Universidade de São Paulo
Instituto de Física de São Carlos

O gerenciamento e o suporte a aplicações em *clusters* de alta
disponibilidade utilizando objetos distribuídos

Cláudio Roberto de Vincenzi

Tese apresentada ao Instituto de
Física de São Carlos da
Universidade de São Paulo, para
obtenção do título de Doutor em
Ciências: Física Aplicada – opção
Física Computacional.

Orientador: Prof. Dr. Gonzalo Travieso

Departamento de Física e Informática
São Carlos – 2003

USP/IFSC/SBI

IFSC-USP SERVIÇO DE BIBLIOTECA
INFORMAÇÃO

OK

Vincenzi, Cláudio Roberto de

“O gerenciamento e o suporte a aplicações em *clusters* de alta disponibilidade utilizando objetos distribuídos” / Cláudio Roberto de Vincenzi – São Carlos, 2003

Tese (Doutorado) – Instituto de Física de São Carlos da Universidade de São Paulo, 2003 - Páginas: 335

Área: Física Aplicada – opção Física Computacional

Orientador: Prof. Dr. Gonzalo Travieso

1. Alta Disponibilidade; 2. Clusters; 3. Objetos Distribuídos.

I. Título

**UNIVERSIDADE
DE SÃO PAULO**

Instituto de Física de São Carlos

Av. Trabalhador São-carlense, 400
Caixa Postal 369
13560-970 - São Carlos - SP - Brasil

**MEMBROS DA COMISSÃO JULGADORA DA TESE DE
DOUTORADO DE CLÁUDIO ROBERTO DE VINCENZI,
APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS,
UNIVERSIDADE DE SÃO PAULO, EM 31-03-2003.**

COMISSÃO JULGADORA:

Prof. Dr. Gonzalo Travieso (Presidente) - IFSC-USP

Dr. Attilio Cucchieri - IFSC / USP

Prof. Dr. Jan Frans Willem Slaets - IFSC / USP

Prof. Dr. Onofre Trindade Júnior - ICMC / USP

Prof. Dr. Luis Carlos Trevelin - UFSCar

Dedicatória

A meus pais, Fausto e Célia, pelo incentivo constante,

À minha esposa Ana Lucia, pelo seu apoio inestimável,

Ao meu filho Lucas, pelo seu amor e companheirismo.

O GERENCIAMENTO E O SUPORTE A
APLICAÇÕES EM *CLUSTERS* DE ALTA
DISPONIBILIDADE UTILIZANDO OBJETOS
DISTRIBUÍDOS

Cláudio Roberto de Vincenzi

Orientador:
Prof. Dr. Gonzalo Travieso

Este trabalho procura averiguar as dificuldades e os problemas envolvidos na especificação e implementação de uma arquitetura de objetos de serviço para gerenciamento e suporte a aplicações distribuídas em uma plataforma de computadores em configuração tipo *cluster*, em particular daqueles que devem cumprir requisitos de alta disponibilidade. Tal arquitetura deve apresentar-se como uma infra-estrutura distribuída de serviços e compor uma camada intermediária entre as partes das aplicações distribuídas e os recursos do *cluster*, compondo desse modo uma camada de *middleware*.

Uma arquitetura de gerenciamento e de suporte a aplicações distribuídas, para que possa atingir plenamente seus objetivos, deve ser projetada como uma infra-estrutura distribuída de serviços de modo a oferecer acesso fácil e eficiente a estes. Neste trabalho, investigamos a possibilidade de se implementar tal infra-estrutura utilizando a arquitetura de objetos distribuídos OMG CORBA, a qual tem por objetivo facilitar a especificação, implementação e o acesso a interfaces de serviços de objetos distribuídos.

Uma infra-estrutura de serviços de suporte a aplicações e de gerenciamento de *cluster* é composta de dezenas de interfaces dos mais diversos tipos e finalidades, compondo vários sistemas e subsistemas distribuídos cujos serviços devem ser fácil e transparentemente acessíveis através da rede de comunicação que interconecta os nós do *cluster*. Essa rede tipicamente utiliza os protocolos de comunicação *Ethernet* IEEE 802.3, IP e TCP/UDP em seus níveis de enlace, rede e transporte, respectivamente. Por essa ótica,

CORBA atende de antemão vários dos requisitos necessários, porém em diversos aspectos da arquitetura e dos serviços houve necessidade de se averiguar com mais profundidade a capacidade de CORBA de atendê-los. Como exemplo, averiguou-se a capacidade de CORBA lidar com problemas ligados à falhas parciais de comunicação, comunicação em grupo, comunicação não-bloqueante e outras.

Um dos desafios da abordagem utilizando CORBA reside no fato de que se tem o pré-requisito de alta disponibilidade. A arquitetura proposta deve ser projetada levando-se em conta que certas partes do sistema, tais como subsistemas de armazenamento, processamento, rede, aplicações e outros podem, a qualquer momento, falhar ou tornar-se inoperantes. O grande desafio deste trabalho consiste assim em averiguar quais são as dificuldades de se utilizar CORBA como parte central da arquitetura, uma vez que a infra-estrutura obtida necessita ser ela própria um conjunto de serviços de alta disponibilidade, caso contrário ela não conseguirá cumprir seus objetivos. Investigaremos então possíveis modos de implementação de objetos CORBA que venham a cumprir os requisitos de alta disponibilidade necessários.

Por fim, este trabalho procura averiguar a possibilidade de se implementar tal arquitetura em *clusters* baseados em elementos de *hardware* e *software* disponíveis em larga escala e a baixo custo no mercado – tais como PCs ou servidores baseados em processador Intel rodando os sistemas operacionais Linux ou FreeBSD e interconectados por redes locais de baixo custo. Tais plataformas são popularmente conhecidas como *Beowulf Clusters*. Este aspecto da investigação terá influência no projeto da arquitetura proposta, uma vez que procuramos enfocar aspectos de alta escalabilidade para o *cluster*, uma tendência para o uso desse tipo de *cluster* em aplicações de alta demanda de processamento e recursos, tais servidores *web* e de banco de dados de grandes provedores de conteúdo e serviços da Internet.

ABSTRACT

This work intends to investigate for the difficulties and problems involved in the specification and implementation of an object-oriented architecture of services for management and support of distributed applications in a platform of computers in cluster configuration, in particular for those that must fulfill requirements of high availability. Such architecture must present itself as a infrastructure of distributed services and should compose an intermediate layer between the parts of the cluster distributed applications and the cluster resources, composing in this way a middleware layer.

In order to reach its objectives, this support and management architecture must be designed as a distributed infrastructure of services. By this way, it can offer easy and efficient access to its services. In this text, we investigate the possibility of implementing such infrastructure of services using the distributed object architecture proposed by OMG, the so-called CORBA architecture. CORBA has many objectives and features as to facilitate the specification, implementation and access to distributed service objects.

An infrastructure of services for the support and management of clusters and cluster applications is composed of a set of interfaces of the most diverse types and purposes, composing some distributed systems and subsystems whose services must be easy and transparently accessible through the communication networks that interconnects the cluster computer nodes. Cluster networks typically uses standard communication protocols such as Ethernet IEEE 802.3 at the data-link layer, the Internet Protocol (IP) at the network layer and TCP/UDP at the transport layer. In

these aspects CORBA takes care on beforehand of several of these requirements. However, for many service requirements of the proposed architecture it was necessary inquiring in more depth the CORBA capabilities to accomplish them. As an example, it was investigated the CORBA capabilities to deal with problems related to partial communication failures, communication for object groups, non-blocking and asynchronous communication and others.

One of the challenges of the CORBA approach for the proposed architecture inhabits in the fact that the system must accomplish the prerequisite of high availability. The architecture proposal must be designed taking in account that certain parts of the system, such as storage subsystems, processors, network, applications and others can, at any time, fail or become inoperative. The great challenge of this proposal work consists thus inquiring the difficulties of adopting CORBA as a central part of the architecture, since the gotten infrastructure needs to be a proper high availability set of services. In contrary case, it will not fulfill its primary objectives. We will investigate possible ways of implementing CORBA objects in such a way that they fulfill the necessary high availability requirements.

Finally, this work looks for the possibility of implementing the proposed system in clusters built with wide scale and low cost hardware and software elements available in the market, such as Intel-processor-based PCs and servers running open-source and low-cost operating systems such as Linux or FreeBSD and interconnected by low cost local area networks. Such platforms are popularly known as Beowulf Clusters. This aspect of the inquiry will have influence in the architecture proposal, once we are focusing aspects of high scalability for the cluster since there is a trend for the use of this type of cluster in applications of high demand of processing

and resources, such as web servers for great suppliers of content and services on the Internet.

ÍNDICE ANALÍTICO

RESUMO.....	IV
ABSTRACT	VI
ÍNDICE ANALÍTICO	IX
ÍNDICE DE FIGURAS.....	XIV
AGRADECIMENTOS	XIX
CAPÍTULO I INTRODUÇÃO.....	4
1 INTRODUÇÃO.....	4
1.1 PROBLEMA DE INVESTIGAÇÃO	4
1.2 OBJETIVOS.....	8
1.3 CONTRIBUIÇÕES.....	12
CAPÍTULO II <i>CLUSTERS</i>	15
2 <i>CLUSTERS</i>	15
2.1 O QUE SÃO <i>CLUSTERS</i>	15
2.2 TIPOS DE <i>CLUSTER</i>	16
2.2.1 <i>Clusters de Alta Disponibilidade</i>	17
2.2.2 <i>Clusters para Computação de Alto Desempenho</i>	18
2.2.3 <i>Clusters para Escalação Horizontal</i>	19
2.2.4 <i>Economia de clusters</i>	20
2.2.5 <i>Componentes de um cluster</i>	22
2.3 OS NÓS DE UM <i>CLUSTER</i>	23
2.3.1 <i>Funções de um nó do cluster</i>	23
2.3.1.1 Nó de Computação	24
2.3.1.2 Nó de Controle	24
2.3.1.3 Nó de Armazenamento.....	25
2.3.1.4 Nó de Instalação	25
2.3.1.5 Nó de Gerenciamento.....	26
2.3.1.6 Nó de Usuário.....	26
2.3.2 <i>Tipos de nó de um cluster</i>	27
2.3.2.1 Nó-Cabeça (Head Node).....	27
2.3.2.2 Nó de Computação	28
2.3.2.3 Nó de Armazenamento.....	28
2.4 REDES DE <i>CLUSTERS</i>	29
2.4.1 <i>Interfaces de Rede</i>	32
2.4.1.1 Adaptador de Serviço	32
2.4.1.2 Adaptador de Espera	33
2.4.1.3 Adaptador de Carga.....	33

2.4.2	<i>Tipos de Rede</i>	34
2.4.2.1	Nome da Rede	34
2.4.2.2	Atributos de Rede	34
2.4.3	<i>Tecnologias de Rede</i>	37
2.4.3.1	Ethernet e Fast Ethernet	37
2.4.3.2	FDDI	42
2.4.3.3	Gigabit Ethernet	47
2.4.3.4	Myrinet	48
2.4.4	<i>Redes com Chaveamento</i>	49
2.4.4.1	Chaves de Rede em Clusters	50
2.4.4.2	Operação de uma Chave de Rede	50
2.5	SUBSISTEMAS DE E/S	51
2.5.1	<i>O Barramento SCSI</i>	51
2.5.1.1	SCSI-1	52
2.5.1.2	SCSI-2	53
2.5.1.3	SCSI-3	54
2.5.2	<i>O Barramento PCI</i>	55
2.6	ARMAZENAMENTO	56
2.6.1	<i>Discos Compartilhados</i>	58
2.6.2	<i>Volumes de Disco</i>	59
2.6.2.1	Volume Físico	59
2.6.2.2	Volume Lógico	59
2.6.2.3	Volume Compartilhado	59
2.6.2.4	Volume de Grupo	60
2.6.2.5	Volume de Grupo Compartilhado	60
2.6.3	<i>Concorrência de Acesso a Discos</i>	60
2.6.4	<i>RAID</i>	61
2.6.4.1	Vantagens do RAID	61
2.6.4.2	RAID de Software vs. RAID de Hardware	62
2.6.4.3	Desempenho do RAID	64
2.6.4.4	RAID nível 0	65
2.6.4.5	RAID nível 1	66
2.6.4.6	RAID nível 0+1	68
2.6.4.7	RAID nível 5	69
2.7	E/S PARALELA	71
2.7.1	<i>O problema de E/S em clusters</i>	71
2.7.2	<i>Clusters e o desempenho de E/S</i>	72
2.7.3	<i>VSD</i>	73
2.7.3.1	Arquitetura do VSD	74
2.7.4	<i>DRBD</i>	76
2.7.4.1	Protocolos DRBD	77
2.7.4.2	Arquitetura do DRBD	79
2.8	SOFTWARE DE CLUSTER	80
2.8.1	<i>Sistemas de Arquivo</i>	80
2.8.2	<i>Sistemas de Arquivos Distribuídos</i>	80

2.8.2.1	NFS	81
2.8.2.2	AFS	84
2.8.2.3	Coda	85
2.8.3	<i>Sistemas de Arquivos Paralelos</i>	87
2.8.3.1	Arquitetura de um PFS	89
2.8.3.2	PVFS	90
2.8.3.3	GPFS	95
2.9	OUTROS COMPONENTES DE UM <i>CLUSTER</i>	100
2.9.1	<i>Clientes</i>	100
2.9.2	<i>Servidor de Terminal</i>	100
CAPÍTULO III ALTA DISPONIBILIDADE		102
3	ALTA DISPONIBILIDADE	102
3.1	NECESSIDADE DE ALTA DISPONIBILIDADE	103
3.2	DEFININDO ALTA DISPONIBILIDADE	103
3.2.1	<i>Sistemas Tolerantes a Falhas</i>	104
3.2.2	<i>Clusters de Alta Disponibilidade</i>	105
3.3	ARQUITETURAS DE ALTA DISPONIBILIDADE	106
3.3.1	<i>Clusters e Nós</i>	107
3.3.2	<i>Clusters e Serviços</i>	111
3.3.3	<i>Alocação de Serviços</i>	116
3.3.4	<i>Serviços e Recursos</i>	123
3.3.5	<i>Migração de Serviços</i>	129
3.3.6	<i>Votação de Propostas</i>	134
3.3.7	<i>Votação de Propostas no DSMCA</i>	136
CAPÍTULO IV DSMCA A DISTRIBUTED SYSTEM FOR SUPPORT AND MANAGEMENT OF CLUSTERS AND APPLICATIONS.....		146
4	DSMCA	146
4.1	DSMCA – VISÃO GERAL	150
4.1.1	<i>Aplicações Distribuídas em DSMCA</i>	151
4.1.2	<i>Comunicação Confiável em DSMCA</i>	152
4.1.3	<i>Monitoria de Estado de Nós em DSMCA</i>	154
4.1.4	<i>Conexão aos Serviços de Comunicação</i>	158
4.1.5	<i>Exemplo de Aplicação Distribuída</i>	160
4.1.6	<i>Inter-relação entre Monitoria de Nós e Grupos</i>	165
4.1.7	<i>Clientes de uma Aplicação Distribuída</i>	166
4.1.8	<i>Recursos e Serviços de Cluster</i>	167
4.1.8.1	Recursos de Cluster	168
4.1.8.2	Serviços de Cluster	171
4.1.8.3	Migração de Serviços no Cluster	176
4.1.8.4	Recursos e Eventos	179
4.1.8.5	Recursos e Ações	182

