vez que são valores medidos diretamente com o disco em operação normal, pode-se perceber, mesmo empiricamente, que as curvas obtidas com a Equação 2.1 considerando-se *avSeek* igual a média ponderada, média simples ou um terço do tempo de *full stroke* são bem diferentes da curva real para o Disco 1. Assim, a melhor estimativa o valor de *avSeek* para o Disco 1 é considerá-lo como o tempo de busca por um terço do número de cilindros. Além disso, é interessante observar que, para esse caso, todas as constantes são positivas, condição desejável segundo Lee, 2000.

Figura 8.3 Curvas de tempo de busca para o Disco 1 variando-se o cálculo do tempo médio

De posse dessa informação parametrizou-se o Disco 1 com base nas características coletadas por Schindler & Ganger e *avSeek* igual ao tempo de busca por um terço do número de cilindros. Foram definidas três cargas de trabalho: tempo entre chegadas exponencialmente distribuído com

médias 15, 17 e 20ms e tamanho da requisição exponencialmente distribuído com média de 5 setores. As requisições são uniformemente distribuídas ao longo do disco. As simulações foram executadas variando-se a semente do gerador de números aleatórios e a forma de cálculo do tempo de busca (através da equação de Lee ou por interpolação linear). O Anexo B.5 apresenta, como exemplo, o arquivo de configuração utilizado pelo Simulador para o Disco 1 onde o tempo entre chegadas é exponencialmente distribuído com média 20. Foram coletados o tempo médio de resposta, o tempo médio de busca e o número de requisições atendidas por segundo. Os resultados estão representados na Tabela 8.3, com intervalo de confiança de 95%.

<i>Tempo entre requisições</i>		<i>Exponencial com média de 15ms</i>	<i>Exponencial com média de 17ms</i>	<i>Exponencial com média de 20ms</i>
Lee	ES/s	66,57 $\pm$ 0,28	59,32 $\pm$ 0,35	50,35 $\pm$ 0,09
	Busca (ms)	9,51 $\pm$ 0,01	9,55 $\pm$ 0,01	9,55 $\pm$ 0,00
	Resposta (ms)	156,19 $\pm$ 11,64	55,40 $\pm$ 2,11	33,67 $\pm$ 0,41
Tabelado	ES/s	67,04 $\pm$ 0,14	59,17 $\pm$ 0,04	49,95 $\pm$ 0,20
	Busca (ms)	9,13 $\pm$ 0,01	9,14 $\pm$ 0,01	9,14 $\pm$ 0,00
	Resposta (ms)	96,43 $\pm$ 4,49	47,07 $\pm$ 1,20	29,75 $\pm$ 0,42

Tabela 8.3 Resultados de simulação do Disco 1

A diferença no tempo médio de resposta é consequência direta da diferença dos tempos de busca. Os valores obtidos através da Equação 2.1 para o Disco 1 são, em sua grande parte, maiores que os tempos reais de busca, como pode ser visto na Figura 8.3. Essa diferença é maior quando o tempo entre chegadas é menor pois, levando mais tempo na busca, o tempo de serviço aumenta e, por consequência, aumenta a fila na entrada do disco. Com filas maiores e tempo de serviço independente do tamanho da fila ou do tempo entre chegadas, os tempos de resposta tendem a ser maiores. É interessante observar, também, que os tempos de busca tendem a se manter constante pois dependem unicamente da distribuição espacial das requisições sobre o disco, que são iguais nas 6 situações apresentadas: uniformemente distribuída ao longo dos setores do disco.

Para o Disco 2, o valor de *avSeek* também estava indisponível. Optou-se, tal como para o Disco 1, testar as quatro possíveis formas de cálculo do mesmo e comparar os resultados. Os valores obtidos estão representados na Tabela 8.4.

Média simples	12,721091ms
Média ponderada	6,220502ms
1/3 tempo de <i>full stroke</i>	6,366266ms
Tempo de busca por (<i>numCyl</i> /3)	10,898045ms

Tabela 8.4 Tempo médio de busca para Disco 2

A diferença entre o menor e o maior tempo é, agora, superior a 100%. Novamente optou-se por calcular a tripla de constantes e traçar as curvas para observar o comportamento das mesmas frente aos valores tabelados. Os valores obtidos para as constantes estão na Tabela 8.5.

	<i>a</i>	<i>b</i>	<i>c</i>
Média simples	0,556936	-0,003966	1,014397
Média ponderada	-0,080622	0,008540	1,014397
1/3 tempo de <i>full stroke</i>	-0,066326	0,008259	1,014397
Tempo de busca por (<i>numCyl</i> /3)	0,378137	-0,000459	1,014397

Tabela 8.5 Constantes de Lee variando-se o tempo médio de busca do Disco 2

É interessante observar que, para o Disco 2, nenhuma das triplas de constante apresenta todos os valores estritamente positivos. Lee afirma que essa é uma condição desejável, mas não necessária para a utilização da Equação 2.1 (Lee, 2000). Mesmo com elementos negativos nas triplas, pode-se perceber na Figura 8.4 que, novamente, a curva da Equação 2.1 que mais se aproxima da curva obtida com os valores tabelados é aquela parametrizada pela triplas obtidas com *avSeek* igual ao tempo de busca por um terço do número de cilindros.

Figura 8.4 Curvas de tempo de busca para o Disco 2 variando-se o cálculo do tempo médio

Como o Disco 2 tem um tempo de busca médio maior que o Disco 1 (10,9ms contra 8,7ms), e consequentemente um tempo de serviço maior para uma requisição com as mesmas características, as cargas de trabalho aplicadas ao Disco 1 saturam o Disco 2 logo no início da simulação. Assim optou-se por definir novas cargas de trabalhos para o Disco 2, mais “leves”, aumentando o tempo médio entre chegadas.

Desta forma as cargas de trabalho aplicadas ao Disco 2 foram definidas com o tempo entre chegadas exponencialmente distribuído com médias 20, 25 e 30ms e tamanho da requisição

exponencialmente distribuído com média de 5 setores. A distribuição das requisições continuou uniforme ao longo do disco.

Tal como no Disco 1, as simulações foram executadas variando-se a semente do gerador de números aleatórios e a forma de cálculo do tempo de busca (através da equação de Lee ou por interpolação linear). Os resultados obtidos estão apresentados na Tabela 8.6, com intervalo de confiança de 95%.

<i>Tempo entre requisições</i>		<i>Exponencial com média de 20ms</i>	<i>Exponencial com média de 25ms</i>	<i>Exponencial com média de 30ms</i>
Lee	ES/s	50,16 $\pm$ 0,06	40,03 $\pm$ 0,03	33,44 $\pm$ 0,03
	Busca (ms)	12,21 $\pm$ 0,03	12,21 $\pm$ 0,00	12,20 $\pm$ 0,00
	Resposta (ms)	122,82 $\pm$ 6,05	44,71 $\pm$ 0,05	33,54 $\pm$ 0,03
Tabelado	ES/s	50,07 $\pm$ 0,36	40,09 $\pm$ 0,07	33,26 $\pm$ 0,08
	Busca (ms)	11,30 $\pm$ 0,01	11,32 $\pm$ 0,01	11,30 $\pm$ 0,00
	Resposta (ms)	78,11 $\pm$ 3,48	38,55 $\pm$ 0,10	29,90 $\pm$ 0,07

Tabela 8.6 Resultados de simulação do Disco 2

Tal como no Disco 1, os valores obtidos através da Equação 2.1 para o Disco 2 são, em sua grande parte, maiores que os tempos reais de busca, como pode ser visto na Figura 8.4, justificando a diferença no tempo médio de busca encontrado durante a simulação. Os tempos médios de busca também se mantêm constantes pois, tal como no Disco 1, as cargas de trabalho variam, mas o deslocamento médio dos braços imposto pelas mesmas é igual nas três cargas aplicadas.

8.3 Limitações do protótipo

Neste primeiro protótipo a carga de trabalho foi definida como 100% de leitura. Determinar qual fração da carga de trabalho é leitura, e qual é escrita passa pela inclusão de mais um parâmetro na configuração do Simulador e inclusão de menos de uma dezena de linhas de código na seção de leitura de parâmetros e de determinação do tempo de serviço.

Na versão atual é possível definir apenas uma carga de trabalho, e todas as requisições geradas são independentes entre si, ou seja, a emissão de uma segunda requisição é independente do fato de a primeira requisição já ter sido atendida.

O protótipo de simulador de discos magnéticos possui uma boa precisão, mas esta pode, e deve, ser aumentada na versão final que fará parte do Simulador de RAID. Hoje, a precisão do tempo de serviço é maior para pequenas requisições. Se forem solicitados, por exemplo, 500 setores de uma trilha de um disco com 300 setores por trilha, 200 setores da requisição estarão, ou no próximo

cilindro ou na próxima superfície, dependendo da forma de mapeamento dos setores no disco. Nesses casos, onde uma requisição lê além da fronteira de uma trilha, é necessário acrescentar ao tempo de serviço os tempos de troca de cabeça e trilha que por ventura ocorram. Esta primeira versão do simulador não considera esses tempos.

Como já discutido no início do capítulo, o determinismo do atraso rotacional pode ser incluído no modelo do disco e, por consequência no programa de simulação, através de variáveis como o deslocamento de trilha e cilindro (*track and cylinder skew*) e a quanto tempo o disco está girando.

8.4 Considerações finais

A decisão pela implementação de um simulador de RAID, desde as suas bases, para ser utilizado no Ambiente de Simulação e Aprendizado Inteligente para RAID proposto nesta dissertação, é uma alternativa viável e que não deve ser descartada. A principal vantagem dessa abordagem em detrimento da utilização de um ou outro simulador de RAID disponível é que um simulador construído com esse propósito específico conterá todas as características desejáveis em um simulador que será utilizado pelo Ambiente, como flexibilidade e facilidade de configuração por arquivos externos.

Além disso, o simulador será desenvolvido dentro do grupo de trabalho que o utilizará e, dessa forma, pode ser facilmente alterado ou ajustado em funções de inclusão ou remoção de características de projeto do Ambiente proposto, pois o código estará disponível e devidamente documentado.

A base para o simulador de RAID que pode vir a ser implementado, como já afirmado no início desse capítulo, é o simulador de discos magnéticos. Uma vez definido e implementado o simulador de discos magnéticos, as tarefas restantes para a implementação efetiva do simulador de RAID são a implementação das camadas da controladora e da arquitetura. O protótipo aqui apresentado é um simulador de discos suficientemente preciso e confiável que pode ser utilizado por esse simulador.

Esta página foi deixada intencionalmente em branco.

Capítulo 9

Observações Finais e Conclusões

Este capítulo apresenta as conclusões desta dissertação. Inicialmente são feitas considerações sobre a revisão bibliográfica. Em seguida são apresentadas as conclusões, contribuições e sugestões de trabalhos futuros.

9.1 Considerações sobre a revisão bibliográfica

Como discutido na seção 2.1, os discos magnéticos apresentaram um aumento de desempenho, ao longo do tempo, limitado por componentes mecânicos. Uma forma de suplantar as limitações é a criação de matrizes de discos. Junto com o aumento de desempenho, as matrizes trazem problemas da confiabilidade, que é inversamente proporcional ao número de discos na matriz. Como forma de tentar aliar desempenho e confiabilidade surgem os RAIDs, introduzindo o conceito de redundância na matriz de discos. As formas como os dados são distribuídos nos discos e a redundância é calculada dão origem aos níveis RAID.

São vários os níveis RAID, e cada um com suas peculiaridades de desempenho e confiabilidade, alguns prezando por um desempenho maior em detrimento da confiabilidade, alguns dando preferência à confiabilidade em detrimento do desempenho. O estudo e avaliação das diferentes configurações de RAID é uma tarefa não-trivial, principalmente para um usuário não familiarizado com a terminologia adotada para distinguir as diferentes formas de organização da matriz de discos.

A computação oferece ferramentas que permitem auxiliar o ensino de certas áreas do conhecimento. Uma dessas técnicas é a utilização de Ambientes Inteligentes de Simulação e Aprendizado (ISLEs) que permitem, ao mesmo tempo, simular o funcionamento de um modelo e transmitir um determinado conhecimento ao usuário.

Este trabalho propõe a utilização da capacidade de simulação oferecida pelos ISLEs para a avaliação de desempenho de RAIDs. Os aspectos de transmissão do conhecimento são utilizados para auxiliar no aprendizado da taxonomia e características de desempenho dos RAIDs.

9.2 Conclusões e contribuições

A linha de pesquisa desta dissertação reforça e expande as linhas de pesquisa de avaliação de desempenho em sistemas computacionais e simulação sequencial do Grupo de Sistemas Distribuídos e

Programação Concorrente do ICMC-USP, fazendo-as atuar na área de subsistemas de entrada e saída. Além disso, tenta agregar à elas os aspectos de ferramentas de auxílio ao aprendizado e, assim, abrir uma nova linha de pesquisa para o Grupo.

O enfoque principal deste trabalho é demonstrar como um ambiente de simulação e aprendizado inteligente pode ser utilizado para auxiliar no ensino de RAID. A área em questão foi escolhida exatamente por sua taxonomia relativamente complexa e por ser uma área que se beneficia diretamente da simulação para avaliação de desempenho. A construção física de matrizes de disco para efetuar as medidas de desempenho *in loco* é uma atividade cara, demorada e complexa. A simulação pode resolver o problema de forma rápida, fácil e menos onerosa.

Na ferramenta proposta, o estudante, através de simulações, exercícios e um ambiente onde é possível sua interferência no fluxo e na forma de transmissão do conhecimento, analisa os diferentes tipos de RAID, sua arquitetura, desempenho e confiabilidade. Assim, o usuário pode formular hipóteses, experimentar parâmetros e observar o comportamento da matriz de discos, validando ou não suas hipóteses e aprendendo de forma construtivista.

Como consequência direta da modularização do ambiente proposto tem-se a aplicabilidade do mesmo a outras áreas (como por exemplo redes de computadores) com alterações localizadas nos módulos que manipulam o conhecimento. Ao aplicar o Ambiente a uma nova área, é necessário estudar novos simuladores para a área em questão e alterar a classe de informações armazenadas nos depósitos de dados, mas a estrutura do Ambiente permanece inalterada.

Para o desenvolvimento de uma ambiente desse tipo, utilizam-se conceitos de diversas áreas da computação:

- ≠# **Inteligência computacional** e suas ferramentas de auxílio ao aprendizado, como os sistemas tutores e ambiente de aprendizado e simulação inteligente, utilizada para a especificação dos meta-módulos do ambiente proposto;
- ≠# **Simulação** e sua propriedade de avaliar desempenho de sistemas computacionais sem a necessidade da presença física dos mesmos, reduzindo os custos e a complexidade do problema, aplicada durante o estudo dos simuladores e a implementação do protótipo de simulador de discos magnéticos;
- ≠# **Hipermídia**, utilizada na construção dos documentos multimeios que armazenam o conhecimento, será importante no momento da implementação real do ambiente
- ≠# Sistemas de **bancos de dados** e suas técnicas de armazenamento de dados não triviais, como documentos multimeios, serão de extrema importância no momento de decidir como armazenar os documentos gerados pelo ambiente, e;

O **projeto de interfaces** da engenharia de software será fundamental para a análise dos aspectos de ergonomia e IHM da interface para o ambiente.

Um enfoque especial foi dado para a escolha de um simulador de RAIDs para o ambiente proposto. Tanto o RAIDframe quanto o DiskSim, os dois simuladores estudados, apresentam a propriedade de permitir a avaliação de desempenho de RAID. Apesar de diferirem na abordagem dada à solução do problema (enquanto um é um prototipador, o outro é apenas um simulador), os dois possuem características desejáveis a um simulador que será utilizado pelo ambiente, como possibilidade de configuração das matrizes que serão experimentadas através de arquivos externos. Entretanto, nenhum deles permite a inclusão de arquiteturas novas sem a necessidade de recompilação do simulador. Além disso, um tem poucos parâmetros para o modelo, enquanto o outro possui parâmetros em demasia.

Tendo em vista principalmente essas limitações, sugere-se a implementação de um simulador próprio para o ambiente, que combine os principais pontos positivos dos dois simuladores estudados. Caminhando nessa direção, esta dissertação apresenta, também, um protótipo de simulador de discos magnéticos que pode vir a ser utilizado como ponto de partida para o desenvolvimento do simulador de RAID para o ambiente.

Tanto o protótipo de simulador apresentado como os dois simuladores estudados utilizam equações disponíveis na literatura (Equação 2.1 e Equação 2.4) para estimar o tempo de busca para uma determinada distância em cilindros dentro de um disco magnético. As equações disponíveis não apresentam um comportamento apropriado nos disco atuais. Hoje, a forma mais precisa de se estimar esse tempo é, tendo disponível uma tabela com tempos de busca para uma certa quantidade de distâncias, interpolar as que forem necessárias e não estiverem presentes na tabela.

Para se utilizar a Equação 2.1 é necessário o tempo médio de busca e a bibliografia apresenta quatro formas de estimar esse tempo. Apesar de Ruemmler e Wilkes (Ruemmler et al., 1994) afirmarem que a forma de cálculo que dá melhor aproximação do tempo médio de busca é a média dos tempos de busca conhecidos ponderada pelo número de buscas de cada tamanho, a seção 8.2 mostra, experimentalmente, que a melhor aproximação para uso com a Equação 2.1 é estimá-lo como o tempo de busca por um terço do número de cilindros.

Diante do exposto nesta seção, pode-se citar como contribuições deste trabalho:

- ## Proposta de um ambiente para a simulação e aprendizado de RAID que pode ser adaptada a outras áreas do conhecimento;
- ## Avaliação dos simuladores para RAID disponíveis na bibliografia;

- ≠# Criação de um protótipo de simulador de discos magnéticos;
- ≠# Avaliação da forma de cálculo do tempo médio de busca disponíveis na literatura e seu impacto da determinação da curva de tempo de busca obtida através da Equação 2.1;
- ≠# Início de uma nova linha de pesquisa dentro do Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP.

9.3 Sugestões de trabalhos futuros

Durante o desenvolvimento deste trabalho surgiram várias frentes que podem ser exploradas no futuro, como a implementação efetiva do Ambiente proposto e avaliação das benesses introduzidas por ele no aprendizado de RAID. Sugere-se ainda, que a estrutura proposta para o Ambiente seja utilizada para o ensino de outras áreas de conhecimento que se beneficiem da simulação como ferramenta de auxílio ao aprendizado. É importante efetuar, também, estudos sobre a interface ideal para o Ambiente proposto à luz de aspectos de ergonomia, interação usuário-computador e cognição humana, tal como os levantados por Nielsen e Norman (Norman, 1986; Norman, 1990; Nielsen, 1993; Norman, 1999) entre outros.

No tocante ao protótipo de simulador de discos magnéticos apresentado no Capítulo 8, sugere-se que o mesmo seja expandido e estendido, com o propósito de aumentar a precisão dos resultados fornecidos por ele através da suplantação das limitações expostas na seção 8.3. Ainda sobre o protótipo, é interessante incluir nele a capacidade de simular não apenas discos magnéticos, mas outras formas de armazenamento, como discos ópticos ou discos de estado sólido (SSD), que possuem características físicas e de tempo de busca bem diferentes dos discos magnéticos.

Considerando o resultado da análise das características dos simuladores feita na seção 7.3 e as características desejáveis a um simulador de RAID para o Ambiente proposto, sugere-se a efetiva implementação de um simulador para ser utilizado como padrão do Ambiente.

Em decorrência das diferenças encontradas entre as estimativas de tempo de busca obtidas pela Equação 2.1 e os valores obtidos através da interpolação linear em uma tabela de tempos conhecidos, sugere-se efetuar estudos adicionais sobre a qualidade dos tempos obtidos com a Equação 2.1 para os discos atuais e propor ajustes, ou a substituição da mesma por outra que represente melhor as características.

Por fim, o ASiA (Ambiente de Simulação Automático), desenvolvido no Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP, pode ser expandido para simular discos magnéticos através da inclusão de um módulo próprio, desenvolvido com base no protótipo de simulador de discos magnéticos apresentado nesta dissertação.

Bibliografia

- ACN, 1998 ADVANCED COMPUTER & NETWORK. **RAID Technology**. [Online]. 1998. [28/03/99]. Disponível na Internet: <<http://www.acnc.com/raid.html>>
- Alvarez et al., 1998 ALVAREZ, Guillermo A.; BURKHARD, Wlateral A.; STOCKMEYER, Larry J.; CRISTIAN, Flaviu. Declustered Disk Array Architectures with Optimal and Near-optimal Parallelism. *In: ANNUAL ACM/IEEE, 25.; INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE. Proceedings*. Barcelona, 1998.
- ASP, 1998 ADVANCED SYSTEM PRODUCTS. **Enhanced IDE and SCSI**. [Online]. 1998. [05/02/99]. Disponível na Internet: <http://www.advansys.com/white_paper/wpide.htm>
- C3D, 1999 C3D DATA STORAGE TECHNOLOGY. **Preparing for the First Generation of FMC & FMD Products**. [Online]. 1999. [11/10/99]. Disponível na Internet: <<http://www.c-3d.net/product.htm>>
- Cao et al., 1992 CAO, Pei; LIM, Swee B.; VENKATARAMAN, Shivakumar; WILKES, John. **The TickerTAIP Parallel RAID Architecture**. Hewlett-Packard Labs. 1992. (Technical Report HPL-2-151)
- Chen et al., 1990 CHEN, Peter; PATTERSON, David. Maximizing Performance in a Striped Disk Array. *In: INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE. Proceedings*. 1990. p.322-331.
- Chen et al., 1994 CHEN, Peter M., LEE, Edward K., PATTERSON, David A., GIBSON, Garth A., KATZ, Randy H. RAID: High-Performance, Reliable Secondary Storage. **ACM Computing Surveys**, v.26, n.2, p.145-181. 1994.
- Chen et al., 1995 CHEN, Peter M.; LEE, Edward K. Striping in a RAID Level 5 Disk Array. *In: ACM SIGMETRICS; INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS. Proceedings*. Ottawa, Canadá. p.136-145. 1995.
- Chen et al., 1996 CHEN, Shenze; TOWSLEY, Don. A Performance Evaluation of RAID Architecture. **IEEE Transactions on Computers**, v.45, n.10, p.1116-1130. 1996.
- Chervenak, 1990 CHERVENAK, Ann L. **Performance Measurements of the First RAID Prototype**. Berkeley, Califórnia: University of Berkeley, 1990. (Technical Report UCB/CSD 90/574)
- Courtright II et al., 1996 COURTRIGHT II, William; GIBSON, Garth; HOLLAND, Mark; ZELENKA, Jim. **RAIDframe: A rapid prototyping tool for RAID systems**. Parallel Data Laboratory, Carnegie Mellon University. 1996. (Manual do usuário)
- Foster, 1995 FOSTER, Ian T. **Designing and Building Parallel Programs**. Estados Unidos: Addison-Wesley Publishing Company Inc., 1995.

- Freire, 1998 FREIRE, Mirla E. P. **O Sistema Tutor de um Ambiente Inteligente para Treinamento e Ensino**. São Carlos, 1998. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional), Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- Ganger, 1995 GANGER, Gregory R. **System-Oriented Evaluation of I/O Subsystem Performance**. Ann Arbor, Michigan, Estados Unidos, 1995. Tese (Doctor of Philosophy – Computer Science and Engineering), Department of EECS, University of Michigan.
- Ganger et al., 1999 GANGER, Gregory R.; WORTHINGTON, Bruce L.; PATT, Yale N. **The DiskSim Simulation Environment Version 2.0 Reference Manual**. Ann Arbor, Michigan: University of Michigan, December 1999.
- Garg et al., 1998 GARG, S et al. Achieving Large Scale Parallelism Through Operating System Resource Management on the Intel TFLOPS Supercomputer. **Intel Technology Journal**, Santa Clara, v.Q1. 1998.
- Gibson et al., 1995 GIBSON, Garth, COURTRIGHT II, Willian, HOLLAND, Mark, ZELENKA, Jim. **RAIDframe: rapid prototyping for disk arrays**. Pittsburgh: Carnegie Mellon University, 1995. (Technical Report CMU-CS-95-200).
- Gross, 1997 GROSS, Frederik. **The Harddisk Guide**. [Online]. [08/02/99]. Disponível na Internet <<http://sysdoc.pair.com/hdd.html>>
- Harrow, 1999 HARROW, Jeffrey. **The Rapidly Changing Face of Computing**. Compaq Computer Corporation. 11 de outubro de 1999.
- Holland, 1994 HOLLAND, Mark. **On-line data reconstruction in redundant disk arrays**. Pittsburgh: Carnegie Mellon University, 1994. (Technical Report CMU-CS-94-164).
- Hoover et al., 1990 HOOVER, Stewart, PERRY, Ronald. **Simulation: a problem-solving approach**. Estados Unidos: Addison-Wesley, 1990.
- IDG, 2000 IDG Now. **Intel impulsiona nova interface para discos rígidos**. [Online]. [16/02/00]. Disponível na Internet: <<http://www.uol.com.br/idgnow/corp/corp2000-02-16a.shl>>
- ISI, 1999 INTELLIGENT SOLUTIONS INC. **A RAID Taxonomy**. [Online]. [14/04/99]. Disponível na Internet: <<http://www.intel-sol.com/solutions/sc/wp RAID7.html>>
- Kozierok, 1998 KOZIEROK, Charles. **The PC Guide**. [Online]. 1998. [23/02/99]. Disponível na Internet: <<http://www.pcguide.com/ref/hdd>>
- Lee et al., 1991a LEE, Edward K.; KATZ, Randy H. Performance Consequences of Parity Placement in Disk Arrays. *In*: INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 4. **Proceedings**. 1991. p.190-199.
- Lee et al., 1991b LEE, Edward K.; KATZ, Randy H. **An Analytic Performance Model of Disk Arrays and its Application**. Berkeley, Califórnia: University of Berkeley, 1991. (Technical Report UCB/CSD 91/660)

- Lee et al., 1996 LEE, Edward K.; THEKKATH, Chandramohan A. Petal: distributed virtual disks. *In: INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS*, 7. **Proceedings**. 1996. p.84-92.
- Lee, 2000 LEE, Edward K. **Disk Average Seek Time**. 2000. [05/04/00] (Mensagem eletrônica trocada com Daniel Corrêa Lobato)
- Lucent, 1999 LUCENT Technologies. **Lucent Technologies and Imation Corp. sign joint agreement to develop holographic storage technology**. [Online]. 1999. [11/10/99]. Disponível na Internet: <<http://www.lucent.com/press/0899/990811.nsb.html>>
- Marvin, 1997 MARVIN, Chris; SIKERNITSKY, Frank; TRINITY College. **Philosophers: Socrates**. [Online]. 1997. [05/02/99]. Disponível na Internet: <<http://www.trincoll.edu/~phil/philo/phils/socrates.html>>
- MacDougall, 1987 MACDOUGALL, Myron H. **Simulating computer systems**. Cambridge, MA, USA: MIT Press, 1987.
- Mattson et al., 1998 MATTSON, T., HENRY, G. An Overview of the Intel TFLOPS Supercomputer. **Intel Technology Journal**, Santa Clara, v.Q1. 1998.
- McLean, 1991 McLEAN, Pete. **An Introduction to RAID**. Digital Equipment Corporation, 1991
- MCP, 1998 MACRO COMPUTER PRODUCTS. **MacroRaid 7 History**. [Online]. 1998. [28/03/99]. Disponível na Internet: <http://www.macrocomp.com/Mraid_History.htm>
- Microsoft, 1998 MICROSOFT CORPORATION. **Microsoft TerraServer**. Redmont, Washington. 1998. (White Paper).
- Monteiro, 1995 MONTEIRO, Mário A. **Introdução à Organização de Computadores**. Rio de Janeiro: Livros Técnicos e Científicos, 1995.
- Nielsen, 1993 NIELSEN, Jakob. **Usability Engineering**. Mountain View, CA: Academic Press, 1993.
- Norman, 1986 NORMAN, Donald A. Cognitive Engineering. *In: NORMAN, D. A.; DRAPER, S. W. (eds). User Centered Design: New Perspectives on Human-Computer Interaction*. New Jersey: Erlbaum Associates, 1986.
- Norman, 1990 NORMAN, Donald A. **The Design of Everyday Things**. Estados Unidos: Bantam Doubleday Dell, 1990.
- Norman, 1999 NORMAN, Donald A. Affordance, Conventions, and Design. **ACM Interactions**. may/june. p.38-42. 1999.
- Oliveira, 1998 OLIVEIRA, Laís L. **Um Módulo de Análise de Saídas para o ASiA**. São Carlos, 1998. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional), Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- Orlandi, 1995 ORLANDI, Regina C. G. S. **Ferramenta para Análise de Desempenho de Sistemas Computacionais Distribuídos**. São Carlos, 1995. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional), Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.

- Patterson et al., 1988 PATTERSON, David A.; GIBSON, Garth A.; KATZ, Randy H. A Case for Redundant Array of Inexpensive Disks (RAID). *In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA. Proceedings*. p.109-116. 1988.
- Pinker, 1998 PINKER, Steven. **Como a Mente Funciona**. São Paulo: Schwarcz, 1998.
- Pollatschek, 1995 POLLATSCHEK, Moshe A. **Programming Discrete Simulations: Tools for Modeling the Real World**. Estados Unidos: R&D Books, 1995.
- Plank, 1997 PLANK, James S. A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. **Software – Practice and Experience**, v.27, p.995-1012, 1997
- Quantum, 1996 QUANTUM CORPORATION. **HDD Operational vs. Theoretical MTBF**. [Online]. [03/11/98]. Disponível na Internet: <<http://www.quantum.com/>>. 1996
- Quantum, 1999 QUANTUM CORPORATION. **Storage Basics**. [Online]. [09/04/99]. Disponível na Internet: <http://www.quantum.com/src/storage_basics>
- Randall, 1998a RANDALL, Neil. Hard Disk Troubles. **PC Magazine**. v.17, n.22, 1998.
- Randall, 1998b RANDALL, Neil. SCSI Just Keeps on Rolling. **PC Magazine**. v.17, n.22, 1998.
- Ruemmler et al., 1994 RUEMMLER, Chris; WILKES, John. An Introduction to Disk Drive Modeling. **IEEE Computer**, v.27, n.3, p.17-29. 1994.
- Santana et al., 1996 SANTANA, M. J.; SANTANA, R. H. C.; SPOLON, R.; SPOLON, R. Automatic Generation of Discret System Simulation Programs. *In: SUMMER COMPUTER SIMULATION CONFERENCE. Proceedings*. Portland, 1996. p.133-138.
- Sato et al., 1996 SATO, Liria; MIDORIKAWA, Edson; SENGHER, Hermes. Introdução a Programação Paralela e Distribuída. *In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA*, 15. **Anais**. Recife: Sociedade Brasileira de Computação, 1996.
- Scherer, 1999 SCHERER, Joachim. What's in a Disk? **Layers Magazine on-line**. [Online]. [22/02/99]. Disponível na Internet: <<http://www.layers.com/bpslay/layersms/mag4.htm>>
- Schindler et al., 1999 SCHINDLER, Jiri; GANGER, Gregory. Automated Disk Drive Characterization. **The PDL Packet: The Newsletter on Parallel Data Systems**. p.14. Parallel Data Laboratory, Carnegie Mellon University. Fall, 1999.
- Seagate, 1997 SEAGATE TECHNOLOGY, INC. **ST-34371W/WC Ultra-SCSI Wide Barracuda 4LP Data Sheet**. [Online]. 1997. [30/09/98]. Disponível na Internet: <<http://www.seagate.com/>>
- Shaffer, 1998 SHAFFER, Steve; KRUMM, John; BRUMITT, Barry; MEYERS, Brian; CZERWINSKI, Mary; ROBBINS, Daniel. The New EasyLiving Project at Microsoft Research. *In: DARPA/NIST SMART SPACES WORKSHOP. Proceedings*. Maryland. 1998