4.1.8.6	Monitoração Remota de Recursos	186
4.1.8.7	O Serviço de Dados do Cluster	189
4.2	DSMCA – VISÃO ESTRUTURAL	191
4.2.1	<i>Resource Managers</i>	191
4.2.2	<i>Tipagem e Identificação de Recursos</i>	192
4.2.3	<i>Estrutura de um Gerenciador de Recursos</i>	196
4.2.3.1	Construção da Árvore de Recursos	200
4.2.3.2	Administração de Recursos	204
4.2.3.3	Eventos de Recursos	207
4.2.4	<i>Módulos do Sistema DSMCA</i>	213
4.2.4.1	Localizador e Ativador de Recursos	213
4.2.5	<i>Seletor de Eventos</i>	218
4.2.6	<i>Seletor de Ações</i>	221
4.2.7	<i>Gerenciador de Recursos de Cluster</i>	224
4.2.8	<i>Executor de Ações</i>	228
4.2.9	<i>Gerenciador de Serviços</i>	230
4.2.10	<i>Gerenciador de Serviços de Cluster</i>	235
4.2.11	<i>Serviço de Grupos</i>	242
4.2.12	<i>Monitor de Adaptadores de Comunicação</i>	245
4.2.13	<i>Monitor de Nós</i>	246
4.3	ANÁLISE DE IMPLEMENTAÇÕES DE CORBA	248
CAPÍTULO V TRABALHOS RELACIONADOS		252
5	TRABALHOS RELACIONADOS	252
5.1	LVS	252
5.2	O PROJETO LINUX VIRTUAL SERVER	256
5.2.1	<i>Heartbeat</i>	256
5.2.2	<i>Fake</i>	262
5.2.3	<i>MON</i>	265
5.2.4	<i>O cluster HAL: LVS, Heartbeat, Fake e MON</i>	271
	<i>Bom desempenho</i>	275
	<i>Boa disponibilidade</i>	275
	<i>Boa Escalabilidade</i>	275
	<i>Boa Flexibilidade</i>	275
5.3	SGI LINUX FAILSAFE TM	278
5.3.1	<i>Visão Geral da Arquitetura</i>	280
5.3.2	<i>CMS – Cluster Membership Service</i>	281
5.3.3	<i>GCS – Group Communication Services</i>	282
5.3.4	<i>SRM – System Resource Manager</i>	282
5.3.5	<i>O Módulo LinuxFailSafe</i>	283
5.3.6	<i>Ferramentas Administrativas Linux FailSafe</i>	283
5.3.7	<i>Base de Dados de Configuração</i>	284
5.3.8	<i>Uma análise do SGI Linux FailSafe</i>	284
5.4	IBM HACMP TM E PSSP TM	286

5.4.1 IBM PSSP.....	287
5.4.2 IBM HACMP.....	288
5.4.3 TS - Topology Services	288
5.4.4 RML - Reliable Messages Library.....	289
5.4.5 GS - Group Services	290
5.4.6 EM – Event Management	292
5.4.7 Uma análise do IBM HACMP e PSSP/RSCT	293
5.5 COMPARAÇÕES COM O DSMCA.....	295
5.5.1 Comparação Arquitetural.....	296
5.5.1.1 A Plataforma de Desenvolvimento	296
5.5.1.2 Interfaces Para Desenvolvimento de Aplicações	298
5.5.1.3 Interoperabilidade Entre Sistema e Aplicações.....	301
5.5.1.4 Modelo de Comunicação Inter-Processos	303
5.5.2 Comparação Funcional.....	310
5.5.2.1 Acessibilidade aos Serviços.....	310
5.5.2.2 Serviços de Evento	314
5.5.2.3 Serviços de Ação	318
5.5.2.4 Gerenciamento de Recursos e Serviços.....	320
CAPÍTULO VI CONCLUSÕES E TRABALHOS FUTUROS.....	323
6 CONCLUSÕES E TRABALHOS FUTUROS.....	323
REFERÊNCIAS BIBLIOGRÁFICAS	332

ÍNDICE DE FIGURAS

<u>Número</u>	<u>Descrição</u>	<u>Página</u>
Figura 2-1	O projeto “Tertiary Disk” da Universidade de Berkeley	22
Figura 2-2	O <i>cluster</i> “Tertiary Disk” e sua implementação utilizando armários (<i>racks</i>) para organização dos nós.	22
Figura 2-3	Estrutura lógica de um <i>cluster</i> (reproduzido de [FERREIRA2002])	24
Figura 2-4	Um <i>cluster</i> e seus diversos tipos de nós (reproduzido de [FERREIRA2002])	29
Figura 2-5	Um <i>cluster</i> de servidores com armazenamento via Storage Area Network (SAN) (reproduzido de [IMEXSAN])	30
Figura 2-6	Uma topologia de conexão em estrela (<i>star</i>) (reproduzido de [CISCOETH])	39
Figura 2-7	As camadas do padrão <i>Ethernet IEEE 802.3</i> (reproduzido de [CISCOETH])	40
Figura 2-8	Compatibilidades necessárias entre as subcamadas MAC e PHY em uma rede IEEE 802.3 (reproduzido de [CISCOETH])	41
Figura 2-9	Uma chave <i>FastEthernet</i> da Cisco Systems (WS-C2924).	42
Figura 2-10	Um adaptador <i>FastEthernet</i> da 3Com.	42
Figura 2-11	Os dois anéis de uma rede FDDI (reproduzido de [CISCOFDDI])	43
Figura 2-12	Uma rede FDDI reconfigurada devido à falha de um dos nós (reproduzido de [CISCOFDDI])	45
Figura 2-13	Uma rede FDDI reconfigurada devido à falha de um dos enlaces de fibra (reproduzido de [CISCOFDDI])	45
Figura 2-14	Fibras <i>monomodo</i> e <i>multimodo</i> (reproduzido de [CISCOFDDI])	46
Figura 2-15	As quatro camadas que especificam o padrão FDDI	46
Figura 2-16	Chaves <i>GigaEthernet</i> da Cisco Systems (Catalyst 3550-12T and 3550-12G)	47
Figura 2-17	<i>Myrinet-2000-Fiber/PCI</i> interface card – um modelo de adaptador <i>Myrinet</i>	49
Figura 2-18	Uma chave de rede local (<i>LAN switch</i>) e a camada OSI onde ela atua	51
Figura 2-19	Um disco compartilhado pelos nós de um <i>cluster</i> através de barramentos SCSI (reproduzido de [CHIAKPO1997])	58
Figura 2-20	Controladora RAID da Intel, de 2 canais SCSI Ultra160	64
Figura 2-21	Armazenamento da sequência de dados ABCDEF... em um arranjo RAID de nível 0 composto por quatro dispositivos de disco	66

Figura 2-22 um subsistema de armazenamento composto de quatro arranjos RAID de nível 1	68
Figura 2-23 Um RAID de nível 0+1 composto de quatro discos. Ocorre <i>striping</i> e espelhamento simultâneo dos dados.	69
Figura 2-24 Um RAID 5 composto de 5 volumes de disco.	71
Figura 2-25 A arquitetura do VSD – Virtual Shared Disk da IBM	75
Figura 2-26 Arquitetura de um <i>cluster DRBD</i> de dois nós realizando espelhamento dos discos.	79
Figura 2-27 A nós de um sistema de arquivos PVFS (reproduzido de [PVFS])	92
Figura 2-28 Fluxo de dados em uma operação em metadados (reproduzido de [PVFS])	94
Figura 2-29 Fluxo de dados no acesso a um arquivo PVFS (reproduzido de [PVFS])	94
Figura 2-30 Acesso a arquivo PVFS através do <i>kernel</i> (reproduzido de [PVFS])	94
Figura 2-31 Arquitetura do GPFS (reproduzido de [GPFS])	97
Figura 3-1 Disponibilidade em função do <i>downtime</i> ao ano.	104
Figura 3-2 Cluster DSMCA típico	108
Figura 3-3 Dois <i>clusters</i> DSMCA compartilhando redes	110
Figura 3-4 Seis nós compondo três clusters com superposição.	111
Figura 3-5 Um serviço <i>single-instanced</i>	112
Figura 3-6 Serviços <i>multi-instanced</i>	113
Figura 3-7 Um <i>cluster</i> dotado de dois domínios de nós de serviço	115
Figura 3-8 Domínios de nós de serviço superpostos	116
Figura 3-9 Precedências dos nós para os serviços 1, 2 e 3	118
Figura 3-10 Nó não-operacional (2) em um domínio de serviço	119
Figura 3-11 Uma variação do caso anterior, em que o nó 1 não está operacional	120
Figura 3-12 Falha de um segundo nó concomitantemente à falha do primeiro.	121
Figura 3-13 Recursos típicos de ambientes de <i>cluster</i>	125
Figura 3-14 Distribuição dos recursos do <i>cluster</i> aos serviços.	127
Figura 3-15 Relação de dependência entre os recursos de um serviço	128
Figura 3-16 Falha de um nó executor de serviço	131
Figura 3-17 O processo de votação sobre o destino do serviço 3, cujo nó falhou (Figura 3-16)	132
Figura 3-18 O <i>cluster</i> após a eleição do nó 3.	134
Figura 3-19 Envio da proposta por um membro do grupo	135
Figura 3-20 Solicitação de apreciação da proposta	135
Figura 3-21 Envio do voto por cada membro do grupo	137
Figura 3-22 Comunicação do resultado	137
Figura 3-23 Proposta feita pelo nó #4, contendo 2 itens, em 2 fases	138
Figura 3-24 Recebimento do item 1 da proposta	139
Figura 3-25 Votação da proposta de dois itens	141

Figura 4-1 Arquitetura do DSMCA	149
Figura 4-2 Modelo de interconexão bidirecional de objetos CORBA utilizado por DSMCA	151
Figura 4-3 Comunicação entre nós via Cluster Message Service utilizando nós encaminhadores intermediários em redes particionadas.	155
Figura 4-4 Um <i>heartbeat ring</i> de adaptadores composto de cinco nós	157
Figura 4-5 A infra-estrutura de serviços de comunicação do sistema DSMCA	159
Figura 4-6 Grupos de objetos coordenados pelo Serviço de Comunicação em Grupo de DSMCA, compondo possíveis aplicações distribuídas.	161
Figura 4-7 Um servidor de banco de dados distribuído operando em três nós com o suporte dos serviços de DSMCA	163
Figura 4-8 Os recursos e gerenciadores de recurso da aplicação Servidor de Banco de Dados Distribuído	170
Figura 4-9 Dois serviços e seus respectivos recursos em execução em um nó de <i>cluster</i> , sob gerenciamento do <i>Service Manager</i>	172
Figura 4-10 Dois serviços compartilhando o recurso Adaptador de Rede em um nó do <i>cluster</i>	174
Figura 4-11 A gerência global de serviços de <i>cluster</i> do serviço <i>Cluster Manager</i> em um <i>cluster</i> DSMCA com cinco serviços ativos.	177
Figura 4-12 Recursos encaminham seus eventos ao "hub", que por sua vez os encaminha ao serviço de eventos de DSMCA	181
Figura 4-13 Os objetos e serviços relacionados a Ações de Recurso e seus relacionamentos com o serviço de Eventos em DSMCA	184
Figura 4-14 O serviço de monitoração remota de recursos de DSMCA	188
Figura 4-15 O serviço de dados de DSMCA (<i>Cluster DB Manager</i>) e seus clientes no sistema.	190
Figura 4-16 Níveis de um ResourceInfo	193
Figura 4-17 Estrutura interna primária de um gerenciador de recurso	199
Figura 4-18 O primeiro e o segundo passo na criação de um ramo da árvore administrativa de recursos em um gerenciador	201
Figura 4-19 Dois ramos administrativos são criados a partir da administração-topo da árvore do gerenciador	202
Figura 4-20 As visões estrutural e administrativa da árvore do gerenciador	203
Figura 4-21 Os objetos de administração e gerenciamento de recursos nos primeiros níveis do gerenciador MGR1	205
Figura 4-22 O dispositivo Dev1 gerenciado pelo gerenciador MGR1 e as áreas e classes gerenciais ativas	206
Figura 4-23 Encaminhadores de Evento nos níveis mais altos de um gerenciador de recurso	208
Figura 4-24 Os objetos de encaminhamento de eventos dos níveis mais baixos do gerenciador. Podemos ver aqui os encaminhadores do dispositivo Dev1, de suas áreas, classes e objetos de quantidade.	210

Figura 4-25 A estrutura em árvore de encaminhadores de evento e a presença de geradores de evento em qualquer nível.....	212
Figura 4-26 Os encaminhadores de evento na raiz dos gerenciadores enviam seus eventos para o <i>hub</i> . Processadores locais de evento podem estar conectados em qualquer ponto.	214
Figura 4-27 Localizador e Ativador de Recursos de DSMCA	217
Figura 4-28 Seletor de Eventos do DSMCA	220
Figura 4-29 Seletor de Ações do DSMCA	223
Figura 4-30 Gerenciador de Recursos de Cluster de DSMCA	227
Figura 4-31 Executor de Ações do DSMCA	229
Figura 4-32 Gerenciador de Serviços locais de DSMCA	234
Figura 4-33 Gerenciador de Serviços de Cluster de DSMCA	241
Figura 4-34 Monitoria de Adaptadores, Nós e Grupos	247
Figura 5-1 O sistema LVS compondo um pequeno <i>cluster</i> de quatro nós, sendo três deles nós servidores e um deles realizando balanceamento da carga de trabalho.....	254
Figura 5-2 Um <i>cluster</i> de dois nós utilizando <i>heartbeat</i> como monitor de estado operacional de seus nós.	258
Figura 5-3 O mesmo <i>cluster</i> após a falha do nó Servidor 1	259
Figura 5-4 Um pequeno <i>cluster</i> de dois nós utilizando <i>Fake</i> para o <i>takeover</i> de endereços IP, em seu estado normal.....	264
Figura 5-5 O estado dos serviços e recursos do <i>cluster</i> após falha do nó <i>Servidor 1</i> e o <i>takeover</i> de recursos, inclusive do endereço IP	265
Figura 5-6 Cluster executando quatro serviços monitorados por MON.....	269
Figura 5-7 Falha do serviço HTTP é detectada por MON no nó Servidor 2.....	269
Figura 5-8 O serviço HTTP e o recurso IP 200.190.10.3 são finalmente tomados pelo nó Servidor 2	271
Figura 5-9 Um <i>cluster</i> de alta disponibilidade com arquitetura HAL	272
Figura 5-10 Um <i>cluster</i> HAL dotado de redundância no balanceador de carga	274
Figura 5-11 Um cluster SGI Linux FailSafe [FAILSAFE] típico.	280
Figura 5-12 Arquitetura do sistema Linux FailSafe.....	280
Figura 5-13 A infra-estrutura de suporte para HA da IBM em seus clusters (reproduzido de [CHIAKPO1997]).....	287
Figura 5-14 Os serviços de topologia (TS), de Grupos (GS) e a biblioteca de mensagens confiáveis em um <i>cluster</i> RSCT	290
Figura 5-15 O serviço de grupos (GS) de PSSP/RSCT (reproduzido de [CHIAKPO1996]).....	292
Figura 5-16 Objeto-cliente solicita <i>proxy</i> para conexão ao serviço DSMCA	312
Figura 5-17 Objeto-cliente conecta-se ao <i>proxy</i> de serviço e passa a fazer solicitações.	312
Figura 5-18 Um segundo objeto-cliente faz também solicitação por um <i>proxy</i>	313
Figura 5-19 O segundo objeto-cliente conecta-se ao <i>proxy</i> do serviço.	313

AGRADECIMENTOS

Ao professor Dr. Gonzalo Travieso, meu orientador neste trabalho, pelas muitas discussões, revisões e por sua infinita paciência.