- Soares, 1990 SOARES, Luiz F. G. **Modelagem e Simulação Discreta de Sistemas**. *In: ESCOLA DE COMPUTAÇÃO*, 7. São Paulo: Universidade de São Paulo, Instituto de Matemática e Estatística, 1990.
- Spolon, 1994 SPOLON, Renata. **Um Editor Gráfico para um Ambiente de Simulação Automática**. São Carlos, 1994. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional), Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.
- Stam, 1996 STAM, Nick. **SCSI vs. EIDE: The Real Story**. [Online]. 1996. [04/01/99]. Disponível na Internet: <<http://www.zdnet.com/pcmag/pclabs/report/r960702a.htm>>
- Stodolsky et al., 1993 STODOLSKY, Daniel, HOLLAND, Mark, COURTRIGHT II, William, GIBSON, Garth. **A Redundant Disk Array Architecture for Efficient Small Writes**. Pittsburgh: Carnegie Mellon University, 1993. (Technical Report CMU-CS-94-170).
- Thekkath et al., 1992 THEKKATH, Chandramoham A.; WILKES, John. **Techniques for file system simulation**. Hewlett-Packard Labs. 1992. (Technical Report HPL-92-131)
- Trick, 1996 TRICK, Michael A. **Introduction to Simulation**. [Online]. [15/04/99]. Disponível na Internet: <<http://mat.gsia.cmu.edu/simul/>>
- Turk, 1998 TURK, Matthew. Moving from GUIs to PUIs. *In: PERCEPTUAL USER INTERFACES WORKSHOP*, 98. **Proceedings**. San Francisco, 1998.
- Ulson, 1999 ULSON, Roberta S. **Simulação Distribuída em Plataformas de Portabilidade: Viabilidade de Uso e Comportamento do Protocolo CMB**. São Carlos, 1999. Tese (Doutorado em Física Computacional), Instituto de Física de São Carlos, Universidade de São Paulo.
- Wehman et al., 1996 WEHMAN, John; HERWEIJER, Peter. **The IDE/Fast-ATA/ATA-2 Frequently Asked Questions**. [Online]. [23/02/99]. Disponível na Internet: <<http://members.aa.net/~obata/atafaq.htm>>
- Weikum, 1992 WEIKUM, Gerhard. Tuning of Striping Unit in Disk-Array-Based File Systems. *In: INTERNATIONAL WORKSHOP ON RESEARCH ISSUES ON DATA ENGINEERING, TRANSACTION AND QUERY PROCESSING*, 2. **Proceedings...**, 1992. p.80-87.
- Worthington et al., 1995 WORTHINGTON, Bruce L.; GANGER, Gregory R.; PATT, Yale N.; WILKES, John. On-Line Extraction of SCSI Disk Drive Parameters. *In: ACM SIGMETRICS CONFERENCE. Proceedings*. 1995
- Worthington et al., 1996 WORTHINGTON, Bruce L.; GANGER, Gregory R.; PATT, Yale N.; WILKES, John. **On-Line Extraction of SCSI Disk Drive Parameters**. Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, The University of Michigan. 1996 (Technical Report CSE-TR-323-96)
- Yourdon, 1990 YOURDON, Edward. **Análise Estruturada Moderna**. Rio de Janeiro: Campus, 1990.

Anexo A

Erros na utilização de interfaces

Os erros na utilização da interface são classificados em dois grupos: os *slips* ou deslizes, e os *mistakes* ou erros de uso (Norman, 1990).

Os deslizes são resultados de comportamento automático, quando ações subconscientes para satisfazer os objetivos do usuário são executadas pelo mesmo automaticamente. Os erros de uso resultam de deliberações conscientes. O mesmo princípio que torna o ser humano criativo e capaz de ver as relações entre coisas aparentemente não relacionadas, obtendo conclusões corretas a partir de evidências parciais ou mesmo incorretas, pode levar a erros, tanto deslizes quanto de uso.

Se o objetivo estiver correto mas a forma de execução estiver incorreta, tem-se um deslize. Deslizes são geralmente coisas pequenas: uma ação fora do tempo, um objeto movido incorretamente, ou uma ação desejada mas que não foi executada. Além disso, eles são facilmente detectáveis pela simples observação e monitoramento, uma vez que sabe-se o objetivo e, no momento em que o curso da ação desviar dele, começa-se a suspeitar que algo está errado.

Se a escolha do objetivo estiver errada, tem-se um erro de uso. Os erros de uso geralmente são maiores e eles são difíceis e podem ser impossíveis de detectar, uma vez que as ações que estão sendo executadas são apropriadas para o objetivo escolhido, mesmo que este seja incorreto.

A maior parte dos erros são deslizes. Pressionar o botão errado, que estava ao lado do botão correto, arrastar um objeto para a posição incorreta. Os deslizes tendem a se tornar mais frequentes com o domínio da interface e do sistema que está sendo utilizado. Normalmente os usuários não cometem deslizes em sistemas que ainda estão aprendendo. Em parte, os deslizes ocorrem por falta de atenção. Os deslizes podem ser de seis tipos:

##Deslizes de captura: quando duas seqüências de atividades têm um início em comum, sendo que uma delas ainda não é familiar e a outra é bem conhecida. Geralmente, a seqüência conhecida captura a desconhecida;

##Deslizes de descrição: quando ações diferentes possuem descrições parecidas, existe uma grande chance de executar uma quando a outra deveria ser executada, principalmente se uma delas for executada com mais frequência que as demais e, por isso, não se está prestando muita atenção durante a execução;

##*Deslizes dirigidos pelos dados*: muitas das ações podem ser automáticas e são ativadas por dados – ativadas pela chegada de algum dado sensorial. Algumas vezes, a chegada de um dado sensorial comum pode disparar uma ação automática que não deveria ser executada naquele momento;

##*Deslizes de ativação associativa*: se dados externos podem disparar certas ações, assim também ocorre com dados internos – pensamentos e associações. O usuário pensa em alguma coisa que não deveria ser feita e, sem querer, faz;

##*Deslizes de perda de ativação*: um dos deslizes mais comuns é esquecer de fazer algo. Mais interessante é esquecer parte da ação, lembrando do resto. Um termo menos técnico mas mais comum seria “esquecer” ou “dar branco”;

##*Deslizes de modo*: ocorrem quando o sistema tem diferentes modos de operação numa mesma interface e uma determinada ação em um modo tem significado ou resultado diferente em outro modo. Geralmente ocorrem quando existem mais opções que controles disponíveis, forçando o usuário a saber que determinado elemento age de uma forma em um modo e de outra em outro.

Apesar de serem relativamente fáceis de detectar, uma vez que existe uma diferença clara entre o objetivo e o resultado, a detecção de deslizes só é possível se houver *feedback*. Se o resultado da ação não estiver visível torna-se impossível a detecção.

Os deslizes podem ser evitados ou então detectados e evitados. Por exemplos, os deslizes de modo podem ser reduzidos com a redução de modos ou pelo menos deixando os mesmos bem visíveis.

É comum prevenir deslizes pedindo confirmação antes da execução de um comando, especialmente quando esse comando irá destruir algum dado importante. O problema dessa solução é que o pedido de confirmação não está bem colocado: ele vem logo após o usuário iniciar a ação e, portanto, plenamente certo de seu desejo.

Ao remover um arquivo, por exemplo, o sistema pede a confirmação para a remoção de determinado arquivo, mas o usuário confirma a remoção, e não o arquivo. Por isso, pedir confirmação nem sempre evita os deslizes. O ideal seria desabilitar ações irreversíveis, tal como foi feito em sistemas operacionais como o OS/2 e o Windows 95: o arquivo não é removido, e sim transferido para um local temporário para, só depois, ser efetivamente removido. Dessa forma, o usuário tem tempo de reconsiderar e desfazer a ação.

Anexo B

Códigos fonte e configuração do protótipo de simulador de discos magnéticos

Nas seções seguintes estão os códigos fontes dos programas de simulação utilizados.

B.1 Código do programação de simulação gerado pelo ASiA

```
/* ----- */
#include <dir.h>
#include <stdio.h>
#include <conio.h>
#include "smpl.h"
#include "rand.h"

main()
{
    /* definições */
    float Te = 200000;
    int Event = 1, Customer = 1, Aleatorio;
    real Ta1 = 10, Ts1 = 10;
    int Server1;
    FILE *p, *saida;
    saida = fopen("disco.out", "w");
    if ((p = sendto(saida)) == NULL)
        printf("Erro na saida\n");

    /* prepara o sistema de simulação e dá nome ao modelo */
    smpl(0, "Simulação de disco");

    /* cria e dá nome as facilidades */
    Server1 = facility("Disco", 1);

    /* escalona a chegada do primeiro cliente */
    schedule(1, 0, Customer);

    while ( (time() < Te) )
    {
        cause(&Event, &Customer);
        switch(Event)
        {
            case 1:
                schedule(2, 0.0, Customer);
                schedule(1, uniform(Ta1, 10), Customer);
                break;
            case 2:
                if (request(Server1, Customer, 0) == 0)
                    schedule(3, expntl(Ts1), Customer);
                break;
            case 3:
                release(Server1, Customer);
                break;
        }
    }

    /* gera o relatório da simulação */
    report();
    fclose(saida);
}
/* ----- */
```

B.2 Código do protótipo de simulador de discos magnéticos

```
/////////////////////////////////////////////////////////////////
//  Prototipo de simulador de discos magneticos v0.1al fa
/////////////////////////////////////////////////////////////////
//
//  Autor: Daniel Correa Lobato a partir de esqueleto gerado pelo ASiA
//  Data: abril, 2000
//
#include <dir.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <iomanip.h>
#include <math.h>
#include <string.h>
#include "sss.h"

#define MAX_ZONES 50
#define MAX_SEEK 200

#define ARRIVL 1
#define STARTA 2
#define ENDACT 3

char* LeLinha (FILE* InStream, FILE* OutStream)
{
    char ENTRADA[100];
    char COMENT[1] = "#";
    do {
        fgets(ENTRADA, 100, InStream);
        strcat(ENTRADA, "\0");
        fprintf(OutStream, "%s", ENTRADA);
    } while ((strncmpi(ENTRADA, COMENT, 1) == 0) && !feof(InStream));
    if (feof(InStream)) {
        printf("EOF do arquivo de configuracao antes do tempo!");
        exit(1);
    }
    return(ENTRADA);
}

/////////////////////////////////////////////////////////////////
//
//  INICIO DA ROTINA PRINCIPAL
//
main(int argc, char* argv[]) {
    // Variaveis do simulador
    long int event = 0;
    int busy = 0;
    // Tempo de execucao da simulacao e numero maximo de requisicoes
    double Te, NumReq;
    // Arquivos que serao lidos/escritos e de tabela de tempos de busca
    FILE *arq_entrada, *saida, *debug, *tabbusca;
    // Setor que vai ser lido e cilindro anterior
    unsigned long int setor_atual, cilindro_ant;
    // Cilindro atual e cabeca de leitura anterior
    unsigned long int cilindro_atual, cabeca_ant;
    // Cabeca atual e deslocamento
    unsigned long int cabeca_atual, x;
    // Tamanho da requisicao, zona, contador de I/Os
    unsigned long int s, zona, contador;
    // Semente do gerador de numeros aleatorios
    int s1;
    // Tipos de distribuicao
    int tipodistribcliente, tipodistribsetor, tipodistribtr;
    // Parametros das distribuicoes
    double parC1, parC2, parS1, parS2, parT1, parT2;
    // Tempo do proximo evento
    double proximo_tempo;
```

```

////////////////////////////////////
// VARIÁVEIS COM CARACTERÍSTICAS DO DISCO
// Tempo de revolução e Tempo de busca
double tr, tb;
// Constantes da equação de Lee
double a, b, c;
// Tempo médio, de full stroke e de troca de trilha do disco
double avseek, fullstroke, trocatilha;
// Total de blocos do disco e de blocos por superfície
unsigned long int totalbloco;
unsigned long int totalblocosuperficie;
// Atraso imposto pelo disco para escrita (não utilizado na simulação)
double write_delay;
// Tempo de busca por 1..10 trilhas
double seek[11];
// Tempo de troca de cabeça e de transferência de um setor
double trocacabeca, sectortransfer;
// Número de RPMs, de superfícies de dados, cilindros e número de zonas
int RPM, datasurface, cilindros, numband;
// Como calcula o tempo de busca?
int comocalcula;
// Estrutura utilizada para armazenar as zonas do disco
struct s_Zona {
    unsigned long int inicio, fim, sectrilha, setores, acumuladoV;
    unsigned long int acumuladoH;
} zonas[MAX_ZONES];
// Estrutura utilizada para armazenar a tabela de tempos de busca
struct s_Tempo {
    unsigned long int distancia;
    double tempo;
} tempos[MAX_SEEK];
unsigned long int tabela;

////////////////////////////////////
// VARIÁVEIS TEMPORÁRIAS
// Buffer de entrada
char TMP[100];
// Variáveis auxiliares nos cálculos
unsigned long int offset_zona, sec_cil, resto;
double passo;
// Contador eventual
int i;

////////////////////////////////////
//
// IDENTIFICAÇÃO DOS ARQUIVOS DE ENTRADA, SAÍDA E DEBUG
//
// Obtem o arquivo de entrada de dados
printf("////////////////////////////////////\n");
printf("// Protótipo de simulador de discos magnéticos v0.1a1fa\n");
printf("////////////////////////////////////\n");
printf("//\n");
printf("// Autor: Daniel Correa Lobato\n");
printf("// Data: abril, 2000\n");
printf("//\n");
// Checa se todos os parâmetros estão presentes
// O simulador vai ser utilizado apenas para calcular as constantes da
// equação de Lee.
if (argc == 5) {
    printf("// Módulo de cálculo das constantes da equação de Lee\n");
    printf("//\n");
    printf("// miSeek: %9.6f\n", atof(argv[1]));
    printf("// avSeek: %9.6f\n", atof(argv[2]));
    printf("// maSeek: %9.6f\n", atof(argv[3]));
    printf("// numCyl: %9.0f\n", atof(argv[4]));
    printf("//\n");
    // A, B e C da equação de Lee
    trocatilha = atof(argv[1]);
    avseek = atof(argv[2]);
    fullstroke = atof(argv[3]);
    cilindros = atof(argv[4]);
}

```

```

a = (-10*trocatriilha + 15*avseek - 5*fullstroke) / (3*sqrt(cilindros));
b = (7*trocatriilha - 15*avseek + 8*fullstroke) / (3*cilindros);
printf("// Constantes da equacao de Lee\n");
printf("//\n");
printf("// a: %9.6f\tb: %9.6f\tc: %9.6f\n", a, b, trocatriilha);
printf("//\n");
printf("////////////////////////////////////////\n");
exit(0);
}
if (argc < 6) {
printf("// Forma de uso: \n");
printf("// C: \\>Simulador <input> <seed> <output> <seek?> <debug> ");
printf("[seekfile]\n");
printf("//\n");
printf("// <input>... Arquivo de configuracao\n");
printf("// <seed>... Semente do gerador de numeros aleatorios\n");
printf("// <output>... Arquivo de saida dos resultados\n");
printf("// <seek?>... Tempos de busca calculados por (1) Lee ou\n");
printf("// (2) tabela externa\n");
printf("// <debug>... Arquivo para debug da simulacao\n");
printf("// [seekfile]. Se <seek?>=2, arquivo com a tabela de tempo ");
printf("de busca\n");
printf("//\n");
printf("// C: \\>Simulador <miSeek> <avSeek> <maSeek> <numCyl>\n");
printf("//\n");
printf("// Calcula as constantes da equacao de Lee\n");
printf("//\n");
printf("////////////////////////////////////////\n");
exit(1);
}
printf("// arquivo com configuracao do disco ");

// Abre o arquivo de configuracao somente para leitura
if ((arq_entrada = fopen(argv[1], "r")) == NULL) {
printf("*** ERRO: impossivel abrir arquivo de configuracao desejado\n");
exit(1);
}
printf("%s\n", argv[1]);

// Qual a semente?
printf("// semente do gerador ");
s1 = atoi(argv[2]);
printf("%d\n", s1);

// Descobrir o nome do arquivo de saida e abri-lo
printf("// arquivo de saida da simulacao ");
if ((saida = fopen(argv[3], "w")) == NULL) {
printf("*** ERRO: impossivel abrir arquivo de saida\n");
fclose(arq_entrada);
exit(1);
}
printf("%s\n", argv[3]);

// Como vai ser o calculo do tempo de busca???
printf("// calculo do tempo de busca ");
comocalcula = atoi(argv[4]);
if (comocalcula == 1) printf("Lee\n");
else if (comocalcula == 2) printf("Tabela\n");
else {
printf("ERRO!!!");
fclose(arq_entrada);
fclose(saida);
}

// Descobrir o nome do arquivo de debug e abri-lo
printf("// arquivo de debug da simulacao ");
if ((debug = fopen(argv[5], "w")) == NULL) {
printf("*** ERRO: impossivel abrir arquivo de debug\n");
fclose(arq_entrada);
fclose(saida);
exit(1);
}

```

```

    }
    printf("%s\n", argv[5]);

// Vai buscar em tabela??? Qual o nome?
if (comocalcula == 2) {
    if (argc == 7) {
        printf("//          arquivo com tempos de busca> %s\n", argv[6]);
    }
    else {
        printf("*** ERRO: arquivo com tempos de busca nao fornecido\n");
        fclose(arq_entrada);
        fclose(saida);
        fclose(debug);
        exit(1);
    }
}

////////////////////////////////////
//
// LEITURA DOS PARAMETROS DA SIMULACAO
//
printf("// Lendo configuracao ");
fprintf(debug, "*****\n");
fprintf(debug, "*** Lendo configuracoes a partir do arquivo %s\n", argv[1]);
fprintf(debug, "*****\n");
fprintf(debug, "\n");

// Leitura dos parametros da simulacao
strcpy(TMP, LeLinha(arq_entrada, saida)); Te = atof(TMP);
fprintf(debug, "Tempo de simulacao (ms)> %f\n", Te);
strcpy(TMP, LeLinha(arq_entrada, saida)); NumReq = atof(TMP);
fprintf(debug, "Numero maximo de requisicoes> %f\n", NumReq);
printf("\b-");

// Leitura dos parametros da distribuicao de chegada de clientes
strcpy(TMP, LeLinha(arq_entrada, saida)); tipodistribcliente = atoi(TMP);
if ((tipodistribcliente < 1) || (tipodistribcliente > 7)) {
    fprintf(debug, "Tipo de distribuicao de chegada de cliente ");
    fprintf(debug, "(%d) invalido!", tipodistribcliente);
    fclose(arq_entrada); fclose(debug); fclose(saida);
    exit(1);
}
strcpy(TMP, LeLinha(arq_entrada, saida)); parC1 = atof(TMP);
strcpy(TMP, LeLinha(arq_entrada, saida)); parC2 = atof(TMP);
fprintf(debug, "Distribuicao de chegada e parametros> ");
fprintf(debug, "%d - %8.3f - %8.3f\n", tipodistribcliente, parC1, parC2);
printf("\b\\");

// Leitura dos parametros da distribuicao do setor a ser lido
strcpy(TMP, LeLinha(arq_entrada, saida)); tipodistribsetor = atoi(TMP);
if ((tipodistribsetor < 1) || (tipodistribsetor > 7)) {
    fprintf(debug, "Tipo de distribuicao do setor a ser lido ");
    fprintf(debug, "(%d) invalido!", tipodistribsetor);
    fclose(arq_entrada); fclose(debug); fclose(saida);
    exit(1);
}
strcpy(TMP, LeLinha(arq_entrada, saida)); parS1 = atof(TMP);
strcpy(TMP, LeLinha(arq_entrada, saida)); parS2 = atof(TMP);
fprintf(debug, "Distribuicao do setor e parametros> ");
fprintf(debug, "%d - %8.3f - %8.3f\n", tipodistribsetor, parS1, parS2);
printf("\b|");

// Leitura dos parametros da distribuicao do tamanho da requisicao
strcpy(TMP, LeLinha(arq_entrada, saida)); tipodistribtr = atoi(TMP);
if ((tipodistribtr < 1) || (tipodistribtr > 7)) {
    fprintf(debug, "Tipo de distribuicao do tamanho da requisicao ");
    fprintf(debug, "(%d) invalido!", tipodistribtr);
    fclose(arq_entrada); fclose(debug); fclose(saida);
    exit(1);
}
strcpy(TMP, LeLinha(arq_entrada, saida)); parT1 = atof(TMP);

```



```

strcpy(TMP, LeLi nha(arq_entrada, sai da)); parT2 = atof(TMP);
fprintf(debug, "Di stribui cao do tamanho da requisi cao e parametros> ");
fprintf(debug, "%d - %8.3f - %8.3f\n", ti podi stribtr, parT1, parT2);
printf("\b/");

////////////////////////////////////
//
// LEITURA DOS PARAMETROS DO DISCO
//
// Tempo de troca de trilha
strcpy(TMP, LeLi nha(arq_entrada, sai da)); trocatri lha = atof(TMP);
fprintf(debug, "Val or assumi do para trocatri lha> %f\n", trocatri lha);
printf("\b-");

// Tempo medio de busca
strcpy(TMP, LeLi nha(arq_entrada, sai da)); avseek = atof(TMP);
fprintf(debug, "Val or assumi do para avseek> %f\n", avseek);
printf("\b\\");

// Tempo de fullstroke
strcpy(TMP, LeLi nha(arq_entrada, sai da)); fullstroke = atof(TMP);
fprintf(debug, "Val or assumi do para fullstoke> %f\n", fullstroke);
printf("\b|");

// Atraso de escri ta
strcpy(TMP, LeLi nha(arq_entrada, sai da)); wri te_del ay = atof(TMP);
fprintf(debug, "Val or assumi do para wri te_del ay> %f\n", wri te_del ay);
printf("\b/");

// Se for o caso, ler os tempos de busca a partir da tabela
if (comocal cula == 2) {
    if ((tabbusca = fopen(argv[6], "r")) == NULL) {
        printf("*** ERRO: impossivel abrir %s com tempos de busca\n", TMP);
        fclose(arq_entrada); fclose(debug); fclose(sai da);
        exit(1);
    }
    strcpy(TMP, LeLi nha(tabbusca, sai da));
    // Ha, tabel ado, tempo de busca para <tabela> di stancias. Vamos le-los...
    tabela = atoi (TMP);
    if (tabela > MAX_SEEK){
        printf("*** ERRO: Excedido o li mi te de li nhas na tabela de tempos ");
        printf(" de busca. Li mi te atual: %d\n", MAX_SEEK);
        fclose(arq_entrada); fclose(debug); fclose(sai da);
        exit(1);
    }
    for(i=1; i<=tabela; i++) {
        strcpy(TMP, LeLi nha(tabbusca, sai da));
        // Para a di stancia tempos[i]. di stancia...
        tempos[i]. di stancia = atoi (TMP);
        strcpy(TMP, LeLi nha(tabbusca, sai da));
        // ...o tempo de busca eh de tempos[i]. tempo
        tempos[i]. tempo = atof(TMP);
        // E tenho o di to!
        fprintf(debug, " Busca por %5d ci li ndros", tempos[i]. di stancia);
        fprintf(debug, " demora %3.6fms\n", tempos[i]. tempo);
        if (i % 4 == 0) printf("\b-");
        if (i % 4 == 1) printf("\b\\");
        if (i % 4 == 2) printf("\b|");
        if (i % 4 == 3) printf("\b/");
    }
    fclose(tabbusca);
}

// Tempo de busca por 1..10 trilhas
for (i=1; i<=10; i++) {
    strcpy(TMP, LeLi nha(arq_entrada, sai da)); seek[i] = atof(TMP);
    fprintf(debug, "Val or assumi do para seek[%d]> %f\n", i, seek[i]);
    if (i % 4 == 0) printf("\b-");
    if (i % 4 == 1) printf("\b\\");
    if (i % 4 == 2) printf("\b|");
    if (i % 4 == 3) printf("\b/");
}

```

```

    }

// Tempo de troca de cabeca
strcpy(TMP, LeLi nha(arq_entrada, saida)); trocacabeca = atof(TMP);
fprintf(debug, "Val or assumi do para trocacabeca> %f\n", trocacabeca);
printf("\b/");

// RPM
strcpy(TMP, LeLi nha(arq_entrada, saida)); RPM = atoi (TMP);
fprintf(debug, "Val or assumi do para RPM> %d\n", RPM);
printf("\b-");

// Superfi cies de dados
strcpy(TMP, LeLi nha(arq_entrada, saida)); datasurface = atoi (TMP);
fprintf(debug, "Val or assumi do para datasurface> %d\n", datasurface);
printf("\b\\");

// Cil indros
strcpy(TMP, LeLi nha(arq_entrada, saida)); cilindros = atoi (TMP);
fprintf(debug, "Val or assumi do para cilindros> %d\n", cilindros);
printf("\b|");

// Tempo de transferencia de um setor
strcpy(TMP, LeLi nha(arq_entrada, saida)); sectortransfer = atof(TMP);
fprintf(debug, "Val or assumi do para sectortransfer> %f\n", sectortransfer);
printf("\b/");

// Numero de zonas
strcpy(TMP, LeLi nha(arq_entrada, saida)); numband = atoi (TMP);
fprintf(debug, "Val or assumi do para numband> %d\n", numband);
if (numband > MAX_ZONES){
    printf("*** ERRO: Excedi do o li mi te de zonas no di sco. ");
    printf("Li mi te atual: %d\n", MAX_ZONES);
    fclose(arq_entrada); fclose(debug); fclose(sai da);
    exit(1);
}
printf("\b-");

// Agora, popular a estrutura de zonas numband vezes
for (i=1; i<=numband; i++) {
    fprintf(debug, "    Dados da banda %d\n", i);
    strcpy(TMP, LeLi nha(arq_entrada, saida)); zonas[i].i ni ci o = atoi (TMP);
    fprintf(debug, "        i ni ci o. . . . . %d\n", zonas[i].i ni ci o);

    strcpy(TMP, LeLi nha(arq_entrada, saida)); zonas[i].fi m = atoi (TMP);
    fprintf(debug, "        fi m. . . . . %d\n", zonas[i].fi m);

    strcpy(TMP, LeLi nha(arq_entrada, saida)); zonas[i].sectri l ha = atoi (TMP);
    fprintf(debug, "        sectri l ha. . . %d\n", zonas[i].sectri l ha);

    zonas[i].setores = (zonas[i].sectri l ha * \
        (zonas[i].fi m - zonas[i].i ni ci o + 1));
    fprintf(debug, "        setores. . . . . %d\n", zonas[i].setores);
    if (i == 1) {
        zonas[i].acumul adoH = zonas[i].setores;
        zonas[i].acumul adoV = datasurface * zonas[i].acumul adoH;
    }
    else {
        zonas[i].acumul adoH = zonas[i-1].acumul adoH + zonas[i].setores;
        zonas[i].acumul adoV = datasurface * zonas[i].acumul adoH;
    }
    fprintf(debug, "        acumul adosH. . %d\n", zonas[i].acumul adoH);
    fprintf(debug, "        acumul adosV. . %d\n", zonas[i].acumul adoV);
    if (i % 4 == 0) printf("\b-");
    if (i % 4 == 1) printf("\b\\");
    if (i % 4 == 2) printf("\b|");
    if (i % 4 == 3) printf("\b/");
}

// Fecha o arqui vo de entrada
fclose(arq_entrada);

```

```

printf("\n");

////////////////////////////////////////
//
// CALCULO DE VARIAVEIS QUE DEPENDEM DOS PARAMETROS LIDOS
//
// Calcula o numero de blocos no disco e por superficie
printf("// Calculando variaveis dependentes -\n");
totalblocosuperficie = zonas[numband].acumuladoH;
totalbloco = totalblocosuperficie * datasurface;

fprintf(debug, "\nTotal de blocos por superficie...: ");
fprintf(debug, "%d setores\n", totalblocosuperficie);
fprintf(debug, "Total de blocos no disco.....: ");
fprintf(debug, "%d setores\n", totalbloco);

// Se algum dos parametros de distribuicao de setor a ser lido for 1919, trocar
// pelo numero total de blocos
if (parS1 == 1919) {
    parS1 = (double)(totalbloco-1);
    fprintf(debug, "!! Reajustando o parametro parS1 para o ultimo ");
    fprintf(debug, "bloco (%10.0f) !!\n", parS1);
}
if (parS2 == 1919) {
    parS2 = (double)(totalbloco-1);
    fprintf(debug, "!! Reajustando o parametro parS2 para o ultimo ");
    fprintf(debug, "bloco (%10.0f) !!\n", parS2);
}

// Se algum dos parametros de distribuicao do tamanho da requisicao for 1919,
// trocar pelo numero total de blocos
if (parT1 == 1919) {
    parT1 = (double)(totalbloco-1);
    fprintf(debug, "!! Reajustando o parametro parT1 para o ultimo ");
    fprintf(debug, "bloco (%10.0f) !!\n", parT1);
}
if (parT2 == 1919) {
    parT2 = (double)(totalbloco-1);
    fprintf(debug, "!! Reajustando o parametro parT2 para o ultimo ");
    fprintf(debug, "bloco (%10.0f) !!\n", parT2);
}

// Tempo de revolucao
tr = (double)(1*60*1000)/(double)RPM;
fprintf(debug, "Tempo de revolucao: %fms\n", tr);

// A, B e C da equacao de Lee
a = (-10*trocatriilha + 15*avseek - 5*fullstroke) / (3*sqrt(cilindros));
b = (7*trocatriilha - 15*avseek + 8*fullstroke) / (3*cilindros);
c = trocatriilha;
fprintf(debug, "\nConstantes da equacao de Lee\n");
fprintf(debug, "a: %f\tb: %f\tc: %f\n", a, b, c);

if ((a<0) || (b<0) || (c<0)) {
    fprintf(debug, "ATENCAO! Constantes da equacao de Lee negativas!\n");
    fprintf(debug, "~~~~~ De acordo com (Lee, 2000) prefira calcular o ");
    fprintf(debug, "tempo de busca por interpolacao!!\n");
}

fprintf(debug, "\n===== [ Debug da simulacao ]");
fprintf(debug, " ]===== \n");

fprintf(debug, " cont sector size ! Z Cyl SF Sec ! X");
fprintf(debug, " SeekTime ! ServTime\n");

////////////////////////////////////////
//
// INICIALIZACAO DA SIMULACAO
//
// prepara o sistema de simulacao e da nome ao modelo
printf("//\n");