Ao professor Dr. Jan Frans Willen Slaets, co-orientador, pelas muitas sugestões e apoio.

Ao Instituto de Física de São Carlos - USP, pela colaboração e suporte na realização deste trabalho.

À minha esposa Ana Lucia, pela paciência, compreensão, apoio, amizade e carinho nos momentos mais difíceis.

Ao meu filho Lucas, pelo seu amor, companhia e por ser tão adorável.

Aos amigos Maria Alice e Elias Raimundo, pela amizade e pelas horas tão agradáveis passadas juntos.

Ao meu sobrinho Paulo Ricardo, pelo seu esforço, dedicação, inteligência e por ser tão prestativo.

À reitoria da Universidade de Franca, que me concedeu tempo e suporte para o estudo e elaboração deste trabalho.

Ao professor Dr. Ângelo Colombini, pela compreensão e atendimento às minhas solicitações e pedidos.

Ao Dr. Ronald Ludovice, pelo suporte e apoio nas horas mais complicadas.

Ao Fernando Ewald, pela enorme colaboração, interesse e capacidade de solucionar problemas.

Aos professores Ruy Tadashi e Regiane Máximo, pelos conselhos, direcionamentos, pela amizade e pelos inesquecíveis jantares.

À professora Regina Fumie Eto, pelas ótimas conversas, pela sua amizade e por sua grande dedicação naquilo que faz.

À professora Janine D'Aquino, pelo seu incrível bom humor, incentivo e pelas fabulosas conversas.

Ao professor Maurizio Manzo, pela sua grande amizade, ótima companhia e pelo seu enorme talento.

Ao professor Décio Lago, pela amizade, companhia, pelas infindáveis discussões e por sua sensibilidade aos assuntos desportivos.

Ao professor Marcelo Ruy, pela amizade, companhia, pelas conversas descontraídas e pelo seu vocabulário fantástico.

Aos amigos Jair Marcatti e Lucilla Abdala, pelo apoio, pela compreensão, pelos ótimos momentos juntos e por se amarem tanto.

Aos amigos Gustavo, Marília e Sofia, pela amizade incomparável e pelos ótimos momentos juntos.

A todos aqueles que amo e me amam, sem os quais não teria tido incentivo nem encontrado ânimo e coragem para elaborar e concluir este trabalho.

A todas as pessoas que, direta ou indiretamente, contribuíram e me apoiaram na confecção deste trabalho.

INTRODUÇÃO

1 Introdução

1.1 Problema de Investigação

O arranjo de computadores em *cluster* tem por objetivo estender as capacidades dos computadores convencionais [STE2002]. O termo *cluster* pode ser traduzido como o ato de agrupar, de criação de um grupo de trabalho conjunto, e assim o conceito de *cluster* de computadores tem a ver com as técnicas de se agrupar computadores a fim de que estes realizem trabalho em grupo visando uma melhora no desempenho da realização de suas tarefas. Na natureza, *clustering* é um conceito e um mecanismo fundamental na criação de complexidade e diversidade através da agregação e síntese de elementos básicos e simples. O resultado é nada mais nada menos que a evolução e a estruturação do universo, de suas moléculas componentes que ditam o aspecto e a distribuição de toda a matéria e o comportamento de toda vida multicelular, incluindo o homem. Alguns dos maiores computadores do mundo hoje em dia são sistemas em *cluster*, porém estes estão tendo também um papel extremamente

importante na computação técnica, científica e comercial em média escala, tirando vantagem do baixo custo do mercado de massa dos computadores baseados na tecnologia de PCs – *Personal Computers*. Estes sistemas, categorizados como *clusters* da classe *Beowulf* [STE1995] tornaram-se extremamente populares por apresentar uma excepcional relação preço/desempenho, flexibilidade de configuração e atualização e também por prover grande escalabilidade, abrindo assim um grande leque de oportunidades para aplicações computacionais. Muitas dessas vantagens são uma consequência da superior relação preço/desempenho que estes apresentam em relação a muitos outros sistemas de capacidade comparável. Mas outros atributos importantes apresentados pelos *clusters* são devidos à natureza de sua estrutura e do seu método de implementação. Dentre eles, podemos destacar: facilidade de escalar graus mais elevados de desempenho e capacidade, flexibilidade de configuração e atualização, convergência de arquitetura, rápido acompanhamento tecnológico, maior capacidade e poder computacional para o usuário comum e possibilidade de apresentar alta disponibilidade de recursos e serviços. De um modo geral, pode-se categorizar os *clusters* de acordo com o objetivo de utilização ou o tipo de aplicação [FERREIRA2002]. Existem sistemas *cluster* preparados para rodar aplicações que demandam elevado poder computacional e que são denominados *clusters* para computação de alto desempenho, ou HPCC (*High Performance Computing Clusters*). Existem também sistemas *cluster* preparados para oferecer alta disponibilidade (HA¹) de recursos, os quais são usualmente empregados para rodar aplicações servidoras e são vistos como servidores de elevada capacidade, poder e disponibilidade. Esses sistemas são denominados *clusters* de alta disponibilidade, ou HAC (*High Availability Clusters*). É possível ter essas duas características combinadas em um único *cluster*, configurando um sistema que apresenta capacidade de

¹ HA, sigla oriunda do termo inglês *High Availability*

processamento elevada e alta disponibilidade de recursos. Essa possibilidade realça a flexibilidade que tais arquiteturas de sistema intrinsecamente apresentam. Também, um grande atrativo de tais arquiteturas é a sua notável escalabilidade, o que significa que as capacidades podem ser expandidas e aproveitadas para se obter maior desempenho ao agregar-se ao *cluster* novos elementos e recursos, tais como mais nós processadores, memória e armazenamento. Como exemplo de uma aplicação típica de *clusters* de servidores de alto desempenho e alta disponibilidade, podemos citar [DIAS96].

A característica de alta disponibilidade dos *clusters* é parte central deste trabalho. O conceito de alta disponibilidade tem a ver com a continuidade operacional de um determinado recurso computacional, seja ele um componente de *hardware* ou *software*, o qual é um provedor de serviço básico. Este, por sua vez, é utilizado por aplicações e estas por sua vez podem ser provedoras de serviços mais elaborados, de mais alto nível. Os sistemas em *cluster* podem ser arquitetados para prover serviços para usuários internos a ele, para usuários externos ou para ambas as classes. Em qualquer caso, a disponibilidade de um recurso do sistema pode ter relação direta com a disponibilidade de um serviço oferecido pelo *cluster*. Se um determinado serviço é orientado para usuários externos ao *cluster*, tais como computadores-cliente que vêem o *cluster* como um provedor de serviços, então a disponibilidade do serviço estará correlacionada à disponibilidade dos componentes constituintes do serviço. Prover o *cluster* com características de alta disponibilidade significa garantir uma alta disponibilidade dos recursos básicos de que se utilizam as aplicações do *cluster*. Como exemplo, se uma aplicação necessita de um certo sistema de arquivos para operar, a disponibilidade do serviço daquela aplicação está correlacionada com a disponibilidade daquele sistema de arquivos. Tornar essa aplicação altamente disponível aos seus usuários significa aumentar, entre outras coisas, a disponibilidade desse sistema de arquivos. As

arquiteturas de sistema em *cluster* são especialmente adequadas para prover serviços de alta disponibilidade [YOUN2000]. Com um projeto adequado, elas são capazes de prover redundância de subsistemas e em muitos casos eliminar a presença de pontos únicos de falha, os denominados SPOFs (*Single Points of Failure*), sejam eles relativos ao *hardware*, *software* ou à rede de comunicação. Assim, uma falha em um componente não rompe a disponibilidade do serviço porque um outro dá continuidade ou assume os seus serviços. Essa capacidade pode ser conseguida de diversos modos, sejam elas técnicas associadas a recursos dotados de tolerância a falhas (*fault-tolerance*) onde se consegue disponibilidade ininterrupta do serviço ou em técnicas baseadas na atribuição do serviço ou recurso que apresentou falha a um elemento de reserva (*backup*) sob coordenação da gerência de recursos do *cluster*. Essa última técnica é denominada *failover*. Recursos dotados de tolerância a falha são geralmente componentes específicos e de custo elevado [BAKER1999], enquanto que as técnicas de *failover* podem ser aplicadas a componentes comuns, de baixo custo. Algumas configurações de *cluster* empregam ambas as técnicas, havendo certos componentes específicos dotados de tolerância à falha, garantindo disponibilidade contínua dos mesmos, e outros que têm sua disponibilidade aumentada através de técnicas de *failover*. Muitos integradores e desenvolvedores de *software* e *hardware* para *clusters* oferecem esta facilidade, obtendo-se assim um nível de disponibilidade de serviço adequado às aplicações do mesmo [SHERMAN1998].

Neste trabalho, vamos enfocar as técnicas de alta disponibilidade de serviços de *cluster* baseadas em *failover*. Em particular, essas técnicas prescindem de um forte controle e monitoração dos recursos que possibilite detectar com rapidez falhas no estado operacional dos mesmos. Essa atividade é realizada por certas áreas gerenciais do *cluster*, as quais realizam uma monitoração constante da disponibilidade dos elementos

críticos a fim de que ações de *failover* sejam executadas o mais rapidamente possível, de modo a se conseguir um aumento da continuidade ou do nível de disponibilidade dos serviços do *cluster*. O nível de disponibilidade dos serviços é medido como um percentual do tempo em que o sistema deveria oferece-los aos seus usuários. Se, por exemplo, os serviços de um sistema apresentam-se 99% do tempo disponíveis aos usuários, então temos uma continuidade ou nível de disponibilidade dos mesmos da ordem de 99% [PFISTER1998], [BEYER1995]. Existem várias abordagens para se atingir níveis elevados de disponibilidade dos serviços de um *cluster*, e muitas iniciativas já existem para se conseguir implementá-las e melhorá-las. Algumas delas já surgem como produtos maduros, proprietários e comerciais. Outras são iniciativas de grupos que aceitam colaboradores para auxiliar no seu desenvolvimento e oferecem código aberto e gratuito. As capacidades e o enfoque das diversas soluções e iniciativas existentes são relatadas em [YOUN2000].

1.2 Objetivos

A fim de se conseguir operar e utilizar um *cluster*, são necessários ambientes operacionais, gerenciais e de desenvolvimento específicos. Em alguns casos, sistemas operacionais específicos são utilizados nos nós componentes do sistema. Em outros casos, sistemas operacionais convencionais são empregados juntamente com subsistemas de gerenciamento, operacionalização e suporte ao ambiente multiprocessado e de memória distribuída que se obtém do arranjo. Em ambos os casos, os seguintes suportes e facilidades surgem ou se fazem necessários: suporte à comunicação e rede, suporte à interoperabilidade (*middleware*), utilitários administrativos (instalação, configuração, segurança, controle), armazenamento distribuído (discos e sistemas de arquivo), imagem única do sistema (SSI), suporte a aplicações distribuídas e/ou paralelas

(bibliotecas de passagem de mensagens, compiladores, ambientes de desenvolvimento), monitoração, controle e distribuição de recursos e finalmente suporte e facilidades relacionadas a alta disponibilidade [BAKER2000].

Neste trabalho temos como objetivo investigar a possibilidade de se implementar uma arquitetura de gerenciamento e de suporte a aplicações distribuídas de alta disponibilidade em *clusters* construídas a partir dos recursos da arquitetura de objetos distribuídos CORBA [OMG], [VINOSKI1996], [SCHMIDT1995]. Essa arquitetura deve ser capaz de prover facilidades ao administrador e ao desenvolvedor de aplicações distribuídas do *cluster*, em particular ela deve oferecer facilidades para melhorar a disponibilidade do serviço das aplicações. Existem diversas plataformas para o desenvolvimento de aplicações para *clusters*, algumas são orientadas à construção de aplicações de alto desempenho distribuídas, tais como os padrões de passagem de mensagem MPI [MPI] e PVM [PVM]. Outras são orientadas à construção de aplicações distribuídas que compõem serviços, tais como servidores de banco de dados, servidores de arquivo de alto desempenho e distribuídos, servidores para aplicações Web/Internet e outros. Dessa última categoria, podemos citar como exemplo as plataformas PSSP/RSCT [CHIAKPO1996] e HACMP/ES [CHIAKPO1997] da IBM, o *Microsoft Cluster Services* [MSCS], [SHORT98] da Microsoft e o *SGI Linux Failsafe* [FAILSAFE] da Silicon Graphics.