```

```

printf("// Simulacao iniciada!\n");
printf("//\n");

INI QUE(1, 1, 4);
SETSEE(s1);
CREATE(0.0, 0);
INI STA(2, "TempoBusca", 0, 30, 1, (double)(fullstroke/30));
INI STA(3, "TempoServiço", 0, 30, (double)(0.5*tr), \
(double)((double)(tr+fullstroke)/30));
INI STA(4, "TempoResposta", 0, 30, 14, 2);

// Inicializa contadores auxiliares
cilindro_ant = 0;
cabeca_ant = 0;
contador = 0;

fpri ntf(sai da, "\n\n===== [ Sai da da Simulacao ");
fpri ntf(sai da, "]===== \n");

////////////////////////////////////
//
// LACO PRINCIPAL DA SIMULACAO
//
printf("// Executando E/Ss ");
do {
    if (contador % 1000 == 0) printf("\b-");
    if (contador % 1000 == 250) printf("\b\\");
    if (contador % 1000 == 500) printf("\b|");
    if (contador % 1000 == 750) printf("\b/");
    event = NEXTEV();
    if (event) switch (event) {
        //////////////////////////////////////
        //
        // Hora de escalonar novo cliente
        //
        //
        case ARRIVL:
            SETA(1, T());
            switch (tipodistribcliente) {
                case 1: CREATE(BE(parC1, parC2), IDE() + 1); break;
                case 2: CREATE(ER(parC1, parC2), IDE() + 1); break;
                case 3: CREATE(EX(parC1), IDE() + 1); break;
                case 4: CREATE(GA(parC1, parC2), IDE() + 1); break;
                case 5: CREATE(RN(parC1, parC2), IDE() + 1); break;
                case 6: CREATE(NP(parC1), IDE() + 1); break;
                case 7: CREATE(UN(parC1, parC2), IDE() + 1); break;
            }
            if (busy) QUEUE(1, 0);
            else SCHED(0.0, STARTA, IDE());
            break;
        //////////////////////////////////////
        //
        // Atender um cliente
        //
        case STARTA:
            busy = 1;
            proximo_tempo = 0.00;
            contador++;
            fpri ntf(debug, "%06X ", contador);
            // QUAL setor eu vou ler???
            switch (tipodistribsetor) {
                case 1: setor_atual = (unsigned long int)BE(parS1, parS2); break;
                case 2: setor_atual = (unsigned long int)ER(parS1, parS2); break;
                case 3: setor_atual = (unsigned long int)EX(parS1); break;
                case 4: setor_atual = (unsigned long int)GA(parS1, parS2); break;
                case 5: setor_atual = (unsigned long int)RN(parS1, parS2); break;
                case 6: setor_atual = (unsigned long int)NP(parS1); break;
                case 7: setor_atual = (unsigned long int)UN(parS1, parS2); break;
            }
            // Se solicitar um setor alem do limite do disco, solicita o maior
            // possivel...

```

```

    if (setor_atual > (total_blocos - 1))
        setor_atual = total_blocos - 1;
    fprintf(debug, "%09d ", setor_atual);

// e QUANTOS setores a partir dele
switch(tipodistribtr) {
    case 1: s = (unsigned long int)BE(parT1, parT2); break;
    case 2: s = (unsigned long int)ER(parT1, parT2); break;
    case 3: s = (unsigned long int)EX(parT1); break;
    case 4: s = (unsigned long int)GA(parT1, parT2); break;
    case 5: s = (unsigned long int)RN(parT1, parT2); break;
    case 6: s = (unsigned long int)NP(parT1); break;
    case 7: s = (unsigned long int)UN(parT1, parT2); break;
}
// Se estiver lendo mais setores do que ha no disco, limita no maximo
// possivel...
if (setor_atual + s > (total_blocos - 1))
    s = total_blocos - setor_atual - 1;
fprintf(debug, "%09d ! ", s);

////////////////////////////////////
// DESMONTAR O SETOR EM ZONA, CILINDRO, SUPERFICIE, SETOR
//
// Em qual zona esta o setor desejado? (Passo 1)
for (i=1; i<=numband; i++) {
    if (setor_atual < zonas[i].acumuladoV) {
        zona = i; break;
    }
}

// Temos quantos setores por cilindro para essa zona? (Passo 2)
sec_cil = zonas[zona].sectrilha * datasurface;

// Quantos setores ha antes do inicio da zona? (Passo 3)
if (zona == 1)
    offset_zona = setor_atual - 0;
else
    offset_zona = setor_atual - zonas[zona-1].acumuladoV;

// Sabendo a zona, descubra o cilindro... (Passo 4)
cilindro_atual = (offset_zona / sec_cil) + zonas[zona].inicio;
resto = offset_zona % sec_cil;

// ... e o deslocamento em relacao a requisicao anterior
x = abs(cilindro_atual - cilindro_ant);

// Calcular em qual superficie esta o dado (Passo 5)
cabeca_atual = (resto / zonas[zona].sectrilha);
resto = resto % zonas[zona].sectrilha;

// Achou a cabeca... Se mudou, acrescenta ao tempo...
if (cabeca_atual != cabeca_ant)
    proximo_tempo += trocacabeca;

// Anotar no debug o setor desmontado...
fprintf(debug, "%02d %06d %02d %03d ! ", zona, cilindro_atual, \
        cabeca_atual, resto);

// ... e o deslocamento
fprintf(debug, "%06d ", x);
//
// DESMONTAR O SETOR EM ZONA, CILINDRO, SUPERFICIE, SETOR
////////////////////////////////////
////////////////////////////////////
// CALCULAR O TEMPO DE BUSCA PARA O DESLOCAMENTO EM QUESTAO
//
// Se o deslocamento eh 0, nao tem tempo de busca
if (x == 0)
    tb = (double)0.00;

// Eh para usar Lee?

```

```

else if (comocalcula == 1)
// Se for pequena, usa a tabela de pequenas distancias
    if ((x >= 1) && (x <= 10)) tb = seek[x];
// senao, usa Lee
    else tb = a * (double)sqrt(x - 1) + b * (double)(x - 1) + c;

// Eh para usar interpolacao
else if (comocalcula == 2) {
    for (i=1; i<=tabela; i++) {
        if (tempos[i].distancia == x) {
            tb = tempos[i].tempo;
            break;
        }
        if (tempos[i].distancia > x) {
            passo = (tempos[i].tempo - tempos[i-1].tempo)/\
                (double)(tempos[i].distancia - tempos[i-1].distancia);
            tb = (double)(tempos[i-1].tempo + \
                ((double)(x - tempos[i-1].distancia) * passo));
            break;
        }
    }
}
else fprintf(debug, "@@@" );

// Isso aqui eh o SeekTime
fprintf(debug, "%09.5f ! ", tb);
proximo_tempo += tb;
TALLY(2, proximo_tempo);

//
// CALCULAR O TEMPO DE BUSCA PARA O DESLOCAMENTO EM QUESTAO
//
//
// CALCULAR O ATRASO ROTACIONAL
// E o atraso rotacional ???

// Distribuição uniforme entre 0 e o tempo de rotação
proximo_tempo += UN((double)0.00, tr);

// CALCULAR O ATRASO ROTACIONAL
//
//
// TEMPO DE TRANSFERENCIA DOS SETORES DESEJADOS

proximo_tempo += (double)(sectortransfer * (double)s);
fprintf(debug, "%09.5f\n", proximo_tempo);

// TEMPO DE TRANSFERENCIA DOS SETORES DESEJADOS
//
// Escalona o fim de serviço e liberação do disco
SCHED(proximo_tempo, ENDACT, IDE());
TALLY(3, proximo_tempo);

cilindro_ant = cilindro_atual;
cabeca_ant = cabeca_atual;
break;
//
//
// Liberar o disco para o proximo
//
//
case ENDACT:
    TALLY(4, T() - A(1));
    if ((IDE() >= NumReq) || (T() >= Te)) SIMEND(0);
    DISPOS();
    busy = 0;
    if (NQ(1)) {
        REMVFQ(1, 1);
    }

```

```

        SCHED(0.0, STARTA, IDE());
    }
    break;
}
} while (event);

// Gera o relatório da simulação
printf("\n//\n");
printf("// Fim da simulação!\n");
printf("//\n");
printf("// Numero de E/S: %d\n", contador);
printf("// Tempo de simulação: %12.5fms\n", T());
printf("////////////////////////////////////////\n");

fprintf(saida, "Numero de ESs: ");
fprintf(saida, "%d\n", contador);
fprintf(saida, "ES por segundo: ");
fprintf(saida, "%10.5f\n", (double)contador/((double)T()/1000));
fclose(saida);

fprintf(debug, "Numero de ESs: ");
fprintf(debug, "%d\t", contador);
fprintf(debug, "ES por segundo: ");
fprintf(debug, "%10.5f\n", (double)contador/((double)T()/1000));
fclose(debug);

SUMRY(argv[3]);
}

```

B.3 Exemplo de arquivo de saída de simulação

O arquivo de saída da simulação começa com uma cópia do arquivo de configuração utilizado e, em seguida, o número de operações de entrada/saída efetuadas durante a simulação e o número de ES/s. Depois, são apresentadas as estatísticas de tamanho da fila (*Queue No 1*), tempo de busca, tempo de serviço e tempo de resposta. Para cada um dos tempos é apresentado um histograma de frequência.

```

===== [ Sai da da Simulação ] =====
Numero de I/Os: 500789
I/Os por segundo: 50.07890
Queue No 1 - summary statistics for 9999999.93 time duration
  Average= 0.4042, standard deviation= 0.8698
  minimum= 0.0000, maximum = 11.0000
TempoBusca - summary statistics for sample size 500789
  Average= 6.6423, standard deviation= 3.2192
  minimum= 0.9990, maximum = 16.2925
cell upper frequ
# limit ency one * = 1003 cases
1 1.00 38
2 1.51 0
3 2.02 4938 ****
4 2.54 4393 ****
5 3.05 50130 *****
6 3.56 40364 *****
7 4.07 35666 *****
8 4.58 32999 *****
9 5.10 31208 *****
10 5.61 29368 *****
11 6.12 27489 *****
12 6.63 25448 *****
13 7.14 24347 *****
14 7.66 22369 *****
15 8.17 21224 *****
16 8.68 19708 *****
17 9.19 18308 *****
18 9.70 17059 *****

```

19	10.22	14856	*****
20	10.73	13943	*****
21	11.24	12332	*****
22	11.75	11192	*****
23	12.26	9023	*****
24	12.78	8295	*****
25	13.29	7415	*****
26	13.80	6154	*****
27	14.31	4549	****
28	14.82	3504	***
29	15.34	2606	**
30	15.85	1361	*
31	inf.	503	

TempoServiço - summary statistics for sample size 500789
Average= 11.2874, standard deviation= 3.2629
minimum= 5.1657, maximum = 22.8488

cell #	upper limit	frequency	one * = 1126 cases
1	4.17	0	
2	4.96	0	
3	5.75	22	
4	6.54	4616	****
5	7.33	32298	*****
6	8.12	56278	*****
7	8.91	52923	*****
8	9.70	49833	*****
9	10.48	44989	*****
10	11.27	40427	*****
11	12.06	36208	*****
12	12.85	33201	*****
13	13.64	29784	*****
14	14.43	25900	*****
15	15.22	23018	*****
16	16.01	18933	*****
17	16.80	15498	*****
18	17.59	12687	*****
19	18.38	10059	*****
20	19.17	7206	*****
21	19.96	4278	***
22	20.75	1961	*
23	21.54	516	
24	22.33	123	
25	23.12	31	
26	23.91	0	
27	24.70	0	
28	25.49	0	
29	26.28	0	
30	27.07	0	
31	inf.	0	

TempoResposta - summary statistics for sample size 500788
Average= 19.3590, standard deviation= 12.2596
minimum= 5.2717, maximum = 137.6930

cell #	upper limit	frequency	one * = 4241 cases
1	14.00	212002	*****
2	16.00	46095	*****
3	18.00	39519	*****
4	20.00	32065	*****
5	22.00	25037	*****
6	24.00	21093	****
7	26.00	18291	****
8	28.00	15573	***
9	30.00	13682	***
10	32.00	10863	**
11	34.00	10188	**
12	36.00	8459	*
13	38.00	7501	*
14	40.00	6022	*
15	42.00	5041	*
16	44.00	4608	*
17	46.00	3926	

18	48.00	2984
19	50.00	2840
20	52.00	2394
21	54.00	2068
22	56.00	1609
23	58.00	1286
24	60.00	1124
25	62.00	925
26	64.00	902
27	66.00	796
28	68.00	682
29	70.00	466
30	72.00	347
31	inf.	2400

B.4 Exemplo de arquivo de debug da simulação

Devido ao tamanho do arquivo de debug (facilmente superior a 40Mb), é apresentado abaixo apenas um pequeno trecho. O arquivo começa com mensagens informando os valores atribuídos a cada parâmetro e os valores das constantes para a Equação 2.1. Depois, é apresentado uma lista de todas as requisições emitidas contendo o número da requisição em hexadecimal (*cont*), o setor que foi requisitado (*sector*) e quantos setores a partir dele (*size*). O grupo de colunas seguintes informa a zona, cilindro, superfície e setor (*Z*, *Cyl*, *SF* e *Sec*) correspondente ao setor solicitado. As duas próximas colunas indicam o deslocamento em cilindros entre a requisição anterior e a atual (*X*), e o tempo de busca para aquela distância (*SeekTime*). A última coluna mostra o tempo de serviço necessário (*ServTime*) para atender a requisição em questão.

```
===== [ Debug da simulacao ] =====
cont    sector    size ! Z    Cyl SF Sec !      X  SeekTime !  ServTime
000001  000668231  000000001 ! 01 000261 00 071 ! 000261 004.18328 ! 007.76383
000002  011422299  000000002 ! 05 004701 03 187 ! 004440 010.79940 ! 016.02054
000003  002762369  000000019 ! 02 001086 04 001 ! 003615 009.67765 ! 016.08891
000004  011940910  000000015 ! 05 004932 09 014 ! 003846 010.04458 ! 013.88188
000005  004988622  000000004 ! 03 001977 01 062 ! 002955 008.86755 ! 011.16844
000006  013229778  000000010 ! 06 005532 00 018 ! 003555 009.60520 ! 017.01914
000007  003659544  000000001 ! 02 001442 04 056 ! 004090 010.34930 ! 011.86794
000008  006944081  000000004 ! 03 002791 09 001 ! 000677 005.50063 ! 014.84561
000009  009155290  000000003 ! 04 003716 00 170 ! 000925 005.92900 ! 012.25304
00000A  003156251  000000003 ! 02 001242 07 007 ! 002474 008.23308 ! 015.70306
00000B  008273432  000000011 ! 03 003345 07 232 ! 002103 008.21622 ! 012.01805
:      :      :      :      :      :      :      :      :      :
.      .      .      .      .      .      .      .      .      .
```