Desenvolver aplicações para sistemas distribuídos é uma tarefa complexa e difícil. O programador deve lidar com uma variedade de problemas que em princípio não surgem em sistemas *stand-alone* e, portanto ele deve empregar uma sofisticação muito maior no seu desenvolvimento além de meras habilidades de programação em rede. Dentre os principais problemas que surgem, podemos citar, dentre outros, os relacionados à

heterogeneidade, concorrência, latência e múltiplos modos de falha devido à possibilidade de ocorrência de falhas parciais. Sem o suporte de uma infra-estrutura de comunicação de alto nível, ou *middleware*, que pode em princípio ao menos isolar parcialmente o programador desses problemas, é extremamente difícil construir aplicações distribuídas complexas e confiáveis. A arquitetura OMG CORBA provê uma infra-estrutura orientada a objetos e um ORB – *Object Request Broker*, os quais permitem que objetos comuniquem-se independentemente do mecanismo de transporte, das plataformas e também das técnicas utilizadas para se implementar esses objetos. No topo dessa infra-estrutura o padrão provê, além disso, uma grande variedade de componentes reusáveis de serviço, indo desde componentes de serviço fundamentais, independentes do domínio da aplicação, até componentes de facilidades especializadas verticais, altamente focados em certos domínios de aplicação. Sendo assim, OMG CORBA tem como principais alvos: a heterogeneidade de sistemas distribuídos, redução do esforço de desenvolvimento e de erros de programação. Ao automatizar muitas das tarefas necessárias para a especificação e implementação de grande parte dos sistemas de aplicações distribuídas, CORBA melhora também os aspectos de flexibilidade e portabilidade.

De acordo com nossa proposta, a arquitetura de suporte e gerenciamento em questão deve satisfazer certos requisitos básicos, tais como lidar com condições de falha de elementos internos ao *cluster*, sejam eles nós processadores, de armazenamento, programas e aplicações, recursos, elementos da rede, etc. Sendo assim, essa arquitetura de suporte deve ser composta por um conjunto de programas, gerências e serviços capazes de lidar com essas possíveis falhas. Além disso, ela deve ser capaz de informar eventos, executar ações gerenciais, oferecer mecanismos de coordenação, suporte a grupos de processos, sincronização, consenso e votação, além de prover mecanismos de comunicação confiável. Deve

portanto prover subsídios para que as áreas gerenciais ou as próprias aplicações do *cluster* possam resolver problemas relacionados à falhas através das técnicas já citadas em 1.1 . Essa arquitetura de suporte e gerenciamento recebeu o nome de DSMCA – *Distributed Supporte and Management for Cluster Applications*, e suas características começam a ser apresentadas a partir do tópico 3.3 .

Estabelecemos assim que um dos requisitos básicos dessa infra-estrutura de suporte é que ela deve, em primeiro lugar, ser um elemento altamente disponível. Entretanto, as especificações de CORBA não tratam diretamente sobre as capacidades da arquitetura em oferecer suporte à construção de aplicações de alta disponibilidade, não oferecendo meios diretos de lidar com falhas em elementos fora da camada de *middleware*, tais como falhas em recursos gerenciados pelo sistema operacional. Como exemplo, CORBA e os serviços de transporte do seu ORB, tais como o IIOP, não lidam com falhas parciais na rede de comunicação. Felizmente, algumas implementações ORB CORBA lançam exceções à aplicação de modo que esta possa tentar tratá-las. Criar uma infra-estrutura de alta disponibilidade utilizando CORBA significa projetá-la levando em conta que a ocorrência de tais tipos de falha não deve, até onde for possível, interromper o serviço. A infra-estrutura deve analisar e tentar recuperar-se das possíveis falhas. Essas ações de recuperação farão parte do rol de atribuições das aplicações que compõem a infra-estrutura de serviços de suporte ao *cluster* de modo a oferecer um serviço de alta disponibilidade.

Um dos principais desafios deste trabalho é investigar as atuais implementações de CORBA e como elas lidam com essas questões, a fim de se determinar como CORBA poderia operar em tal ambiente e ainda assim poder prover serviços de alta disponibilidade. Sabe-se de antemão que as especificações recentes do OMG para CORBA incluem serviços que visam tornar a implementação de um ORB CORBA tolerante a falhas

[[FT-CORBA](#)]. Porém, como veremos adiante, o enfoque dessa abordagem é tratar de possíveis falhas internas ao *middleware*, tais como a falha de uma *thread* que executa um *servant* (a implementação de um objeto). Nessa abordagem, réplicas de objetos logicamente idênticos são utilizadas concorrentemente de modo a se alcançar algum nível de tolerância a falhas internamente ao ORB.

1.3 Contribuições

Existem diversas implementações de infra-estruturas de suporte e gerenciamento para *clusters*, muitas delas comerciais e proprietárias [[MSCS](#)], [[CHIAKPO1997](#)]. Algumas outras são resultados de esforços de grupos envolvidos no desenvolvimento dessas plataformas para ambientes de código aberto, como os esforços do grupo “High Availability Linux” [[HAL](#)] e do “Linux Virtual Server” [[LVS](#)]. Os produtos comerciais citados no primeiro grupo em geral são capazes de suportar *clusters* com grande número de nós. Como exemplo, os serviços de suporte do PSSP/RSCT da IBM são projetados para lidar com aplicações nas plataforma SP/2 [[IBMSPP2](#)] e em sua linha IBM eServer *Clusters* [[IBMCLUSTER](#)], que pode operar em configuração *cluster* de até centenas de nós. Em contrapartida, uma constante no segundo grupo é o fato de que as implementações atuais visam satisfazer as necessidades de alta disponibilidade de pequenos *clusters*, em particular no caso especial de *clusters* compostos de apenas dois nós. Como premissa desta investigação, vamos procurar especificar uma arquitetura aberta e apta a suportar *clusters* com um número maior de nós, com vistas a atender uma gama maior de aplicações e grande escalabilidade.

A escolha de CORBA como plataforma de *middleware* para a infra-estrutura de suporte pode ser justificada pelo fato de que as especificações do OMG para CORBA já contemplam diversos tipos de serviços, tanto

serviços específicos, verticais, quanto serviços gerais, horizontais, os quais podem ser empregados na especificação de nossa arquitetura. Podemos dizer de antemão que alguns deles podem ser empregados diretamente, tais como os serviços de Nome [[ORBNaming](#)] e de Propriedades [[ORBProperties](#)]. Outros foram investigados e suas funcionalidades serviram de base para a especificação de serviços específicos da arquitetura, uma vez que se detectou que certos serviços necessários eram muito específicos ao sistema. Como exemplo destes, temos serviços de Eventos [[ORBEvent](#)] e de Notificação [[ORBNotify](#)], os quais serão brevemente explanados e que tiveram algumas de suas metodologias empregadas. Também, as especificações de CORBA tolerante à falhas [[FTCORBA](#)] foram de grande valor para auxiliar na especificação de nossa arquitetura.

Alguns trabalhos anteriores fizeram investigações e propostas para a utilização de CORBA em alguns serviços que têm relação direta ou indireta com nossos requisitos. Como contribuição, procuramos determinar a viabilidade e as possíveis formas de agregação das técnicas apresentadas nesses trabalhos em nossa arquitetura. Como exemplo, os trabalhos de Stefen Havenstein [[HAVENS1999](#)] sobre serviços e sistemas transacionais em grupos de objetos CORBA, os trabalhos de Dukyun Nam et al. [[NAM2000](#)] sobre o suporte de CORBA para a comunicação em grupo, os trabalhos de Rodrigues et al. [[RODRIGUES1993](#)] sobre protocolos de *grupo membership* em redes LAN, os trabalhos de Idit Keidar et al. [[KEIDAR2000](#)] sobre serviços de *group membership* em WANs. Uma contribuição importante vem do trabalho realizado no projeto DOORS [[DOORS](#)], uma abordagem de tolerância a falhas para CORBA que introduziu algumas técnicas e conceitos novos e que serviu de base para as especificações do FT-CORBA. Também, faremos referência ao projeto e especificações de produtos de *cluster* de alta disponibilidade desenvolvidos por alguns fabricantes, tais como IBM,

Microsoft, Sun Microsystems e Silicon Graphics, os quais trazem contribuições valiosas, ainda que estes produtos não tenham utilizado plataformas de objetos distribuídos em seu desenvolvimento.

Capítulo II

CLUSTERS

2 *Clusters*

2.1 O que são *clusters*

O termo *cluster* tornou-se uma das palavras mais usadas na indústria de TI. Contudo, é também um dos termos mais mal entendidos ou ao menos mais mal empregados. Uma das razões para isto é que há muitos tipos de *clusters* e os *clusters* podem ser usados para muitas finalidades diferentes. Em seu formato mais simples, um *cluster* compõe-se de dois ou mais computadores que trabalham em conjunto a fim de atingir determinados objetivos de desempenho, disponibilidade e flexibilidade. Isto não deve ser confundido com o modelo cliente-servidor mais comum, onde uma aplicação pode ser logicamente dividida de modo tal que um ou mais clientes podem requisitar serviços de um ou mais servidores. A idéia por trás dos *clusters* é a de juntar os poderes computacionais dos nós envolvidos para fornecer maior escalabilidade, poder computacional mais elevado ou fornecer uma disponibilidade de serviços mais elevada. Assim, indo além do ambiente convencional, onde clientes simples fazem pedidos a um ou mais servidores, *clusters* utilizam múltiplas máquinas para

fornecer um ambiente computacional mais poderoso com uma imagem única do sistema (SSI).

Os computadores de um *cluster* devem de alguma forma estar cientes de sua condição, isto é, o trabalho que está sendo realizado em um nó específico deve ser coordenado com o trabalho que está sendo feito em outros nós. Isto pode resultar em configurações complexas de conectividade e no uso de sofisticados mecanismos de comunicações interprocessos entre os nós do *cluster*. Além disso, compartilhar dados entre os nós de um *cluster* através de um sistema de arquivos comum é quase sempre uma exigência. Há muitas outras complexidades que são introduzidas por *clusters*, tais como as considerações operacionais de tratar de um número potencialmente grande de computadores como um único recurso. Discutiremos estas considerações e demais outras no decorrer deste capítulo. Por ora, vamos descrever os tipos mais comuns de *clusters*.

2.2 Tipos de *cluster*

Conforme mencionado anteriormente, *clusters* de computadores existem em muitas formas diferentes. Os três tipos mais comuns de *cluster* são os que tem como objetivo ou meta oferecer:

- Disponibilidade Elevada (HA – *High Availability*)
- Desempenho de Computação Elevado (HPC – *High Performance Computing*)
- Escalação Horizontal (HS - *Horizontal Scaling*).

Deve-se notar que os limites entre estes tipos do *cluster* não são bem definidos e freqüentemente um *cluster* real pode ter propriedades ou fornecer a função de mais de um destes tipos do *cluster*.

2.2.1 *Clusters* de Alta Disponibilidade

Clusters de alta disponibilidade são construídos tipicamente com a intenção de fornecer um ambiente a prova de falhas através de redundância de recursos. Isto é, eles fornecem um ambiente computacional onde a falha de um ou mais componente (*hardware*, *software*, ou *networking*) não afeta significativamente a disponibilidade da aplicação ou das aplicações que estão sendo executadas ou disponibilizadas. No caso mais simples, um *cluster* de alta disponibilidade pode ser obtido pela configuração em modo idêntico de dois computadores que tenham acesso compartilhado aos dispositivos de armazenamento. Durante a operação normal, o ambiente da aplicação executa em um dos sistemas, enquanto que o outro sistema permanece simplesmente em espera, pronto para rodar a aplicação no caso de uma falha. Quando uma falha ocorre no primeiro, o segundo sistema assume, configura e instala os recursos apropriados (armazenamento, endereços de rede, serviços, etc.). Este processo é chamado tipicamente de *failover*. O segundo sistema então substitui completamente o sistema que falhou e os usuários finais, na outra extremidade do sistema, não têm nenhuma necessidade de saber que suas aplicações estão agora funcionando em uma máquina física diferente. Como já é possível antever, *clusters* reais de HA podem ter projetos complexos. Compreender todos os possíveis tipos de falha e trabalhar a redundância e os processos requeridos a fim de fornecer um ambiente de *failover* eficiente, suave e transparente faz parte das habilidades de um arquiteto de HA. Algumas das considerações que este arquiteto deve levar em conta incluem:

- manter um ambiente consistente entre o nó primário e o nó de espera,
- reconhecer quando uma falha ocorreu,

- implementar o processo de *failover* de modo que todos os recursos requeridos ficarão disponíveis ao nó de espera,
- prover integridade de dados e de transações no caso de uma falha

Há outras considerações a serem levadas em conta também, mas fica claro por esta curta lista que a configuração de *hardware*, *software* e rede de múltiplos nós, adequada a lidar virtualmente com todos os tipos de falha, podem requerer uma quantidade significativa de planejamento técnico. Além das questões técnicas, há também outras considerações relacionadas a negócio, tais como o custo dos componentes redundantes. Estes custos têm que ser computados levando-se em conta o custo de *downtime* das aplicação. Ou, de uma outra maneira, estimando-se o valor de assegurar que a aplicação estará sempre disponível.

2.2.2 Clusters para Computação de Alto

Desempenho

Clusters para computação de desempenho elevado (HPC) são tipicamente projetados para uso em computação paralela (HPPC) a fim de dedicar mais poder de processadores à solução de um problema. Há muitos exemplos na computação científica, onde o uso de múltiplos processadores de baixo custo em paralelo para executar um grande número operações concorrentes pode resultar em uma relação custo/benefício excelente ao fornecer quantidades maciças de poder computacional. Isto é o que se costuma chamar de computação paralela ou paralelismo. Thomas Sterling, em seu artigo intitulado *Como construir um Beowulf* [STER1995], cita que “paralelismo é a habilidade de muitas linhas de execução independentes fazer progresso simultaneamente para a conclusão de uma tarefa”. Os *clusters* de desempenho elevado são compostos tipicamente de um grande número nós. *Clusters* de 128 ou até

mesmo 1024 nós não são incomuns. Criar uma arquitetura para este tipo de *cluster* traz seus próprios desafios, incluindo:

- como instalar e manter o ambiente operacional e de aplicação em todos os nós
- como controlar proativamente estes nós através do envio de comandos, bem como manipulação de falhas.
- as exigências para acesso concorrente, paralelo e de alto desempenho aos mesmos sistema de arquivos;
- comunicação interprocessos entre os nós a fim de coordenar o trabalho que deve ser feito em paralelo

O objetivo é fornecer uma imagem única do sistema (SSI) [BAKER2000] controlando, operando, e coordenando um grande número de computadores discretos. Neste ambiente, um usuário freqüentemente interage com um nó específico para iniciar ou programar um trabalho a ser rodado. A aplicação, conjuntamente com várias outras funções gerenciais dentro do *cluster*, determina então como este trabalho será distribuído pelos nós do *cluster* a fim de tirar vantagem dos recursos disponíveis para produzir o resultado desejado.