B.5 Exemplo de arquivo de configuração para o protótipo de simulador de discos

```
#####
# Arquivo de configuracao para o Simulador de Discos v0.1a1fa
# Parametros relativos ao disco Quantum QM39100TD-SW
#####
#
# TODOS OS TEMPOS EM MILISEGUNDOS (ms)
#
# Os parametros TEM QUE SEGUIR a ordem em que estao!!!
#####
#
# P A R A M E T R O S   D A   S I M U L A C A O
```

```

#
# A simulacao executa ate que o tempo de execucao *OU* o numero
# maximo de requisicoes seja atingido, o que ocorrer primeiro
#
# Tempo de execucao, em ms
9999999
# Numero maximo de requisicoes a serem emitidas
9999999999
#####
#
# P A R A M E T R O S   D A   D I S T R I B U I C A O
# D E   C H E G A D A   D E   C L I E N T E S
#
# Tipo de distribuicao de chegada das requisicoes de cliente
# 1. Beta          media = P1/(P2+P1)
# 2. Erlang        P1 estagios e media P1*P2
# 3. Exponencial  media = P1
# 4. Gama          Erlang com P1 inteiro
# 5. Normal        media = P1 e desvio padrao = P2
# 6. Poisson       media = P1
# 7. Uniforme      entre P1 e P2
3
# Parametros que serao usados na funcao acima.
# OBRIGATORIAMENTE DOIS!!! Se um nao for usado, colocar 0
# -----
15
0
#####
#
# P A R A M E T R O S   D A   D I S T R I B U I C A O
# D O   S E T O R   A   S E R   R E Q U I S I T A D O
#
# Tipo de distribuicao de chegada das requisicoes de cliente
# Igual ao da distribuicao de chegada de clientes
7
# Parametros que serao usados na funcao acima.
# OBRIGATORIAMENTE DOIS!!! Se um nao for usado, colocar 0
# -----
# Se quiser usar o numero total de setores, colocar 1919
0
1919
#####
#
# P A R A M E T R O S   D A   D I S T R I B U I C A O
# D O   T A M A N H O   D A   R E Q U I S I C A O
#
# Tipo de distribuicao de chegada das requisicoes de cliente
# Igual ao da distribuicao de chegada de clientes
3
# Parametros que serao usados na funcao acima.
# OBRIGATORIAMENTE DOIS!!! Se um nao for usado, colocar 0
# -----
# Se quiser usar o numero total de setores, colocar 1919
5
0
#####
# P A R A M E T R O S   D O   D I S C O
#
# Tempo de troca de trilha
1.663
# Tempo medio de busca
8.7015
# Tempo de full stroke
15.36
# Atraso de escrita (write delay)
0.0
# Tempo de busca por 1..10 trilhas, um por linha
1.66
1.76
1.77
1.82

```

```

1.99
2.09
2.14
2.20
2.22
2.35
# Tempo de troca de cabeca
0.999
# Rotacoes por minuto (RPM)
7200
# Numero de superficies de dado
10
# Numero de cilindros
8057
# Tempo de transferencia de um setor
0.106
# Numero de zonas
9
# Dados das zonas (inicio, fim, setores por trilha, um por linha)
# Dados da zona 1
0
615
256
#
616
1823
252
#
1824
3375
240
#
3376
4143
238
#
4144
5199
224
#
5200
5975
208
#
5976
6583
196
#
6584
7343
182
#
7344
8056
168
##### [ EOF ]

```

Anexo C

Arquivos de configuração e resultados do DiskSim

C.1 Arquivo de parâmetros de disco no DiskSim

Disk brand name: QUANTUM_QM39100TD-SW	Slip: 589275	
Access time (in msec): -2.0	Slip: 591794	
Seek time (in msec): -5.0	Slip: 599352	
Single cylinder seek time: 1.66300	Slip: 1035697	
Average seek time:	Slip: 1190435	
atlas_III.seek	Slip: 1354801	
Full strobe seek time: 15.36000	Slip: 1367405	
Add. write settling delay: 0.00000	Slip: 1502250	
HPL seek equation values: 0 0 0 0 0 0	Slip: 1570676	
First 10 seek times: 1.66 1.76 1.77 1.82 1.99 ∞	Slip: 1999037	
2.09 2.14 2.20 2.22 2.35	Slip: 2773978	
Head switch time: 0.99900	Slip: 2773979	
Rotation speed (in rpms): 7200	Slip: 2773980	
Percent error in rpms: 0.0	Slip: 594314	
Number of data surfaces: 10	Slip: 596834	
Number of cylinders: 8057	Slip: 586755	
Blocks per disk: 17783250	Slip: 601875	
Per-request overhead time: 0.000000	Slip: 584236	
Time scale for overheads: 1.0	Slip: 604396	
Bulk sector transfer time: 0.10600	Slip: 581717	
Hold bus entire read xfer: 0	Slip: 606917	
Hold bus entire write xfer: 0	Number of defects: 0	
Allow almost read hits: 0	Band #3	
Allow sneaky full read hits: 0	First cylinder number: 1824	
Allow sneaky partial read hits: 0	Last cylinder number: 3375	
Allow sneaky intermediate read hits: 0	Blocks per track: 240	
Allow read hits on write data: 1	Offset of first block: 0.000000	
Allow write prebuffering: 0	Empty space at zone front: 0	
Preseeking level: 0	Skew for track switch: 71.000000	
Never disconnect: 0	Skew for cylinder switch: 81.000000	
Print stats for disk: 1	Number of spares: 0	
Avg sectors per cylinder: 2207	Number of slips: 5	
Max queue length at disk: 1	Slip: 1539463	
Scheduling policy: 1	Slip: 1639406	
Cylinder mapping strategy: 0	Slip: 1913779	
Write initiation delay: 0.0	Slip: 2828533	
Read initiation delay: 0.0	Slip: 3366781	
Sequential stream scheme: 0	Number of defects: 0	
Maximum concat size: 0	Band #4	
Overlapping request scheme: 0	First cylinder number: 3376	
Sequential stream diff maximum: 0	Last cylinder number: 4143	
Scheduling timeout scheme: 0	Blocks per track: 238	
Timeout time/weight: 0	Offset of first block: 0.000000	
Timeout scheduling: 0	Empty space at zone front: 0	
Scheduling priority scheme: 0	Skew for track switch: 70.000000	
Priority scheduling: 0	Skew for cylinder switch: 81.000000	
Number of buffer segments: 6	Number of spares: 0	
Maximum number of write segments: 1	Number of slips: 6	
Segment size (in blks): 64	Slip: 413741	
Use separate write segment: 0	Slip: 438564	
Low (write) water mark: 0.00	Slip: 763719	
High (read) water mark: 0.00	Slip: 813699	
Set watermark by reqsize: 1	Slip: 838463	
Calc sector by sector: 1	Slip: 874162	
Enable caching in buffer: 1	Number of defects: 0	
Buffer continuous read: 4	Band #5	
Minimum read-ahead (blks): 0	First cylinder number: 4144	
Maximum read-ahead (blks): 64	Last cylinder number: 5199	
Read-ahead over requested: 0	Blocks per track: 224	
Read-ahead on idle hit: 0	Offset of first block: 0.000000	
Read any free blocks: 0	Empty space at zone front: 0	
Fast write level: 2	Skew for track switch: 65.000000	
Immediate buffer read: 1	Skew for cylinder switch: 75.000000	
Immediate buffer write: 1	Number of spares: 0	

Combine seq writes:	1	Number of slips:	7
Stop prefetch in sector:	0	Slip: 451189	
Disconnect write if seek:	0	Slip: 1073879	
Write hit stop prefetch:	1	Slip: 1154587	
Read directly to buffer:	1	Slip: 1156827	
Immed transfer partial hit:	1	Slip: 1933162	
Read hit over. after read:	0.37400	Slip: 2287972	
Read hit over. after write:	0.33500	Slip: 2323796	
Read miss over. after read:	0.00000	Number of defects:	0
Read miss over. after write:	0.00000	Band #6	
Write hit over. after read:	0.41400	First cylinder number:	5200
Write hit over. after write:	0.39600	Last cylinder number:	5975
Write miss over. after read:	0.47828	Blocks per track:	208
Write miss over. after write:	0.50020	Offset of first block:	0.000000
Read completion overhead:	0.00000	Empty space at zone front:	0
Write completion overhead:	0.00000	Skew for track switch:	61.000000
Data preparation overhead:	0.00000	Skew for cylinder switch:	71.000000
First reselect overhead:	0.00000	Number of spares:	0
Other reselect overhead:	0.00000	Number of slips:	2
Read disconnect afterread:	0.00000	Slip: 865580	
Read disconnect afterwrite:	0.00000	Slip: 1134869	
Write disconnect overhead:	0.00000	Number of defects:	0
Extra write disconnect:	0	Band #7	
Extradi sc command overhead:	0.00000	First cylinder number:	5976
Extradi sc disconnect overhead:	0.00000	Last cylinder number:	6583
Extradi sc inter-di sconnect delay:	0.00000	Blocks per track:	196
Extradi sc 2nd disconnect overhead:	0.00000	Offset of first block:	0.000000
Extradi sc seek del ta:	0.00000	Empty space at zone front:	0
Minimum seek delay:	0.00000	Skew for track switch:	57.000000
LBN-to-PBN mapping scheme:	0	Skew for cylinder switch:	67.000000
Sparing scheme used:	7	Number of spares:	0
Rangesize for sparing:	1	Number of slips:	2
Number of bands:	9	Slip: 208490	
Band #1		Slip: 372067	
First cylinder number:	0	Number of defects:	0
Last cylinder number:	615	Band #8	
Blocks per track:	256	First cylinder number:	6584
Offset of first block:	0.000000	Last cylinder number:	7343
Empty space at zone front:	128000	Blocks per track:	182
Skew for track switch:	75.000000	Offset of first block:	0.000000
Skew for cylinder switch:	86.000000	Empty space at zone front:	0
Number of spares:	0	Skew for track switch:	53.000000
Number of slips:	4	Skew for cylinder switch:	62.000000
Slip: 272037		Number of spares:	0
Slip: 1464716		Number of slips:	2
Slip: 1467276		Slip: 559442	
Slip: 1508869		Slip: 560014	
Number of defects:	0	Number of defects:	0
Band #2		Band #9	
First cylinder number:	616	First cylinder number:	7344
Last cylinder number:	1823	Last cylinder number:	8056
Blocks per track:	252	Blocks per track:	168
Offset of first block:	0.000000	Offset of first block:	0.000000
Empty space at zone front:	0	Empty space at zone front:	0
Skew for track switch:	74.000000	Skew for track switch:	49.000000
Skew for cylinder switch:	85.000000	Skew for cylinder switch:	57.000000
Number of spares:	0	Number of spares:	0
Number of slips:	25	Number of slips:	3
Slip: 406795		Slip: 175954	
Slip: 406796		Slip: 390386	
Slip: 406797		Slip: 1039498	
Slip: 589274		Number of defects:	0

C.2 Configuração de RAID 0 com 3 discos e geração de carga sintética

Byte Order (Endian): Big
Init Seed: PID
Real Seed: PID
Statistic warm-up period: 2 seconds
Stat (dist) definition file: statdefs
Output file for trace of I/O requests
simulated: 0

I/O Subsystem Input Parameters

PRINTED I/O SUBSYSTEM STATISTICS

```
Print driver size stats?      1
Print driver locality stats?  1
Print driver blocking stats?  1
Print driver interference stats? 1
Print driver queue stats?     1
Print driver crit stats?      1
Print driver idle stats?      1
Print driver intarr stats?    1
Print driver streak stats?    1
Print driver stamp stats?     1
Print driver per-device stats? 1
Print bus idle stats?         0
Print bus arbit wait stats?   0
Print controller cache stats? 0
Print controller size stats?  0
Print controller locality stats? 0
Print controller blocking stats? 0
Print controller interference stats? 0
Print controller queue stats? 0
Print controller crit stats?  0
Print controller idle stats?  0
Print controller intarr stats? 0
Print controller streak stats? 0
Print controller stamp stats? 0
Print controller per-device stats? 0
Print disk queue stats?       0
Print disk crit stats?        0
Print disk idle stats?        0
Print disk intarr stats?      0
Print disk size stats?        0
Print disk seek stats?        0
Print disk latency stats?     0
Print disk xfer stats?        0
Print disk acctime stats?     0
Print disk interfere stats?    0
Print disk buffer stats?      0
```

GENERAL I/O SUBSYSTEM PARAMETERS

I/O Trace Time Scale: 1.0
Number of I/O Mappings: 0

COMPONENT SPECIFICATIONS

Number of device drivers: 1

Device Driver Spec #1
drivers with Spec: 1
Device driver type: 1
Constant access time: 0.0
Scheduling policy: 3
Cylinder mapping strategy: 1
Write initiation delay: 0.0
Read initiation delay: 0.0
Sequential stream scheme: 0
Maximum concat size: 0
Overlapping request scheme: 0
Sequential stream diff maximum: 0
Scheduling timeout scheme: 0
Timeout time/weight: 30
Timeout scheduling: 3
Scheduling priority scheme: 0

PHYSICAL ORGANIZATION

Driver #1
of connected buses: 1
Connected buses: 1

Controller #1-3
of connected buses: 1
Connected buses: #+1

Bus #1
of utilized slots: 3
Slots: Controllers 1-3

Bus #2-4
of utilized slots: 2
Slots: Controllers #-1
Slots: Disks #-1

SYNCHRONIZATION

Number of synchronized sets: 1
Number of disks in set #1: 3
Synchronized disks: 1-3

LOGICAL ORGANIZATION

of system-level organizations: 1

Organization #1: Array Striped Noredun Whole
Number of disks: 3
Disks: 1-3
High-level device number: 0
Stripe unit (in sectors): 8
Synch writes for safety: 0
Number of copies: 2
Copy choice on read: 6
RMW vs. reconstruct: 0.5
Parity stripe unit: 8
Parity rotation type: 1
Time stamp interval: 0.000000
Time stamp start time: 60000.000000
Time stamp stop time: 10000000000.000000
Time stamp file name: stamps

of controller-level organizations: 0

Process-Flow Input Parameters

PRINTED PROCESS-FLOW STATISTICS

```
Print per-process stats?      0
Print per-CPU stats?          0
Print all interrupt stats?     0
Print sleep stats?            0
```

GENERAL PROCESS-FLOW PARAMETERS

Number of processors: 5
Process-Flow Time Scale: 1.0

SYNTHETIC I/O TRACE PARAMETERS

Number of generators: 5
Number of I/O requests to generate: 500000
Maximum time of trace generated (in seconds):
50000.0
System call/return with each request: 0
Think time from call to request: 0.0
Think time from request to return: 0.0

Generator description #1
Generators with description: 5

Priority scheduling:	3	Storage capacity per device (in blocks):	6168024
Use queueing in subsystem:	0	Number of storage devices:	1
Number of buses:	4	Blocking factor:	8
Bus Spec #1		Probability of sequential access:	0.0
# buses with Spec:	1	Probability of local access:	0.1
Bus Type:	2	Probability of read access:	0.2
Arbitration type:	1	Probability of time-critical request:	0.1
Arbitration time:	0.0	Probability of time-limited request:	0.3
Read block transfer time:	0.0	Time-limited think times (in milliseconds)	
Write block transfer time:	0.0	Type of distribution:	normal
Print stats for bus:	0	Mean:	30.0
		Variance:	100.0
Bus Spec #2		General inter-arrival times (in milliseconds)	
# buses with Spec:	3	Type of distribution:	exponential
Bus Type:	1	Base value:	0.0
Arbitration type:	1	Mean:	10.0
Arbitration time:	0.0	Sequential inter-arrival times (in milliseconds)	
Read block transfer time:	0.0512	Type of distribution:	exponential
Write block transfer time:	0.0512	Base value:	0.0
Print stats for bus:	1	Mean:	10.0
Number of controllers:	3	Local inter-arrival times (in milliseconds)	
Controller Spec #1		Type of distribution:	exponential
# controllers with Spec:	3	Base value:	0.0
Controller Type:	1	Mean:	10.0
Scale for delays:	0.0	Local distances (in blocks)	
Bulk sector transfer time:	0.0	Type of distribution:	normal
Maximum queue length:	0	Mean:	0.0
Print stats for controller:	1	Variance:	40000.0
Number of disks:	3	Sizes (in blocks)	
Disk Spec #1		Type of distribution:	normal
# disks with Spec:	3	Mean:	2.0
Disk brand name:	HP_C3323A	Variance:	2.0
Disk specification file:	diskspecs		

C.3 Arquivo com resultado parcial de simulação para um disco

Output file name: OUTPUT	System logorg #0 disk Request size average:
Input trace format: ascii	5.512894
I/O trace used: 0	System logorg #0 disk Request size std.dev.: 4.982230
Synthgen to be used?: 1	System logorg #0 disk Request size maximum: 56
Byte Order (Endian): Big (using detected value: Little)	System logorg #0 disk Read request size average: 5.512894
Init Seed: 777	System logorg #0 disk Read request size std.dev.: 4.982230
Real Seed: 777	System logorg #0 disk Read request size maximum: 56
Statistic warm-up period: 0.000000 seconds	System logorg #0 disk Write request size average: 0.000000
Checkpoint to null every: 0.000000 seconds	System logorg #0 disk Write request size std.dev.: 0.000000
Stat (dist) definition file: statdefs	System logorg #0 disk Write request size maximum: 0
Output file for I/O requests simulated: 0	System logorg #0 disk Instantaneous queue length average: 1.000000
I/O Subsystem Input Parameters	System logorg #0 disk Instantaneous queue length std.dev.: 0.000000
-----	System logorg #0 disk Instantaneous queue length maximum: 1
PRINTED I/O SUBSYSTEM STATISTICS	System logorg #0 disk Sub-optimal mapping penalty average: 0.000000
Print driver size stats? 1	System logorg #0 disk Sub-optimal mapping penalty std.dev.: 0.000000
Print driver locality stats? 1	System logorg #0 disk Sub-optimal mapping penalty maximum: 0
Print driver blocking stats? 1	System logorg #0 disk Seeks of zero distance: 23
Print driver interference stats? 1	System logorg #0 disk Seek distance average: 1878.221506
Print driver queue stats? 1	
Print driver crit stats? 1	
Print driver idle stats? 1	
Print driver intarr stats? 1	
Print driver streak stats? 1	
Print driver stamp stats? 1	
Print driver per-device stats? 1	
Print bus idle stats? 1	
Print bus arbwait stats? 1	
Print controller cache stats? 1	

Print controller size stats? 1
 Print controller locality stats? 1
 Print controller blocking stats? 1
 Print controller interference stats? 1
 Print controller queue stats? 1
 Print controller crit stats? 1
 Print controller idle stats? 1
 Print controller intarr stats? 1
 Print controller streak stats? 1
 Print controller stamp stats? 1
 Print controller per-device stats? 1
 Print device queue stats? 1
 Print device crit stats? 1
 Print device idle stats? 1
 Print device intarr stats? 1
 Print device size stats? 1
 Print device seek stats? 1
 Print device latency stats? 1
 Print device xfer stats? 1
 Print device acctime stats? 1
 Print device interfere stats? 1
 Print device buffer stats? 1

Readparams complete
 Kicking off cpu #0
 First event occurs at time 23.002569
 Initialization complete
 Simulation complete

SIMULATION STATISTICS

Total time of run: 1199850.330442
 Warm-up time: 0.000000

PROCESS FLOW STATISTICS

CPU #0 Total idle milliseconds: 0.000000
 CPU #0 Idle time per processor: 0.000000
 CPU #0 Percentage idle cycles: 0.000000
 CPU #0 Total false idle ms: 0.000000
 CPU #0 False idle time per CPU: 0.000000
 CPU #0 Percentage false idle cycles: 0.000000
 CPU #0 Total idle work ms: 0.000000
 CPU #0 Context Switches: 0
 CPU #0 Time spent context switching: 0.000000
 CPU #0 Percentage switching cycles: 0.000000
 CPU #0 Number of interrupts: 363643
 CPU #0 Total time in interrupts: 0.000
 CPU #0 Percentage interrupt cycles: 0.000000
 CPU #0 Time-Critical request count: 0
 CPU #0 Time-Critical Response time average: 0.000000
 CPU #0 Time-Critical Response time std.dev.: 0.000000
 CPU #0 Time-Critical Response time maximum: 0
 CPU #0 Time-Limited request count: 0
 CPU #0 Time-Limited Response time average: 0.000000
 CPU #0 Time-Limited Response time std.dev.: 0.000000
 CPU #0 Time-Limited Response time maximum: 0
 CPU #0 Time-Noncritical request count: 79999
 CPU #0 Time-Noncritical Response time average: 27.560186
 CPU #0 Time-Noncritical Response time: std.dev.: 18.935035
 CPU #0 Time-Noncritical Response time maximum: 170.344578
 CPU #0 Number of I/O interrupts: 243658
 CPU #0 Time spent in I/O interrupts: 0.000000
 CPU #0 Number of clock interrupts: 119985
 CPU #0 Time spent in clock interrupts: 0.000000
 Number of sleep events: 0
 Number of I/O sleep events: 0

System logorg #0 disk Seek distance std.dev.: 1599.006627
 System logorg #0 disk Seek distance maximum: 6549
 System logorg #0 disk Seek time average: 7.271907
 System logorg #0 disk Seek time std.dev.: 3.252169
 System logorg #0 disk Seek time maximum: 16.053525
 System logorg #0 disk Full rotation time: 5.980265
 System logorg #0 disk Zero rotate latency: 0.000000
 System logorg #0 disk Rotational latency average: 2.841384
 System logorg #0 disk Rotational latency std.dev.: 1.721446
 System logorg #0 disk Rotational latency maximum: 5.980193
 System logorg #0 disk Transfer time average: 0.355478
 System logorg #0 disk Transfer time std.dev.: 0.938150
 System logorg #0 disk Transfer time maximum: 7.667007
 System logorg #0 disk Positioning time average: 10.113292
 System logorg #0 disk Positioning time std.dev.: 3.683232
 System logorg #0 disk Positioning time maximum: 21.805213
 System logorg #0 disk Access time average: 10.468769
 System logorg #0 disk Access time std.dev.: 3.687326
 System logorg #0 disk Access time maximum: 24.304774
 System logorg #0 disk Sequential interference: 0
 System logorg #0 disk Local interference: 0
 System logorg #0 disk Number of buffer accesses: 79999
 System logorg #0 disk Buffer hit ratio: 1 0.000013
 System logorg #0 disk Buffer miss ratio: 79998 0.999987
 System logorg #0 disk Buffer read hit ratio: 1 0.000013 0.000013
 System logorg #0 disk Buffer prepend hit ratio: 0 0.000000
 System logorg #0 disk Buffer append hit ratio: 0 0.000000
 System logorg #0 disk Write combinations: 0 0.000000
 System logorg #0 disk Ongoing read-ahead hit ratio: 0 0.000000 0.000000
 System logorg #0 disk Average read-ahead hit size: 0.000000
 System logorg #0 disk Average remaining read-ahead: 0.000000
 System logorg #0 disk Partial read hit ratio: 0 0.000000 0.000000
 System logorg #0 disk Average partial hit size: 0.000000
 System logorg #0 disk Average remaining partial: 0.000000
 System logorg #0 disk Total disk bus wait time: 0.000000
 System logorg #0 disk Number of disk bus waits: 81829

I/O DRIVER STATISTICS

Iodriver Total Requests handled: 79999
 Iodriver Requests per second: 66.674149
 Iodriver Completely idle time: 278542.814196
 0.232148
 Iodriver Response time average: 27.560186
 Iodriver Response time std.dev.: 18.935035
 Iodriver Response time maximum: 170.344578

Average sleep time: 0.000000
 Average I/O sleep time: 0.000000
 PROCESS STATISTICS
 Process Total computation time: 1199850.330442
 Process Last event time: 1199850.330442
 Process Number of I/O requests: 80000
 Process Number of read requests: 80000
 Process Number of C-switches: 0
 Process Number of sleeps: 0
 Process Average sleep time: 0.000000
 Process Number of I/O sleeps: 0
 Process Average I/O sleep time: 0.000000
 Process False idle time: 0.000000
 Process Read Time limits measured: 0
 Process Read Time limit duration average: 0.000000
 Process Read Time limit duration std.dev.: 0.000000
 Process Read Time limit duration maximum: 0
 Process Write Time limits measured: 0
 Process Write Time limit duration average: 0.000000
 Process Write Time limit duration std.dev.: 0.000000
 Process Write Time limit duration maximum: 0
 Process Read Time limits missed: 0
 Process Missed Read Time limit duration average: 0.000000
 Process Missed Read Time limit duration std.dev.: 0.000000
 Process Missed Read Time limit duration maximum: 0
 Process Write Time limits missed: 0
 Process Missed Write Time limit duration average: 0.000000
 Process Missed Write Time limit duration std.dev.: 0.000000
 Process Missed Write Time limit duration maximum: 0

STORAGE SUBSYSTEM STATISTICS

SYSTEM-LEVEL LOGORG STATISTICS

System Logical Organization #0
 System logorg #0 Number of requests: 80000
 System logorg #0 Number of read requests: 80000
 1.000000
 System logorg #0 Number of accesses: 80000
 System logorg #0 Number of read accesses: 80000
 1.000000
 System logorg #0 Average outstanding: 1.837552
 System logorg #0 Maximum outstanding: 14
 System logorg #0 Avg nonzero outstanding: 2.393107
 System logorg #0 Completely idle time: 278542.814196
 System logorg #0 Response time average: 27.560186
 System logorg #0 Response time std.dev.: 18.935035
 System logorg #0 Response time maximum: 170.344578
 System logorg #0 Time-critical reads: 0
 System logorg #0 Time-critical writes: 0
 System logorg #0 Inter-request distance average: -39.734850
 System logorg #0 Inter-request distance std.dev.: 3624116.858344
 System logorg #0 Inter-request distance maximum: 8833557
 System logorg #0 Sequential reads: 0
 0.000000 0.000000
 System logorg #0 Sequential writes: 0
 0.000000 0.000000
 System logorg #0 Interleaved reads: 1

Iodriver Overlaps combined: 0
 0.000000
 Iodriver Read overlaps combined: 0
 0.000000 0.000000
 Iodriver Critical Reads: 0
 0.000000
 Iodriver Critical Read Response time average: 0.000000
 Iodriver Critical Read Response time std.dev.: 0.000000
 Iodriver Critical Read Response time maximum: 0
 Iodriver Non-Critical Reads: 79999
 0.999988
 Iodriver Non-Critical Read Response time average: 27.560186
 Iodriver Non-Critical Read Response time std.dev.: 18.935035
 Iodriver Non-Critical Read Response time maximum: 170.344578
 Iodriver Critical Writes: 0
 0.000000
 Iodriver Critical Write Response time average: 0.000000
 Iodriver Critical Write Response time std.dev.: 0.000000
 Iodriver Critical Write Response time maximum: 0
 Iodriver Non-Critical Writes: 0
 0.000000
 Iodriver Non-Critical Write Response time average: 0.000000
 Iodriver Non-Critical Write Response time std.dev.: 0.000000
 Iodriver Non-Critical Write Response time maximum: 0
 Iodriver Number of reads: 80000 1.000000
 Iodriver Number of writes: 0 0.000000
 Iodriver Sequential reads: 0 0.000000
 0.000000
 Iodriver Sequential writes: 0 0.000000
 0.000000
 Iodriver Base SPTF/SDF Different: 0 / 80000
 0.000000
 Iodriver Timeout SPTF/SDF Different: 0 /
 0 0.000000
 Iodriver Priority SPTF/SDF Different: 0 /
 0 0.000000
 Iodriver runlistlen: 2204787.344842
 Iodriver runoutstanding: 921307.516246
 Iodriver simtime: 1199850.330442
 Iodriver warmuptime: 0.000000
 Iodriver setsize: 1
 Iodriver Average # requests: 1.837552
 Iodriver Maximum # requests: 14
 Iodriver End # requests: 1
 Iodriver Average queue length: 1.069700
 Iodriver Maximum queue length: 13
 Iodriver End queued requests: 0
 Iodriver Queue time average: 16.043498
 Iodriver Queue time std.dev.: 19.223369
 Iodriver Queue time maximum: 164.548472
 Iodriver Avg # read requests: 1.837552
 Iodriver Max # read requests: 14
 Iodriver Avg # write requests: 0.000000
 Iodriver Max # write requests: 0
 Iodriver Physical access time average: 11.516488
 Iodriver Physical access time std.dev.: 3.725684
 Iodriver Physical access time maximum: 25.260754
 Iodriver Inter-arrival time average: 14.998129
 Iodriver Inter-arrival time std.dev.: 15.024533
 Iodriver Inter-arrival time maximum: 162.557912
 Iodriver Read inter-arrival average: 14.998129
 Iodriver Read inter-arrival std.dev.: 15.024533
 Iodriver Read inter-arrival maximum: 162.557912
 Iodriver Write inter-arrival average:

System	logorg #0 Interleaved writes:	0	0.000013	0.000013	0.000000	Iodriver Write inter-arrival std.dev.:	0	0.000000
System	logorg #0 Logical sequential reads:	0	0.000000	0.000000	0.000000	Iodriver Write inter-arrival maximum:	0	0.000000
System	logorg #0 Logical sequential writes:	0	0.000000	0.000000	0.000000	Iodriver Number of idle periods:	18582	
System	logorg #0 Sequential disk switches:	0	0.000000	0.000000	0.000000	Iodriver Idle period length average:	14.989926	
System	logorg #0 Logical local accesses:	0	0.000000	0.000000	0.000000	Iodriver Idle period length std.dev.:	14.957879	
System	logorg #0 Local disk switches:	0	0.000000	0.000000	0.000000	Iodriver Idle period length maximum:	142.016122	
System	logorg #0 Blocking statistics					Iodriver Request size average:	5.512850	
System	logorg #0 Blocking factor: 1					Iodriver Request size std.dev.:	4.982214	
80000	1.000000					Iodriver Request size maximum:	56	
System	logorg #0 Blocking factor: 2					Iodriver Read request size average:	5.512850	
36106	0.451325					Iodriver Read request size std.dev.:	4.982214	
System	logorg #0 Blocking factor: 3					Iodriver Read request size maximum:	56	
21624	0.270300					Iodriver Write request size average:	0.000000	
System	logorg #0 Blocking factor: 4					Iodriver Write request size std.dev.:	0.000000	
14410	0.180125					Iodriver Write request size maximum:	0	
System	logorg #0 Blocking factor: 5					Iodriver Instantaneous queue length average:	2.069000	
10233	0.127913					Iodriver Instantaneous queue length std.dev.:	1.449850	
System	logorg #0 Blocking factor: 6					Iodriver Instantaneous queue length maximum:	13	
7698	0.096225					Iodriver Sub-optimal mapping penalty average:	0.000000	
System	logorg #0 Blocking factor: 7					Iodriver Sub-optimal mapping penalty std.dev.:	0.000000	
5850	0.073125					Iodriver Sub-optimal mapping penalty maximum:	0	
System	logorg #0 Blocking factor: 8							
4499	0.056238							
System	logorg #0 Blocking factor: 9							
3593	0.044913							
System	logorg #0 Blocking factor: 10							
2772	0.034650							
System	logorg #0 Blocking factor: 11							
2178	0.027225							
System	logorg #0 Blocking factor: 12							
1769	0.022113							
System	logorg #0 Blocking factor: 13							
1393	0.017413							
System	logorg #0 Blocking factor: 14							
1152	0.014400							
System	logorg #0 Blocking factor: 15							
868	0.010850							
System	logorg #0 Alignment statistics							
System	logorg #0 Alignment factor: 1							
80000	1.000000							
System	logorg #0 Alignment factor: 2							
40077	0.500962							
System	logorg #0 Alignment factor: 3							
26606	0.332575							
System	logorg #0 Alignment factor: 4							
20037	0.250462							
System	logorg #0 Alignment factor: 5							
15832	0.197900							
System	logorg #0 Alignment factor: 6							
13234	0.165425							
System	logorg #0 Alignment factor: 7							
11472	0.143400							
System	logorg #0 Alignment factor: 8							
10093	0.126163							
System	logorg #0 Alignment factor: 9							
8886	0.111075							
System	logorg #0 Inter-arrival time average:							
	14.998129							
System	logorg #0 Inter-arrival time std.dev.:							
	15.024533							
System	logorg #0 Inter-arrival time maximum:							
	162.557912							
System	logorg #0 Read inter-arrival average:							
	14.998129							
System	logorg #0 Read inter-arrival std.dev.:							
	15.024533							
System	logorg #0 Read inter-arrival maximum:							
	162.557912							
System	logorg #0 Write inter-arrival average:							
	0.000000							
System	logorg #0 Write inter-arrival std.dev.:							
	0.000000							
System	logorg #0 Write inter-arrival maximum:							
	0.000000							