2.2.3 Clusters para Escalação Horizontal

Os *clusters* projetados para escalação horizontal (HS - *Horizontal Scaling*) são usados para fornecer uma única interface a um conjunto de recursos que podem arbitrariamente crescer (ou encolher) em tamanho ao longo do tempo. O exemplo mais comum deste tipo de *cluster* é uma fazenda de servidores *web* (*server farm*). Neste exemplo, uma única interface (URL ou endereço IP e porta TCP) é fornecida, mas requisições vindas através desta interface podem ser alocadas ao longo de um grande conjunto de

servidores que fornecem uma capacidade mais elevada e a habilidade de controlar a experiência do usuário final através de funções tais como balanceamento de carga. Naturalmente este tipo de *cluster* fornece também redundância significativa. Se um servidor de uma grande fazenda falhar, esse evento será provavelmente transparente aos usuários. Conseqüentemente, este modelo tem muito dos atributos de um *cluster* de elevada disponibilidade. Do mesmo modo, uma vez que compartilha carga de trabalho entre muitos nós, realiza também uma forma de computação de alta performance. Na Figura 2-1 e na Figura 2-2 podemos observar o protótipo de um *cluster* que visa atender requisitos de elevada quantidade de recursos e alta disponibilidade. Trata-se do protótipo *Tertiary Disk* da Universidade de Berkeley [TERTIARY], um *cluster* de HS e HA que visa oferecer aos clientes discos virtuais de grande capacidade e alta disponibilidade implementados pela agregação dos recursos de discos convencionais dos nós computadores.

2.2.4 Economia de *clusters*

Muitos dos benefícios dos *clusters* ficam evidentes a partir das descrições acima. Entretanto, há custos óbvios (e outros não assim tão óbvios) envolvidos. Por exemplo, no mais simples *cluster* de disponibilidade elevada, tendo-se um sistema completamente redundante de espera para o caso em que uma falha do sistema primário forneça o benefício da disponibilidade para a aplicação, dobra-se virtualmente o custo do *hardware* e do *software*. Em um *cluster* com um grande número de nós, tal como um *cluster* para computação de alta performance, freqüentemente é desejável minimizar o custo das licenças de *hardware* e de *software* envolvidas. Em um grande *cluster* para computação de alto desempenho, a idéia é ter um ambiente tão simples e eficiente quanto possível e que maximize o trabalho realizado nos nós individuais. Sistemas operacionais tais como Linux, que podem ser obtidos virtualmente de graça fornecem

uma solução muito econômica para o problema de licenciamento de sistema operacional em um grande número nós. Além disso, devido à sua popularidade, há muitas ferramentas, utilitários e aplicações disponíveis para ajudar a construir e controlar um *cluster*. Muitos destes programas estão disponíveis gratuitamente ou por um custo muito razoável. Adotar *software* de sistema operacional econômico tais como Linux e FreeBSD, programas baseados em Linux e *hardware* baseado em processadores Intel e Powerpc, tal como servidores com processadores Pentium ou computadores Macintosh, tornam o custo total de montagem de um *cluster* muito mais baixo do que se verificou no passado. Conseqüentemente, muitas empresas estão começando a empregar tais *clusters* em ambientes onde antes não teria sido justificável. Nas seguintes seções, descreveremos os diferentes tipos de nó que normalmente se encontram nos *clusters*, os tipos de funções que cada um deve realizar para se operar com sucesso um *cluster* e os tipos de redes de comunicação utilizadas.

Figura 2-1 O projeto "Tertiary Disk" da Universidade de Berkeley

Figura 2-2 O cluster "Tertiary Disk" e sua implementação utilizando armários (*racks*) para organização dos nós.

2.2.5 Componentes de um *cluster*

Os *clusters* são compostos de vários elementos fundamentais. O principal elemento é o nó, o qual consiste de ao menos um processador, memória e alguma interface de comunicação. Os nós de um *cluster* típico são descritos na seção 2.3. Um outro componente fundamental de um *cluster* é a sua rede de interconexão de nós, cujos tipos mais comuns são abordados na seção 2.4. Subsistemas de Entrada e Saída convencional (2.5), de Armazenamento (2.6), de E/S Paralela (2.7), Sistemas de Arquivo Distribuídos (2.8.2) e Paralelos (2.8.3) e Sistemas de Gerenciamento, Controle e Monitoração (capítulo 5) são elementos importantes e abordados nas próximas sessões.

2.3 Os nós de um *cluster*

Os *clusters*, em especial aqueles para desempenho elevado, têm tipicamente um grande número computadores, os quais são geralmente chamados de nós. Estes nós são em geral configurados de modo idêntico. A idéia é que as tarefas individuais que fazem parte de uma aplicação paralela devem funcionar igualmente bem em qualquer que seja o nó para onde for despachada. Entretanto, certos nós de um *cluster* freqüentemente possuem diferenças físicas e lógicas. Nas seções seguintes, enfatizaremos essas diferenças, as funções lógicas de cada nó do *cluster* e os tipos físicos de nós.

2.3.1 Funções de um nó do *cluster*

Um *cluster* consiste de dois ou mais computadores trabalhando conjuntamente como uma unidade lógica única que provê serviços. Apesar de que externamente o *cluster* possa parecer um sistema único, as funções internas que fazem com que isto aconteça são bastante complexas. Ainda que possam ser dependentes do tipo específico de *cluster* e de seus objetivos, as seguintes funções lógicas são encontradas na maioria dos *clusters* (Figura 2-3). Lembre-se que estas são funções lógicas e em alguns casos múltiplas funções lógicas podem residir em um mesmo nó físico e, em outros casos, uma função lógica pode se estender por diversos nós físicos.

Figura 2-3 Estrutura lógica de um *cluster*
(reproduzido de [FERREIRA2002])

2.3.1.1 Nó de Computação

É nos nós de computação do *cluster* que as computações ocorrem de verdade. A maioria dos nós de um *cluster* é de computação. Baseado nos algoritmos de escalonamento do *cluster*, as tarefas são distribuídas pelos nós de computação para executarem a sua parte no trabalho total.

2.3.1.2 Nó de Controle

Os nós de controle do *cluster* provêm serviços que ajudam os outros nós do *cluster* a trabalhar em conjunto e obterem o resultado desejado. Nós de controle fornecem dois conjuntos de funções:

- Serviços de configuração automática dos nós, tais como *bootp*, *dhcp*, *DNS* e outras funções similares. Essas funções possibilitam que os nós sejam facilmente adicionados ao *cluster* e que estes possam comunicar-se com os outros nós.

- Escalonar quais as tarefas que devem ser executadas e por quais nós. Desse modo, se um nó de computação termina uma tarefa e fica disponível para realizar trabalho adicional, o nó de controle pode atribuir a ele a tarefa seguinte que necessita o trabalho de um nó de computação.

2.3.1.3 Nó de Armazenamento

Para a maioria das aplicações que rodam em um *cluster*, os nós de computação precisam ser rápidos, confiáveis e ter acesso simultâneo aos sistemas de armazenamento. Isto pode ser conseguido de varias maneiras dependendo dos requisitos específicos da aplicação. Em alguns casos, armazenamento pode ser diretamente ligado aos nós de computação. Em outros casos, os requerimentos de armazenamento podem ser alcançados através de um ou mais dispositivos compartilhados através da rede. Neste caso, os nós de armazenamento permitem e controlam o acesso aos subsistemas de armazenamento.

2.3.1.4 Nó de Instalação

Na a maioria dos *clusters* é comum que os nós de computação (e outros) necessitem ser reconfigurados e/ou reinstalados com uma nova imagem com uma frequência significativa. O nó de instalação prove as imagens e os mecanismos para uma fácil e rápida instalação ou reinstalação do *software* dos nós do *cluster*.

2.3.1.5 Nó de Gerenciamento

Clusters são ambientes complexos e o gerenciamento dos componentes individuais é muito importante. O nó de gerenciamento provê muitas capacidades e facilidades, tais como:

- Monitoramento dos nós individuais

- Envio de comandos de gerenciamento para os nós individuais a fim de corrigir problemas ou para prover comandos simples que realizam funções tais como *power-on/power-off*.
- Tratar eventos ou alarmes originários de vários nós do *cluster*

Não podemos subestimar a importância do gerenciamento do *cluster*. É uma atividade imperativa quando se está tentando coordenar as atividades de um grande número de sistemas.

2.3.1.6 Nó de Usuário

Os nós de um *cluster* em geral encontram-se em uma rede privada a qual não pode ser acessada diretamente por redes externas ou pela rede corporativa. Mesmo se fossem acessíveis, a maioria dos nós de um *cluster* estaria necessariamente configurada para prover uma interface de usuário apropriada. O nó de usuário é um tipo de nó que é configurado para prover uma interface adequada para usuários, possivelmente localizados em redes externas. Estes podem ganhar acesso ao *cluster* e solicitar que um trabalho seja executado ou acessar os resultados de um trabalho anterior.

2.3.2 Tipos de nó de um *cluster*

Agora que compreendemos os tipos de funções lógicas que os nós de um *cluster* possuem, discutiremos os tipos físicos de nós existentes, ou seja, as variações dos tipos de nó a partir de uma perspectiva do *hardware* dos mesmos. Geralmente, os nós de um *cluster* são categorizados em três tipos: Cabeça, Computação e Armazenamento. Na Figura 2-4 temos um exemplo de um pequeno *cluster* com diversos tipos de nós.

2.3.2.1 Nó-Cabeça (*Head Node*)

O termo *Head Node* é um termo genérico. O nó-cabeça é aquele que em geral hospeda as funções de gerenciamento e controle do *cluster*, mas pode ser utilizado de outras maneiras. O Nó-cabeça geralmente executa uma ou mais das seguintes funções lógicas enunciadas nas seções anteriores:

- Nó de Gerenciamento
- Nó de Controle
- Nó de Instalação
- Nó de Armazenamento
- Nó de Usuário

Em um *cluster* de pequeno porte, digamos, de oito computadores, todas essas funções podem ser combinadas em um único nó-cabeça. Em *clusters* de grande porte, com muitos nós, essas funções devem ser distribuídas ao longo de diversas máquinas, objetivando maior segurança, confiabilidade e também devido a razões ligadas a desempenho.

2.3.2.2 Nó de Computação

Os nós de computação formam o coração do *cluster*. Os nós de usuário, controle, gerenciamento e armazenamento são projetados para dar suporte aos nós de computação. É nos nós de computação que a maior parte das computações e processamentos são realizados. Esses nós são logicamente agrupados pelos escalonadores de tarefas para a realização de tarefas de acordo com as necessidades do trabalho ou das aplicações.

2.3.2.3 Nó de Armazenamento

Freqüentemente nas discussões a respeito de *clusters*, os nós de armazenamento são definidos como um terceiro tipo de nó. Entretanto, na

prática freqüentemente o nó de armazenamento é apenas uma versão especializada de um nó de computação ou de um nó-cabeça. A razão pela qual nós de armazenamento são designados como uma categoria à parte dos demais é porque os requerimentos de *hardware* e também do *software* para suportar dispositivos de armazenamento podem ser diferentes dos existentes em nós cabeça ou de usuário. Dependendo dos requerimentos de armazenamento e do tipo de acesso ao armazenamento desejado, pode ser necessário incluir-se adaptadores especiais, drivers de disco que suportem modos de redundância tais como RAID5, dispositivos de armazenamento ligados através de canais especiais tais como Storage Area Network [IMEXSAN], Fibre Channel e outros (Figura 2-5).

Figura 2-4 Um *cluster* e seus diversos tipos de nós (reproduzido de [FERREIRA2002])

2.4 Redes de *Clusters*

Os nós dentro de um *clusters* têm muitas razões para se comunicar. Afinal, eles estão cooperando entre si para fornecer um resultado total.

High Performance Parallel Application Cluster

(using VIA/SAN)

Figura 2-5 Um *cluster* de servidores com armazenamento via Storage Área Network (SAN) (reproduzido de [IMEXSAN])

Algumas das comunicações-chave que devem ocorrer incluem:

- **Comunicação interprocessos (IPC):** os nós em um *cluster* estão trabalhando juntos para fornecer uma solução total. Conseqüentemente, a tarefa que cada nó está executando é relacionada freqüentemente com as tarefas que outros nós estão executando. Exatamente como no caso de multitarefa ou múltiplas linhas de execução em um único sistema, há quase sempre a necessidade de comunicação entre os processos a fim de coordenar o processamento e gerenciar o acesso simultâneo a recursos compartilhados.
- **Operações de gerenciamento:** pelo fato dos *clusters* terem potencialmente um grande número de nós, é importante para a gerência e monitoração reconhecer quando um nó individual está

falhando e poder emitir proativamente comandos que podem impedir que um erro ocorra. Não é viável controlar individualmente os múltiplos nós de um *cluster* de um modo autônomo. Deve haver um meio de comunicação entre os nós de gerência com todos os outros nós do *cluster*.

- **Instalação de *software*:** Por uma série de razões, os nós individuais frequentemente necessitam ter seus *softwares* reinstalados. Isto pode dever-se a uma reconfiguração para uma tarefa específica a ser realizada pelo *cluster* ou simplesmente para recolocar o ambiente operacional após a ocorrência de uma falha de disco. De qualquer modo, é importante que imagens do sistema possam ser eficientemente definidas e distribuídas. Uma vez que tais operações podem ser realizadas em alguns nós ao mesmo tempo em que o resto do *cluster* está operacional e realizando trabalho, é importante que esta operação de empurrar o novo ambiente operativo para esses nós não afete adversamente o desempenho dos demais nós.
- **Acesso a armazenamento:** na maioria dos *clusters*, os nós individuais necessitam ler e/ou escrever dados em alguma espécie de dispositivo de armazenamento compartilhado. Em especial no caso de computação de alto desempenho (HPC), seria contraditório aos objetivos do *cluster* se o acesso a esse dispositivo de armazenamento se mostrasse um gargalo, cujo resultado seria outros nós tendo que esperar para acessar tal dispositivo. Portanto, é importante ter tanto uma rede eficiente para acesso rápido ao armazenamento e um sistema de arquivos de acesso eficiente e concorrente aos dados.

Dependendo das aplicações e dos requisitos de desempenho, as diversas comunicações necessárias entre os diversos nós do *cluster* podem ser

realizadas através de diversos e diferentes canais de rede. Assim, tipicamente tem-se mais de uma rede e mais que um tipo de rede realizando a interconexão dos nós do *cluster*.

2.4.1 Interfaces de Rede

Uma interface (adaptador) de rede tem a finalidade de conectar um nó do *cluster* a uma rede de comunicação. Um nó é configurado tipicamente com ao menos dois adaptadores de rede para cada rede a que se conecta: um deles mantém-se como interface de serviço, manipulando todo o tráfego de rede interno ao *cluster*, e um ou mais adaptadores mantém-se como interfaces de espera (*stand-by*). Os adaptadores em um *cluster*, em particular naqueles dotados de facilidades de HA, são identificados por uma rótulo e por uma função. Em redes TCP/IP, o rótulo do adaptador é o nome associado com um endereço IP específico. Assim, um único nó terá diversos rótulos de adaptador e endereços IP atribuídos a ele. O rótulo de um adaptador não é o *hostname* da máquina. Em um *cluster* de HA, cada adaptador tem uma função específica que é identificada pelo papel que este executa no *cluster*. A função de um adaptador pode ser: adaptador de serviço, de espera ou adaptador de carga.

2.4.1.1 Adaptador de Serviço

O adaptador de serviço constitui a conexão primária entre o nó e a rede. É a interface pela qual as aplicações dos usuários ou dos clientes na outra extremidade da rede alcançam os serviços que o nó está oferecendo. Um nó tem um adaptador de serviço para cada rede física a que se conecta.

2.4.1.2 Adaptador de Espera

O adaptador de espera ou adaptador de *backup* é um adaptador capaz de assumir o lugar do adaptador de serviço daquela rede. Pode ser configurado para assumir o endereço IP deste e, em certos casos, o

endereço de *hardware* desse adaptador de serviço. O adaptador de serviço pode estar no mesmo nó ou em um nó diferente no *cluster*.

O processo de mover o endereço IP de um adaptador cujo serviço falhou para um adaptador de espera no mesmo nó é denominado *troca de adaptador (adapter swap)*. O processo de mover o endereço IP de um adaptador de serviço de um nó que falhou para um adaptador de espera em um outro nó do *cluster* é chamado *tomada (takeover) de endereço IP (IPAT²)*. O processo de mover um endereço de *hardware* entre dois adaptadores de rede é chamado de *troca de endereço de hardware (HAS³)*.

Adaptadores de espera devem ser configurados em uma sub-rede física de comunicação diferente de todos os adaptadores de serviço no mesmo sistema, e seu uso deve ser reservado para ações de *takeover*.

2.4.1.3 Adaptador de Carga

A ação de tomada de endereço é uma facilidade de HA que permite que o adaptador de espera de um nó assuma o endereço de rede (IP) e/ou de *hardware* de um adaptador de serviço que falhou em um nó. Quando o nó falhado reiniciar, seu adaptador de serviço necessitará de um segundo endereço para proceder com a carga (*boot*) a fim coexistir na mesma rede com o nó de *takeover*. Isto porque seu endereço IP original já estará em uso na rede pelo nó de *takeover*. Assim, um rótulo de adaptador de carga e um endereço IP provisórios são atribuídos a cada adaptador de serviço que tiver sido configurado com a facilidade de tomada de endereço IP. Nessa configuração, o adaptador é denominado de *adaptador de carga (boot adapter)*. O nó que falhou dará procedimento a seu carregamento com este

² IPAT, sigla do termo inglês *IP Address Takeover*

³ HAS, sigla do termo inglês *Hardware Address Swap*

endereço e retoma seu endereço de serviço original somente após o nó de *takeover* ter liberado-o durante o processo de reintegração.

2.4.2 Tipos de Rede

Clusters de Alta Disponibilidade (HAC) em geral necessitam ter um esquema de identificação mais forte de suas rede e adaptadores de rede. Uma forma tipicamente utilizada para se classificar e identificar os adaptadores de rede já foi descrita em 2.4.1. Veremos agora a classificação das redes de um *cluster*. Em um *cluster*, a rede interconecta múltiplos nós e é definida por um *nome* e por *atributos*.

2.4.2.1 Nome da Rede

O nome da rede é em geral um valor simbólico, identificando uma rede que participe de um *cluster* de HA. Os processos do *cluster* usam este nome para determinar quais adaptadores são conectados à uma mesma rede física. O nome da rede é um valor arbitrário, mas deve ser usado consistentemente.

2.4.2.2 Atributos de Rede

Um atributo de rede é um valor que descreve alguma característica fundamental da mesma. Dentre estes, podemos citar:

- Endereço e Máscara
- Taxa de dados (*Mbps*)
- Modalidades de envio (*unicast, multicast, broadcast*)
- Ponto-a-ponto ou Multiponto
- Classificação no *cluster*

Destes atributos, o último corresponde a um valor que determina a função e a visibilidade dos nós dessa rede em relação aos nós-cliente do *cluster*. Uma rede participante de um *cluster* pode ser classificada como:

- **Rede Pública.** Também chamada de rede externa, conecta-se a múltiplos nós do *cluster* e permite conexões a sistemas cliente (externos) ao *cluster*. As redes *Ethernet*, *Token-Ring* e *FDDI* são usadas geralmente como redes públicas em *clusters*. A operação de troca de endereço de *hardware* (2.4.1.2) é suportada em todos esses tipos de rede exceto em redes *FDDI*.
- **Rede Privada.** Também chamada de rede interna ou rede de controle, essa rede fornece um canal de comunicação somente entre os nós do *cluster*. As tecnologias candidatas a rede privada de *cluster* são *Ethernet*, *FDDI* e *Myrinet*. Esta classe de rede é opcional e é usada para gerenciamento interno do *cluster* e para o sistema de coordenação de aplicações e serviços. Assim, *clusters* que provêm serviços distribuídos tais como sistemas de arquivos distribuídos ou subsistemas de coordenação e gerenciamento distribuído irão utilizar esta rede para seu correto funcionamento.
- **Rede Serial.** Essa classe de rede é tipicamente uma rede não-TCP/IP conectando pares de nós do *cluster*. É utilizada tipicamente como redundância para a rede privada e seu tráfego consiste essencialmente de mensagens de controle e *heartbeats*. Visa dar confiabilidade ao fluxo de mensagens de gerenciamento e controle caso o subsistema de comunicação via protocolos TCP/IP falhar. Uma rede serial pode ser uma barramento SCSI⁴

⁴ SCSI: *Small Computer System Interface* - um subsistema de E/S com topologia em barramento para interconexão entre computador e periféricos. Suporta discos, fitas, leitores e gravadores de CD/DVD.

diferencial em modalidade *Target Mode SCSI* ou uma conexão via porta serial RS232 simples, *null-modem*.

Cada nó de um *cluster* de HA deve ser conectado ao menos a uma rede pública e a uma rede serial. A conexão a uma rede pública deve fornecer serviços críticos aos usuários da outra extremidade e a conexão a rede serial deve impedir que o nó fique isolado. Havendo mecanismos para evitar o isolamento, os nós configurados para fazer *failover* de determinados recursos não o farão quando estes recursos estiverem ainda sendo utilizados pelo nó que os possui. Logicamente, esse comportamento é determinado (a) pela lógica do sistema de gerenciamento, que deve negar permissão de *takeover* aos nós que solicitarem essa ação em tais recursos e (b) pela troca de mensagens de coordenação e permissão através de canais de comunicação contingenciais. A isolação de um nó ocorre quando este perde todos os seus canais de comunicação com os demais. Isto pode ocorrer devido à falha do subsistema de protocolos TCP/IP ou das redes físicas sobre as quais os protocolos TCP/IP apóiam-se. A isolação de nós pode ser parcial, quando apenas um ou alguns nós ficam isolados, ou total, quando a totalidade dos nós ficam isolados. Se os nós do *cluster* continuam funcionando porém não podem se comunicar, temos uma situação que pode levar a gerência do *cluster* existente em cada um dos nós isolados a concluir que os outros nós estão inoperantes e iniciar uma reconfiguração do *cluster*. O resultado da isolação de um nó é imprevisível. Conseqüentemente, é essencial que cada nó tenha ao menos uma conexão a uma rede não-TCP/IP. Tipicamente, a gerência do *cluster* tem como ação padrão nessas situações de isolamento desativar os serviços ou iniciar um *shutdown* a fim de minimizar o risco de conflitos.

2.4.3 Tecnologias de Rede

Os tipos mais comuns de redes usadas em *clusters* são:

- *Ethernet* (10Mbps)
- *Fast Ethernet* (100Mbps)
- *FDDI* (100Mbps)
- *Gigabit Ethernet* (1Gbps)
- *Myrynet* (2/4Gbps)

Nas sessões seguintes descrevemos brevemente cada uma dessas tecnologias de rede e a sua utilização mais comum em *clusters*.

2.4.3.1 Ethernet e Fast Ethernet

O termo *Ethernet* refere-se à uma família de produtos de rede local (LAN⁵) cobertos pelo padrão IEEE 802.3, o qual define o que é comumente conhecido como protocolo de acesso CSMA/CD. Três taxas de dados são definidas atualmente, tanto para uso com fibras óticas quanto para cabos de pares trançados:

- *Ethernet* de 10 Mbps - 10Base-T e 10Base-F
- *Ethernet* de 100 Mbps - *Fast Ethernet* – 100Base-T e 100Base-F
- *Ethernet* de 1000 Mbps (1 Gbps) - *Gigabit Ethernet* – 1000Base-T e 1000Base-F

Esta última (1Gbps) será mais comentada no próximo tópico. Um padrão *Ethernet* de 10 Gbps está sob desenvolvimento e será publicado provavelmente como o suplemento IEEE 802.ae ao padrão base IEEE 802.3 ainda em 2002. A rede *Ethernet* sobreviveu como a tecnologia principal para LANs (é usada atualmente por aproximadamente 85 por

⁵ LAN, sigla para designar redes locais, do inglês *Local Area Network*

cento das LANs de computadores pessoais e de estações de trabalho no mundo). O protocolo tem as seguintes características:

- É fácil de compreender, implementar, controlar, e manter.
- Permite implementações de baixo custo.
- Fornece flexibilidade topológica extensiva para a instalação da rede.
- Garantia de interconexão e operação bem sucedidas entre produtos que seguem o padrão, independentemente de fabricante.

As redes Ethernet originais foram implementadas com uma topologia em barramento utilizando cabo coaxial grosso. Os comprimentos dos segmentos são limitados a 500 metros e até 100 estações podem ser conectadas a um segmento. Os segmentos individuais podem ser interconectados por repetidores desde que trajetos múltiplos não existam entre nenhum par de estações da rede e que o número destas não exceda 1024. A distância total do trajeto entre o par mais distante de estações não pode também exceder um valor máximo prescrito. Embora as redes *Ethernet* mais novas não sejam mais conectadas em uma configuração de barramento, algumas redes mais velhas em barramento ainda existem e são usadas. Desde o começo dos anos 1990, a topologia de interconexão preferencial da rede *Ethernet* passou a ser a topologia estrela, mostrada na Figura 2-6. A unidade central da rede é um repetidor de múltiplas portas (conhecido também como *hub*) ou uma *chave* de rede.

Figura 2-6 Uma topologia de conexão em estrela (*star*) (reproduzido de [CISCOETH])

Todas as conexões em uma rede de estrela são ponto-a-ponto implementadas com cabo de fibra óptica ou cabos de pares trançados. Na Figura 2-7 vemos as camadas lógicas do padrão IEEE 802.3 e seu relacionamento ao modelo de referência OSI. Como em todo padrão de protocolo IEEE 802, a camada de enlace de dados ISO é dividida em duas subcamadas, a Subcamada de Controle de Acesso ao Meio (*Media Access Control* - MAC) e a subcamada Cliente MAC. A camada física da IEEE 802.3 corresponde à camada física do ISO.

Figura 2-7 As camadas do padrão *Ethernet IEEE 802.3* (reproduzido de [CISCOETH])

A subcamada Cliente MAC pode ser uma das seguintes:

- LLC (*Logical Link Control*), se a unidade for um DTE (estação final). Esta subcamada fornece uma interface entre o MAC *Ethernet* e as camadas da parte superior da pilha de protocolos da estação. A subcamada LLC é definida pelos padrões IEEE 802.2.
- Uma entidade-ponte (*bridge*), se a unidade for um DCE (equipamento repetidor). A entidade-ponte fornece uma interface LAN-a-LAN entre LANs que usam o mesmo protocolo (por exemplo, *Ethernet* a *Ethernet*) e também entre protocolos diferentes (por exemplo, *Ethernet* a *token ring*). As entidades-ponte são definidas pelos padrões IEEE 802.1.

Na Figura 2-8 vemos as diferentes exigências de compatibilidade impostas pelo MAC e pelos níveis físicos para a transmissão de dados básica sobre uma rede *Ethernet*.

Figura 2-8 Compatibilidades necessárias entre as subcamadas MAC e PHY em uma rede IEEE 802.3 (reproduzido de [CISCOETH])

A subcamada MAC controla o acesso do nó aos meios físicos da rede e é específica do protocolo individual. Todo MAC IEEE 802.3 deve atender ao mesmo conjunto básico de requerimentos lógicos, independentemente de incluir ou não uma ou mais das extensões opcionais definidas pelo protocolo. A única exigência para uma comunicação básica (uma comunicação que não requeira extensões opcionais do protocolo) entre dois nós de rede é que ambos os MACs devam suportar a mesma taxa da transmissão. As camadas IEEE 802.3 físicas são específicas para cada taxa de dados de transmissão, da codificação do sinal e do tipo de meios físicos que interconectam os dois nós. O *Ethernet* gigabit, por exemplo, é definido para operar sobre cabo de fibra ótica ou par trançado, mas cada tipo específico de cabo ou de codificação de sinal requer uma implementação diferente da camada física.

Ethernet e TCP/IP são os dois padrões mais utilizados na maioria das redes de computadores. São tecnologias simples e de custo efetivo, além do que a tecnologia está sempre sendo melhorada. *Fast Ethernet* funciona

bem com máquinas tipo Intel Pentium II ou III de até 500Mhz de *clock* de CPU operando como nó de um *cluster* tipo *Beowulf* [STE1995]. *Ethernet* pode operar tanto em modo *half-duplex* quanto *full-duplex*, sendo que neste último modo e utilizando equipamento de chaveamento (chave *Fast Ethernet* ou *Switch*) é possível conseguir uma banda agregada de pico de até 200Mbps. Transmissão em modo *full-duplex* pode ser empregada tanto entre as portas da chave, entre um nó e a chave e também entre dois nós. Na Figura 2-9 podemos ver um modelo de chave *FastEthernet* da Cisco Systems Corp. e na Figura 2-10 vemos um adaptador *FastEthernet* da 3Com Corp. Maiores informações sobre *Ethernet* podem ser encontradas em [CISCOETH].

Figura 2-9 Uma chave *FastEthernet* da Cisco Systems (WS-C2924).

Figura 2-10 Um adaptador *FastEthernet* da 3Com.