DISK STATISTICS			

Disk Total Requests handled:	79999		
Disk Requests per second:	66.674149		
Disk Completely idle time:	278542.814196		
	0.232148		
Disk Response time average:	11.516488		
Disk Response time std.dev.:	3.725684		
Disk Response time maximum:	25.260754		
Disk Overlaps combined:	0	0.000000	
Disk Read overlaps combined:	0	0.000000	
	0.000000		
Disk Critical Reads:	0		
	0.000000		
Disk Critical Read Response time average:	0.000000		
Disk Critical Read Response time std.dev.:	0.000000		
Disk Critical Read Response time maximum:	0		
Disk Non-Critical Reads:	79999		
	1.000000		
Disk Non-Critical Read Response time average:	11.516488		
Disk Non-Critical Read Response time std.dev.:	3.725684		
Disk Non-Critical Read Response time maximum:	25.260754		
Disk Critical Writes:	0		
	0.000000		
Disk Critical Write Response time average:	0.000000		
Disk Critical Write Response time std.dev.:	0.000000		
Disk Critical Write Response time maximum:	0		
Disk Non-Critical Writes:	0		
	0.000000		
Disk Non-Critical Write Response time average:	0.000000		
Disk Non-Critical Write Response time std.dev.:	0.000000		
Disk Non-Critical Write Response time maximum:	0		
Disk Number of reads:	79999	1.000000	
Disk Number of writes:	0	0.000000	
Disk Sequential reads:	0	0.000000	
	0.000000		
Disk Sequential writes:	0	0.000000	
	0.000000		
Disk Base SPTF/SDF Different:	0 /	0	

0	0.000000
System logorg #0 Number of streaks: 1	Disk Timeout SPTF/SDF Different: 0 / 0
System logorg #0 Streak length average: 80000.000000	0.000000
System logorg #0 Streak length std.dev.: 0.000000	Disk Priority SPTF/SDF Different: 0 / 0
System logorg #0 Streak length maximum: 80000	0.000000
System logorg #0 Request size average: 5.512850	Disk runl istlen: 921307.516246
System logorg #0 Request size std.dev.: 4.982214	Disk runoutstanding: 921307.516246
System logorg #0 Request size maximum: 56	Disk simtime: 1199850.330442
System logorg #0 Read request size average: 5.512850	Disk warmuptime: 0.000000
System logorg #0 Read request size std.dev.: 4.982214	Disk setsize: 1
System logorg #0 Read request size maximum: 56	Disk Average # requests: 0.767852
System logorg #0 Write request size average: 0.000000	Disk Maximum # requests: 1
System logorg #0 Write request size std.dev.: 0.000000	Disk End # requests: 0
System logorg #0 Write request size maximum: 0	Disk Average queue length: 0.000000
System logorg #0 Number of idle periods: 18582	Disk Maximum queue length: 1
System logorg #0 Idle period length average: 14.989926	Disk End queued requests: 0
System logorg #0 Idle period length std.dev.: 14.957879	Disk Queue time average: 0.000000
System logorg #0 Idle period length maximum: 142.016122	Disk Queue time std.dev.: 0.000000
System logorg #0 disk Total Requests handled: 79999	Disk Queue time maximum: 0
System logorg #0 disk Requests per second: 66.674149	Disk Avg # read requests: 0.767852
System logorg #0 disk Completely idle time: 278542.814196 0.232148	Disk Max # read requests: 1
System logorg #0 disk Response time average: 11.516488	Disk Avg # write requests: 0.000000
System logorg #0 disk Response time std.dev.: 3.725684	Disk Max # write requests: 0
System logorg #0 disk Response time maximum: 25.260754	Disk Physical access time average: 11.516488
System logorg #0 disk Overlaps combined: 0 0.000000	Disk Physical access time std.dev.: 3.725684
System logorg #0 disk Read overlaps combined: 0 0.000000 0.000000	Disk Physical access time maximum: 25.260754
System logorg #0 disk Critical Reads: 0 0.000000	Disk Inter-arrival time average: 14.997950
System logorg #0 disk Critical Read Response time average: 0.000000	Disk Inter-arrival time std.dev.: 10.206451
System logorg #0 disk Critical Read Response time std.dev.: 0.000000	Disk Inter-arrival time maximum: 149.410602
System logorg #0 disk Critical Read Response time maximum: 0	Disk Read inter-arrival average: 14.997950
System logorg #0 disk Non-Critical Reads: 79999 1.000000	Disk Read inter-arrival std.dev.: 10.206451
System logorg #0 disk Non-Critical Read Response time average: 11.516488	Disk Read inter-arrival maximum: 149.410602
System logorg #0 disk Non-Critical Read Response time std.dev.: 3.725684	Disk Write inter-arrival average: 0.000000
System logorg #0 disk Non-Critical Read Response time maximum: 25.260754	Disk Write inter-arrival std.dev.: 0.000000
System logorg #0 disk Critical Writes: 0 0.000000	Disk Write inter-arrival maximum: 0
System logorg #0 disk Critical Write Response time average: 0.000000	Disk Number of idle periods: 80000
System logorg #0 disk Critical Write Response time std.dev.: 0.000000	Disk Idle period length average: 3.481785
System logorg #0 disk Critical Write Response time maximum: 0	Disk Idle period length std.dev.: 9.593626
System logorg #0 disk Non-Critical Writes: 0 0.000000	Disk Idle period length maximum: 142.016122
System logorg #0 disk Non-Critical Write Response time average: 0.000000	Disk Request size average: 5.512894
System logorg #0 disk Non-Critical Write Response time std.dev.: 0.000000	Disk Request size std.dev.: 4.982230
System logorg #0 disk Non-Critical Write Response time maximum: 0	Disk Request size maximum: 56
System logorg #0 disk Number of reads: 79999 1.000000	Disk Read request size average: 5.512894
System logorg #0 disk Number of writes: 0	Disk Read request size std.dev.: 4.982230
	Disk Read request size maximum: 56
	Disk Write request size average: 0.000000
	Disk Write request size std.dev.: 0.000000
	Disk Write request size maximum: 0
	Disk Instantaneous queue length average: 1.000000
	Disk Instantaneous queue length std.dev.: 0.000000
	Disk Instantaneous queue length maximum: 1
	Disk Sub-optimal mapping penalty average: 0.000000
	Disk Sub-optimal mapping penalty std.dev.: 0.000000
	Disk Sub-optimal mapping penalty maximum: 0
	Disk Seeks of zero distance: 23 0.000288
	Disk Seek distance average: 1878.221506
	Disk Seek distance std.dev.: 1599.006627
	Disk Seek distance maximum: 6549
	Disk Seek time average: 7.271907
	Disk Seek time std.dev.: 3.252169
	Disk Seek time maximum: 16.053525
	Disk Full rotation time: 5.980265
	Disk Zero rotate latency: 0 0.000000
	Disk Rotational latency average: 2.841384
	Disk Rotational latency std.dev.: 1.721446
	Disk Rotational latency maximum: 5.980193
	Disk Transfer time average: 0.355478
	Disk Transfer time std.dev.: 0.938150
	Disk Transfer time maximum: 7.667007
	Disk Positioning time average: 10.113292
	Disk Positioning time std.dev.: 3.683232
	Disk Positioning time maximum: 21.805213
	Disk Access time average: 10.468769
	Disk Access time std.dev.: 3.687326
	Disk Access time maximum: 24.304774
	Disk Sequential interference: 0

0.000000		Disk Local interference:	0
System logorg #0 disk Sequential reads:	0	Disk Number of buffer accesses:	79999
0.000000	0.000000	Disk Buffer hit ratio:	1 0.000013
System logorg #0 disk Sequential writes:	0	Disk Buffer miss ratio:	79998
0.000000	0.000000	0.999987	
System logorg #0 disk Base SPTF/SDF Different:		Disk Buffer read hit ratio:	1
0 / 0	0.000000	0.000013	0.000013
System logorg #0 disk Timeout SPTF/SDF Different:		Disk Buffer prepend hit ratio:	0
0 / 0	0.000000	0.000000	
System logorg #0 disk Priority SPTF/SDF Different:	0 / 0	Disk Buffer append hit ratio:	0
0.000000	0.000000	0.000000	
System logorg #0 disk runlistlen:		Disk Write combinations:	0
921307.516246		0.000000	
System logorg #0 disk runoutstanding:		Disk Ongoing read-ahead hit ratio:	0
921307.516246		0.000000	0.000000
System logorg #0 disk simtime:		Disk Average read-ahead hit size:	0.000000
1199850.330442		Disk Average remaining read-ahead:	0.000000
System logorg #0 disk warmuptime:		Disk Partial read hit ratio:	0 0.000000
0.000000		0.000000	
System logorg #0 disk setsize:	1	Disk Average partial hit size:	0.000000
System logorg #0 disk Average # requests:		Disk Average remaining partial:	0.000000
0.767852		Disk Total disk bus wait time:	0.000000
System logorg #0 disk Maximum # requests:	1	Disk Number of disk bus waits:	81829
System logorg #0 disk End # requests:	0		
System logorg #0 disk Average queue length:			
0.000000			
System logorg #0 disk Maximum queue length:	1	SIMPLEDISK STATISTICS	
System logorg #0 disk End queued requests:	0	-----	
System logorg #0 disk Queue time average:		No simpledisk requests encountered	
0.000000			
System logorg #0 disk Queue time std.dev.:		CONTROLLER STATISTICS	
0.000000		-----	
System logorg #0 disk Queue time maximum:	0	Controller #0	
System logorg #0 disk Avg # read requests:		Total controller bus wait time:	0.000000
0.767852			
System logorg #0 disk Max # read requests:	1	BUS STATISTICS	
System logorg #0 disk Avg # write requests:		-----	
0.000000			
System logorg #0 disk Max # write requests:	0	Bus #1	
System logorg #0 disk Physical access time average:		Bus #1 Total utilization time:	95366.01
11.516488		0.07948	
System logorg #0 disk Physical access time std.dev.:		Bus #1 Bus idle period length average:	6.825009
3.725684		Bus #1 Bus idle period length std.dev.:	8.012117
System logorg #0 disk Physical access time maximum:		Bus #1 Bus idle period length maximum:	142.016122
25.260754		Bus #1 Number of arbitrations:	161829
System logorg #0 disk Inter-arrival time average:		Bus #1 Arbitration wait time average:	0.000000
14.997950		Bus #1 Arbitration wait time std.dev.:	0.000000
System logorg #0 disk Inter-arrival time std.dev.:		Bus #1 Arbitration wait time maximum:	0
10.206451			
System logorg #0 disk Inter-arrival time maximum:			
149.410602			
System logorg #0 disk Read inter-arrival average:			
14.997950			
System logorg #0 disk Read inter-arrival std.dev.:			
10.206451			
System logorg #0 disk Read inter-arrival maximum:			
149.410602			
System logorg #0 disk Write inter-arrival average:			
0.000000			
System logorg #0 disk Write inter-arrival std.dev.:			
0.000000			
System logorg #0 disk Write inter-arrival maximum:			
0			
System logorg #0 disk Number of idle periods:			
80000			
System logorg #0 disk Idle period length average:			
3.481785			
System logorg #0 disk Idle period length std.dev.:			
9.593626			
System logorg #0 disk Idle period length maximum:			
142.016122			

Apêndice A

----- Original Message -----

From: "Greg Ganger" <ganger@ece.cmu.edu>

To: "Daniel Correa Lobato" <dclobato@myself.com>

Cc: <ganger@eecs.umi.ch.edu>; <patt@eecs.umi.ch.edu>; <worthington@eecs.umi.ch.edu>

Sent: Tuesday, September 21, 1999 1:15 PM

Subject: Re: DiskSim Results

>
> Daniel,
>
> As the DiskSim web page states, we can not offer personal assistance to
> all people working with the simulator. I'm sure you can understand why
> it is not possible.
>
> Best luck!
>
> Greg
>
>
> [Charset iso-8859-1 unsupported, filtering to ASCII...]
> > Mr. Ganger, Mr. Worthington & Mr. Patt,
> >
> > Sorry for bother you again, Mr. Ganger, and for setting this message as High
> > Priority.
> >
> > I'm working on the specification of an Intelligent Simulation and Learning
> > Environment for RAID's and I'm using DiskSim to validate some aspects of my
> > master degree dissertation, which I'm supposed to finish by january/2000 here
> > at Universidade de Sao Paulo (<http://www.icmc.sc.usp.br>), but I'm having some
> > problems.
> >
> > According to some articles I read (e.g., Chen93b in Mr. Ganger's Ph.D.
> > Thesis), a RAID 5 organization can handle [MAX (1/G, 1/4)] Input-Outputs per
> > second (IO/s) relative to a RAID 0 organization with same group parity size
> > (G). This is the RMW-problem, which can be minimized, for instance, by using
> > the parity-logging or floating-parity techniques.
> >
> > I set up some RAID organizations (varying the value of G) to run at DiskSim in
> > order to get "real" values of this assertion and discuss with my professor
> > but, according to DiskSim's results, RAID 0's performance is ALWAYS around
> > 100 IO/s, and RAID 5 performance starts at 144 IO/s growing up to 339 IO/s
> > (see table below -- sr0-g2 stands for Raid 0, with G=2, and so on).
> >
> > "Config" "IOdriver Total Requests handled" "IOdriver Requests per second"
> > "sr0-g2" 499758 100,060391
> > "sr0-g3" 499740 100,020471
> > "sr0-g4" 499764 100,089097
> > "sr0-g5" 499633 100,101601
> > "sr0-g6" 499623 99,989409
> > "sr0-g7" 499599 99,962006
> > "sr0-g8" 499557 100,199205
> > "sr0-g9" 499567 100,054860
> > "sr5-g2" 899753 144,286985
> > "sr5-g3" 1299024 203,513082
> > "sr5-g4" 1699329 276,578517
> > "sr5-g5" 1699418 314,864206
> > "sr5-g6" 1700152 332,232025
> > "sr5-g7" 1699416 337,730697
> > "sr5-g8" 1700512 339,486944
> > "sr5-g9" 1699551 339,530043
> >

> > These results produces a curve that is somewhat the opposite of what Mr. Chen
> > states: the performance of a RAID5 grows until it reaches a maximum of around 4
> > times RAID 0 performance!

```

> >
> > |
> > I |
> > 0 |
> > / |
> > S | @ @ @ @ @ @ @ @ # Raid 0
> > | #@ #@ #@ #@ #@ #@ #@ @ Raid 5
> > | #@ #@ #@ #@ #@ #@ #@ #@
> > +-----+
> > | 2 3 4 5 6 7 8 9 G
> >

```

> > These results are really confusing me...

> > Another point: in DiskSim's Manual the "RMS vs. reconstruct" parameter is
> > defined as Integer [nonnegative], but it only accepts values between 0 and 1,
> > so I imagine that it's a probability or a %-value. Am I correct?

> > For all the simulations I run, I took par.synthraid5 (distributed with
> > DiskSim) as a start point (disks, buses and synthetic workload generator) for
> > the configuration file. If you want, I can send you the config files I used.

> > Thanks in advance for you attention (and reply) and my best regards,
> > Daniel

```

> > ---
> > Daniel Correa Lobato <dclobato@myself.com>    Phone +55 16 273 9675
> > Universidade de Sao Paulo - ICMC                Caixa Postal 668
> > M.Sc'ing in Distributed Systems                 13560-970 SP BRAZIL
> > S/MIME digital ID available                     ICQ# 8960384
> > PGP Key fingerprint: 3D5B0C15 45CC0BC8 82F2 91CC 5B6DACF9 CFB43F66

```

Esta página foi deixada intencionalmente em branco