2.4.3.2 FDDI

FDDI - *Fiber Distributed Data Interface* – um padrão ANSI denominado X3T9.5 [MIRCHAN93], é a especificação de uma rede LAN de velocidade 100Mbps com arbitramento por passagem de ficha (*token passing*) em topologia de duplo anel usando cabos de fibra ótica. A FDDI

é usada freqüentemente como tecnologia de espinha dorsal (*backbone*) de alta velocidade por causa de sua capacidade de sustentação da largura de banda e de atingir distâncias maiores do que o cobre. Existe uma especificação para transmissão sobre cabos de cobre relacionada, denominada CDDI (*Copper Distributed Data Interface*), que surgiu para fornecer o mesmo serviço de FDDI sobre o cobre a curtas distâncias. CDDI é, portanto, uma implementação dos protocolos FDDI sobre fios de cobre em configuração de pares trançados (*twisted pair* ou TP).

Figura 2-11 Os dois anéis de uma rede FDDI (reproduzido de [CISCOFDDI])

A rede FDDI usa a arquitetura de duplo anel com tráfego em cada anel fluindo em sentidos opostos (num movimento chamado de contra-rotação). Os anéis duplos constituem um anel primário e um anel secundário. Na Figura 2-11 podemos observar o movimento de contra-rotação dos anéis primário e secundário do FDDI. Durante a operação normal, o anel primário é usado para a transmissão de dados enquanto o anel secundário permanece inativo. A finalidade principal dos anéis duplos é fornecer confiabilidade e robustez superiores. Em caso de falha de um nó ou de um enlace de fibra, ocorre uma reconfiguração dos anéis de modo que um novo anel único seja efetivado e garanta as transmissões e

passagem de *tokens*. Na Figura 2-12 podemos observar o estado do anel duplo de uma rede FDDI em seu modo de reconfiguração devido à uma falha da estação 3, enquanto que na Figura 2-13 podemos observar o estado do anel duplo de uma rede FDDI em seu modo de reconfiguração devido à uma falha dos canais de fibra entre as estações 3 e 4.

FDDI define dois tipos de fibra ótica: *monomodo* e *multimodo*. Um *modo* é um raio da luz que entra na fibra em um ângulo particular. A fibra *multimodo* usa um diodo emissor de luz (LED) como o dispositivo gerador de luz, enquanto que a fibra *monomodo* usa geralmente *lasers*. As fibras *multimodo* permitem modos múltiplos de propagação da luz na fibra. Uma vez que estes modos da luz entram na fibra em ângulos diferentes, chegarão na extremidade da fibra em instantes diferentes. Esta característica é conhecida como *dispersão modal* (Figura 2-14). A dispersão modal limita a largura de banda e as distâncias que podem ser conseguidas utilizando-se fibras *multimodo*. Por esta razão, a fibra *multimodo* é usada geralmente para conectividade dentro um edifício ou um ambiente geográfico relativamente limitado. As fibras *monomodo* permitem somente um modo de propagação da luz através da fibra. Uma vez que somente um único modo é usado, a dispersão modal não existe com essas fibras. Conseqüentemente, a fibra *monomodo* é capaz de levar a informação a distâncias consideravelmente maiores e por isso é usada geralmente para a conectividade entre edifícios e em ambientes de larga dispersão geográfica.

Figura 2-12 Uma rede FDDI reconfigurada devido à falha de um dos nós (reproduzido de [CISCOFDDI])

Figura 2-13 Uma rede FDDI reconfigurada devido à falha de um dos enlaces de fibra (reproduzido de [CISCOFDDI])

FDDI especifica as camadas física e de acesso ao meio do modelo de referência OSI. O FDDI não é realmente uma especificação única, mas é uma coleção de quatro especificações separadas, cada uma com uma função específica.

Figura 2-14 Fibras *monomodo* e *multimodo* (reproduzido de [CISCOFDDI])

Figura 2-15 As quatro camadas que especificam o padrão FDDI

Combinadas, estas especificações têm a potencialidade para fornecer conectividade a velocidades elevadas entre os protocolos das camadas

superiores tais como TCP/IP e IPX através de meios tais como cabos de fibras ópticas. O FDDI é similar a *Ethernet* IEEE 802.3 e a *Token Ring* IEEE 802.5 em relação ao modelo OSI. Sua finalidade primária é fornecer conectividade entre as camadas de protocolos superiores comuns do modelo OSI e os meios físicos usados para conectar os dispositivos da rede. A Figura 2-15 ilustra as quatro especificações do FDDI e seus relacionamento e também a subcamada IEEE de Controle do Enlace Lógico (*Logical Link Control* - LLC). A subcamada LLC é um componente da camada 2 (ou camada MAC) do modelo de referência OSI. Maiores informações sobre FDDI podem ser encontradas em [CISCOFDDI].

2.4.3.3 Gigabit Ethernet

Gigabit Ethernet utiliza uma versão modificada do padrão ANSI X3T11 *Fibre Channel Physical Layer* (FC-0) para atingir 1Gbps de banda. *Gigabit Ethernet* suporta fibras ópticas multimodo e monomodo e também enlaces de curta distância utilizando cabos de cobre (100m).

Figura 2-16 Chaves GigaEthernet da Cisco Systems (Catalyst 3550-12T and 3550-12G)

A fibra óptica é o meio ideal para conectividade entre chaves e servidores e pode atingir grandes distâncias (de 500 metros a 10 Km). Na Figura 2-16 podemos ver dois modelos de chave GigaEthernet da Cisco Systems Corp.

2.4.3.4 Myrinet

Myrinet é uma tecnologia de rede de chaveamento de pacotes de alto desempenho projetada em torno de chaves simples, de baixa latência de comunicação, bloqueantes. Uma limitação é que as chaves bloqueiam incrementalmente. *Myrinet* tem um protocolo customizado e utiliza um segundo processador que faz a maior parte do trabalho de protocolo, evitando interrupções no processador principal do computador. Algumas das características da rede *Myrinet* são:

- Portas de chave e de interface operam a 2+2 Gbps *full-duplex*;
- Controle de fluxo, controle de erros, e monitoração de continuidade por *heartbeat* em cada enlace;
- Chaves de baixa latência, *cut-through* e *crossbar* com monitoramento provendo alta disponibilidade para as aplicações;
- Capacidade da rede para escalar a milhares de *hosts*, com taxas de dados até a ordem de terabits por segundo (Tbps);
- Caminhos alternativos de comunicação entre os *hosts*;

Pacotes podem ser de qualquer comprimento e assim podem encapsular outros tipos de pacotes, incluindo pacotes IP sem necessitar de uma camada de adaptação. Cada pacote é identificado por um tipo, de modo que *Myrinet*, assim como *Ethernet*, pode carregar pacotes de protocolos diferentes simultaneamente. Na Figura 2-17 podemos observar um adaptador *Myrinet* para computadores dotados de barramento PCI-64. Mais informação sobre *Myrinet* pode ser encontrada em [\[MYRINET\]](#).

Figura 2-17 Myrinet-2000-Fiber/PCI
interface card – um modelo de adaptador
Myrinet

2.4.4 Redes com Chaveamento

Uma rede pode possuir a característica de chaveamento. Muitas LANs possuem essa capacidade, tais como as rede *Ethernet* e *Myrinet*, o que significa que comunicações simultâneas (concorrentes) entre diversos pares de nós (*hosts*) podem ocorrer nessas redes e assim consegue-se vazões de dados elevadas, latências pequenas e desempenho excelente. Para dotar essas redes de capacidades de chaveamento é necessária a presença de um equipamento denominado *chave* ou *chaveador* (*LAN switch*). A chave é um dispositivo que possibilita um aumento no número de nós na rede e maior densidade de portas nos equipamentos de interconexão a custos menores do que os antigos equipamentos tipo ponte. Para esta razão, as chaves podem acomodar projetos de rede que se caracterizam por poucos usuários por segmento, aumentando desse modo

a largura de faixa disponível média por usuário. Para maiores detalhes sobre chaveamento de redes, consulte [CISCOLANSW].

2.4.4.1 Chaves de Rede em Clusters

Observando os ambientes de *cluster*, notamos que em muitos casos a quantidade de nós pode ser alta, em particular nos casos de *clusters* para HPC. Nesses ambientes, as aplicações demandam comunicação entre os nós com frequência muito alta, exigindo baixas latências, confiabilidade e boa taxa de dados. Para se alcançar esses requerimentos, as chaves são peças fundamentais em qualquer rede de *cluster*. Neste capítulo apresentamos um resumo do funcionamento de uma chave de rede para LANs e apontaremos sua posição no modelo de referência OSI.

2.4.4.2 Operação de uma Chave de Rede

A tendência em redes dotadas de poucos usuários por segmento é conhecida como microsegmentação. A microsegmentação permite a criação de segmentos dedicados a um usuário por segmento. Cada usuário recebe o acesso imediato à largura de banda completa e não tem que realizar contenção e compartilhamento da largura de banda disponível com outros usuários. Em consequência, as colisões (um fenômeno normal nas redes de compartilhamento de meio físico que empregam *hubs*) não ocorrem, desde que o equipamento e os adaptadores nas estações operem em modalidade *full-duplex*. No caso de *clusters* obtém-se desempenho máximo na comunicação ao se conectar cada nó do *cluster* em um microsegmento dedicado, ou seja, dedica-se uma porta da chave para cada nó, de modo a haver uso exclusivo daquela porta por aquele nó.

Uma chave de LAN realiza o envio dos quadros baseada no endereço de destino do quadro, um endereço de camada 2, realizando assim o chamado *Chaveamento de LAN em Camada 2* ou em *Chaveamento de 2º Nível*. Em alguns casos, a chave realiza o envio baseada também no endereço de

camada 3 contido no pacote de rede transportado pelo quadro. Nesse caso, denomina-se o processo de *Chaveamento de LAN Multicamada*. Uma chave de LAN é chamada também chave de quadros porque envia quadros de camada 2. Na Figura 2-18 vemos uma chave de LAN que fornece largura de banda dedicada aos dispositivos e estações a ela conectados e vemos também o relacionamento do chaveamento de LAN em camada 2 com a camada de enlace de dados (*data link layer*) do modelo OSI.

Figura 2-18 Uma chave de rede local (*LAN switch*) e a camada OSI onde ela atua.

2.5 Subsistemas de E/S

2.5.1 O Barramento SCSI

Acrônimo para *Small Computer System Interface*, SCSI é um padrão de interface paralela usado por computadores tais como Apple Macintosh, PCs, Servidores Intel, IBM RISC/6000, Servidores Sun Microsystems e por muitos outros sistemas UNIX, ligando dispositivos periféricos aos computadores. As interfaces SCSI fornecem taxas de transferência de dados muito mais rápidas (até 160 Megabytes por segundo) do que portas

seriais e paralelas padrão. Além disso, pode-se ligar muitos dispositivos a uma única porta SCSI. Portanto, SCSI é de fato um barramento de I/O, mais do que simplesmente uma interface. Embora SCSI seja um padrão do ANSI, há muitas variações dele. Assim, duas interfaces SCSI podem ser incompatíveis. Por exemplo, SCSI suporta diversos tipos de conectores e diversas taxas de dados. A seguir comentamos as variedades de SCSI encontradas atualmente.

2.5.1.1 SCSI-1

SCSI-1 surgiu a partir das especificações de uma interface de sistemas desenvolvida pela *Shugart & Associates* denominada SASI, que foi criada originalmente em 1979. O primeiro padrão SCSI foi aprovado pelo ANSI em 1986 como X3.131-1986. A fim de evitar confusão quando os padrões subsequentes de SCSI fossem estabelecidos, a especificação original foi rebatizada mais tarde "SCSI-1".

O padrão SCSI-1 define os princípios dos primeiros barramentos SCSI, incluindo o comprimento do cabo, características de sinalização, comandos e modalidades de transferência. É um padrão muito limitado, especialmente em relação aos padrões atuais. SCSI-1 define somente as mais fundamentais características de SCSI e das modalidades de transferência. Dispositivos que correspondem ao padrão SCSI-1 empregam um barramento estreito de 8 bits, com uma taxa de transferência máxima de 5 MB/s. Somente transmissão *single-ended* é suportada, com terminação passiva. Houve algumas dificuldades iniciais associadas a conseguir que o padrão ganhasse aceitação universal devido ao fato que muitos fabricantes implementaram subconjuntos diferentes de suas características.

SCSI-1 é hoje em dia obsoleto, e o padrão de fato foi aposentado pelo ANSI. Atualmente são empregados dispositivos SCSI-2 e SCSI-3, mais avançados, que atingem taxas de transferência mais elevadas.

2.5.1.2 SCSI-2

SCSI-2 é uma melhoria e uma extensão ao limitado padrão original SCSI (ou SCSI-1). O conjunto de comandos possíveis para controlar dispositivos SCSI foi padronizado e melhorado. Além disso, o padrão define novas características significativas como adições à especificação SCSI-1 original.

Fast SCSI: Este protocolo de transferência de velocidade elevada dobra a velocidade do barramento para 10 Megahertz, significando taxas de transferência de 10 MB/s com o cabeamento SCSI regular de 8 bits ou mesmo mais elevado quando usado com cabeamento *Wide SCSI*.

Wide SCSI: A largura do barramentos original do SCSI foi aumentada para 16 (ou 32) bits. Este aumento permitiu mais vazão de dados utilizando a mesma sinalização de dados a 10 Megahertz. O *Wide SCSI* substitui eventualmente os barramentos SCSI originais estreitos quando há necessidade de se empregar dispositivos de disco de maior velocidade.

O padrão SCSI-2 permitiu também outras melhorias, tais como maior capacidade de dispositivos por barramento. Nos barramentos *Wide SCSI* até 16 dispositivos são suportados (ao contrário de 8 com o SCSI regular). Os cabos e conectores também foram melhorados. SCSI emprega um grande número de cabos e conectores diferentes. SCSI-2 definiu novas conexões de elevada densidade, estendendo os conectores básicos de 50 pinos definidos em SCSI-1.

Também podemos destacar alguma inovações de SCSI-2 tais como:

Terminação Ativa: A terminação é uma consideração técnica importante na instalação de um barramento SCSI. SCSI-2 definiu o uso da terminação ativa, que fornece uma terminação de maior confiança para o barramento.

Sinalização Diferencial: Para conseguir comprimentos de cabo mais longos, a sinalização diferencial foi introduzida. Ela foi rebatizada mais tarde para "diferencial de alta tensão" (HVD) a fim de distingui-la da sinalização diferencial da baixa tensão (LVD).

Enfileiramento de Comandos: Um dos pontos fortes de SCSI é sua capacidade de permitir o enfileiramento de pedidos múltiplos entre dispositivos do barramento, simultaneamente. O enfileiramento de comandos foi introduzido pelo padrão SCSI-2.

Conjunto de Comandos Adicionais: SCSI-2 adicionou novos conjuntos de comando a fim de suportar o uso de dispositivos mais novos, tais como CD-ROMs, DVD-ROMs, *scanners* e dispositivos de mídia removível. Os antigos conjuntos de comando eram mais focados no controle de discos rígidos.

Houve também diversas outras mudanças menores no padrão. Em sua maior parte, foram mudanças técnicas de baixo nível. É importante notar que um dos critérios principais no projeto da criação de SCSI-2 foi a da compatibilidade reversa com SCSI-1.

2.5.1.3 SCSI-3

SCSI-3 constitui um conjunto de novos padrões SCSI que especificam maiores taxas de transferência de dados. Taxas da ordem de 40MBps, 80MBps e até 160MBps são possíveis. Atualmente, SCSI-3 tornou-se o padrão de mercado para dispositivos de armazenamento de alto desempenho em servidores e em *clusters* de servidores, havendo grande

diversidade de produtos e equipamentos no mercado, a preços competitivos.

2.5.2 O Barramento PCI

O barramento PCI – *Peripheral Component Interconnect* – é parte integrante de virtualmente qualquer computador moderno, sendo um padrão largamente reconhecido para interfaceamento entre periféricos e processadores. PCI substituiu os antigos barramentos ISA e EISA e tornou-se o principal meio para se adicionar periféricos na maioria dos computadores da classe dos PCs e também de muitos modelos de servidores mais poderosos.

O padrão PCI permite transferência rápida de dados a taxas de pico de até 132Mbytes por segundo nos barramentos de largura 32 bits e *clock* de 33Mhz. Uma extensão de largura 64 bits e mesmo *clock* permite taxas de pico de até 264Mbytes por segundo. Uma outra extensão que utiliza o barramento largo de 64 bits e o dobro de *clock* (66Mhz) já está disponível e provê taxas de pico de até 528Mbytes por segundo. Também, uma nova versão de PCI denominada PCI-X deve surgir em breve.

O barramento PCI permite interconexão direta entre quaisquer pares de dispositivos PCI, entre um dispositivo PCI e a memória principal do sistema ou entre o processador do sistema e os dispositivos PCI. O barramento suporta múltiplos arbitradores (*bus masters*) permitindo que qualquer dispositivo tome o controle do barramento e possibilita assim transferências de dados com acesso direto à memória (DMA) sem intervenção do processador. As transferências pode ser conduzidas concorrentemente com as operações do processador e o seu barramento de sistema a fim de evitar atrasos de processamento devidos a operações no barramento PCI.

Apesar de que a carga no barramento limita normalmente o número de *slots* PCI nos sistemas a tipicamente três ou quatro soquetes, cada dispositivo conectado pode representar logicamente até oito funções PCI distintas, totalizando assim até 32 dispositivos diferentes no barramento. Até 256 barramentos PCI podem coexistir em um único sistema, ainda que raramente sejam encontrados sistemas com mais do que dois barramentos. O padrão PCI inclui uma especificação completa a nível de bit dos registradores de configuração. Isto possibilita configuração automática dos dispositivos periféricos conectados ao barramento possibilitando reconfiguração do tipo *plug-and-play*.

Uma grande variedade de dispositivos PCI existe e é largamente empregada em computadores de *cluster*. Dentre elas podemos destacar: adaptadores SCSI, adaptadores RAID, adaptadores de Rede, adaptadores de vídeo e outros.

2.6 Armazenamento

Os nós de um *cluster* podem ter seus próprios dispositivos de armazenamento locais, em geral utilizados para armazenamento de sua cópia de sistema operacional, bibliotecas e aplicativos. Em última análise, o conteúdo desses dispositivos de armazenamento vai definir as funcionalidades daquele nó. Tipicamente, muitos nós constituem-se de réplicas um do outro e formam um grupo lógico de nós com certas capacidades, preparados para executar certos tipos de tarefas. Ocorre assim uma especialização do nó de acordo com o tipo de dados e informações contidas em seus discos locais. Um problema surge quando determinadas aplicações distribuídas necessitam manter estado consistente em todas as suas partes durante certas operações ou transações. Se esse estado é mantido em armazenamento de disco, é necessário que todas as réplicas sejam atualizadas atômicamente, o que pressupõe um protocolo

de mensagens de coordenação da transação, a fim de manter consistência do sistema distribuído. Essa operação protocolar requer um tráfego considerável na rede de comunicação. Essa estratégia pode não servir a determinados tipos de aplicações. Assim, é comum existir no *cluster* o conceito de disco compartilhado, aquele que é acessível diretamente por dois ou mais nós do *cluster* através de algum duto de E/S compartilhado. Esse tipo de arranjo possui a vantagem de permitir acesso a uma cópia única da informação e possibilita fácil tomada de recurso em caso de manobras ou de falhas. Em geral, apenas um nó possui acesso de escrita ao disco compartilhado a cada instante a fim de evitar conflitos. Também, existem técnicas de *hardware* e *software* que visam aumentar a confiabilidade e o desempenho dos dispositivos de armazenamento, aumentando sua tolerância a falhas. As técnicas empregadas em geral são de replicação dos dispositivos de armazenamento, dos dados e a utilização de distribuição espacial de dados. Arranjos constituídos de discos independentes e redundantes são em geral empregados e recebem a designação genérica de RAID. Um RAID é constituído por uma controladora de discos especial ou um *software* de controle e um conjunto de dispositivos de disco, formando um subsistema de armazenamento denominado RAID. Existem, portanto, subsistemas RAID implementados por *hardware* via controladoras de disco especiais e também soluções via *software* utilizando controladoras e discos convencionais. Além disso, existem também implementações de RAID que aproveitam da distribuição espacial de discos através dos diversos nós de um *cluster* e das redes de interconexão de alto desempenho disponíveis, formando sistemas de armazenamento altamente confiável, eficiente, de alto desempenho e aliando alta disponibilidade. São genericamente chamados de *Network RAID*.

2.6.1 Discos Compartilhados

Um disco compartilhado é um disco que está conectado fisicamente a múltiplos nós. Para tanto, alguma espécie de barramento de E/S que permita compartilhamento entre *hosts* dos dispositivos deve ser suportado e utilizado.

Figura 2-19 Um disco compartilhado pelos nós de um *cluster* através de barramentos SCSI (reproduzido de [CHIAKPO1997]).

Talvez o padrão de barramento de E/S dotado dessas habilidades mais largamente suportado e empregado pelas plataformas de *hardware* e de sistema operacional existentes seja o padrão SCSI, discutido em 2.5.1. Na Figura 2-19 podemos ver um esquema típico de compartilhamento de discos entre os nós de um *cluster* através de barramentos SCSI.

2.6.2 Volumes de Disco

2.6.2.1 *Volume Físico*

Um volume físico corresponde a um dispositivo físico de armazenamento e a todo seu espaço de armazenamento. Um volume físico é endereçado pelo hardware de E/S do sistema, tal como barramentos de E/S. Pode-se compreender um volume físico como um dispositivo físico, tal como uma unidade de disco rígido.

2.6.2.2 *Volume Lógico*

Um volume lógico corresponde a um espaço de armazenamento logicamente definido em um volume físico. Em muitos sistemas, um volume físico pode ser desdobrado em muitos volumes lógicos. É muito comum também utilizar-se a denominação de *partição* para um volume lógico.

2.6.2.3 *Volume Compartilhado*

Um *volume compartilhado* é um volume de armazenamento que reside em um disco compartilhado (2.6.1) e é visível aos muitos sistemas aos quais este pode estar fisicamente conectado. Todo sistema de arquivo que reside em um volume compartilhado é um *sistema de arquivos compartilhado*. Em um *cluster*, um volume compartilhado é geralmente definido como um *recurso do cluster*.

2.6.2.4 *Volume de Grupo*

Um *volume de grupo* é um volume de armazenamento constituído por diversos volumes físicos ou, em certos casos especiais, volumes lógicos,

agregados, compreendendo um volume de armazenamento cuja capacidade é proporcional ao número de volumes agregados.

2.6.2.5 Volume de Grupo Compartilhado

Um *volume de grupo compartilhado* é um volume de grupo que consiste inteiramente de volumes compartilhados e é visível aos muitos sistemas aos quais estes podem estar fisicamente conectados. Todo sistema de arquivo que reside em um volume de grupo compartilhado é um *sistema de arquivos compartilhado*. Em um *cluster*, estas entidades são geralmente definidas como *recursos do cluster*.

2.6.3 Concorrência de Acesso a Discos

Em um ambiente não-concorrente, cada recurso de disco é acessado por um nó apenas de cada vez, mas é sabido pelos outros nós que estes são capazes de toma-lo para si em caso de falha (*failover*) ou em caso de parada programada do nó que o detém (*takeover*). Quando uma falha ocorre, existe um *tempo* para que ocorra a tomada do recurso por algum outro nó.

Em ambientes de acesso simultâneo (concorrente), os recursos compartilhados de disco são conectados e acessados ativamente por mais de um nó ao mesmo tempo. Neste ambiente, não há ação de *failover* quando um dos nós falha, dado que os discos já estão conectados ativamente aos demais antes da falha. Nesses ambientes, tempos de *takeover* não são levados em conta. Porém, a integridade dos dados no recurso compartilhado deve ser garantida por algum mecanismo de coordenação entre os diversos sistemas que o acessam simultaneamente.

2.6.4 RAID

RAID⁶ é uma sigla para Arranjo Redundante de Discos de Baixo Custo. Consiste em diversas unidades de disco agrupadas para juntas oferecer uma grande variedade de finalidades e tipos de serviço diferenciados. A conglomeração desses dispositivos de disco pode ser usada para fornecer, por exemplo, redundância se caso um dos discos do arranjo falhar. Esse agrupamento de diversos dispositivos de disco pode ser configurado para aparecer ao sistema computador como uma única e grande unidade de disco. O RAID pode ser implementado tanto via hardware quanto via *software*, conforme será explanado em 2.6.4.2. O agrupamento dos dispositivos de disco pode ser feito de diversas formas lógicas, determinando o que se chama de nível do RAID. Com isso, pode-se obter também algumas características de desempenho surpreendentes. Compreende-se, portanto, que RAID pode ser vantajoso em diversos ambientes computacionais, inclusive em *clusters*. Examinaremos nos próximos tópicos algumas características dos subsistemas RAID. Para maiores detalhes sobre subsistemas RAID, consulte [RAIDEDU] e [BUYYA1999], cap. 19.

2.6.4.1 Vantagens do RAID

RAID pode prover proteção contra falha dos dispositivos de armazenamento, fornecendo um nível de imunidade a falhas que não é possível quando se emprega dispositivos de disco individuais. Os subsistemas RAID podem tolerar a falha de um único dispositivo de disco e continuar trabalhando. Na ocorrência de uma falha em um dispositivo de disco, há diversos cursos de ação possível. Se o subsistema for configurado com discos sobressalentes (*spare disks*), o subsistema RAID começa a recriar os dados que eram do dispositivo de disco que falhou no

⁶ RAID, do inglês *Redundant Array of Inexpensive Disks* ou *Redundant Array of Independent Disks*

dispositivo de disco de reposição. Esta reconstrução pode ocorrer em taxas selecionáveis pelo administrador ou quando os usuários e/ou os processos continuarem a acessar os dados contidos nesse conjunto RAID específico. A reconstrução ocorre em diversos modos, dependendo do nível do RAID. Em alguns casos, a informação está replicada em algum disco do arranjo ou está distribuída pelos outros discos. Em outros, grande parte da informação é recuperada a partir dos demais discos do arranjo e uma outra parte dela é recalculada com base em códigos e algoritmos de redundância e recuperação de informação, tais como paridade, *checksum* e CRC. Se o subsistema não tiver um dispositivo de reposição *on-line*, o subsistema RAID restante pode ainda continuar a operar. O dispositivo de disco defeituoso pode ser removido e substituído por uma unidade nova. Essa ação consiste essencialmente de uma troca a quente, a chamada *hot swap*. Logicamente, unidades de disco especiais para essa finalidade são empregadas. A partir da reposição, os dados podem ser reconstruídos da mesma maneira citada previamente.

2.6.4.2 RAID de Software vs. RAID de Hardware

Existem duas modalidades de implementação de subsistemas RAID: por *software* e por *hardware*.

Em poucas palavras, **RAID de hardware** é qualquer tipo de RAID que se desdobra em um subsistema controlado em nível de *hardware*, seja por um cartão controlador ou por outro dispositivo, independente do sistema operacional. Assim, antes que qualquer tipo de sistema operacional seja instalado no sistema (ou antes que ele se carregue, caso um sistema operacional já esteja presente) este nível da RAID já está habilitado e operante no sistema, pois ele é implementado no *hardware*. Em teoria, a perda de dados, arquivos e de sistema operacional devido a erros ou por questões de configuração neste tipo de configuração de RAID é menos freqüente e mais fácil de ser sanada, pois ele oferece mais facilidades para

o esforço de recuperação dos dados uma vez que a configuração dos discos é armazenada no próprio controlador, em *hardware*. A perda do dispositivo controlador em *hardware*, porém, torna a recuperação de dados muito mais difícil, uma vez que a configuração dos dispositivos de disco é desconhecida pelo sistema operacional (e de todas as ferramentas-padrão de recuperação de dados), pois este vê o espaço combinado de todas os dispositivos de disco como uma única estrutura lógica. Na Figura 2-20 podemos ver uma controladora RAID para discos SCSI Ultra160 da Intel.

Em um **RAID de software**, temos que a funcionalidade é obtida através do software do sistema operacional, na maioria dos casos. É uma solução de RAID em que o próprio sistema operacional cria, armazena e gerencia a estrutura lógica dos dispositivos de disco no arranjo, criando dispositivos virtuais para os serviços do sistema que enxergam o conjunto de discos em questão como uma única e maior unidade de disco. Nesta solução, são utilizadas controladoras de *hardware* e discos convencionais, o que implica em custos menores. O sistema operacional realiza todas as tarefas que no caso do RAID de *hardware* são feitas por controladoras independentes da CPU do sistema, o que implica em maior *overhead*. Também, todo tratamento de erros mais sérios e de outros problemas será feito pelo sistema operacional, havendo apenas o suporte das controladoras convencionais dos discos no tratamento de certos erros internos e recuperáveis.

Figura 2-20 Controladora RAID da Intel, de 2 canais SCSI Ultra160

2.6.4.3 Desempenho do RAID

A agregação de múltiplos dispositivos de disco, que aparecem para o sistema como um único dispositivo de disco maior, permite uma grande melhoria no desempenho. Os dispositivos de disco individuais podem contribuir mais para o sistema enquanto membros de um RAID do que sozinhos – de um modo similar à regra que diz que o todo é maior do que a soma das partes. O que acontece é que cada dispositivo de disco contribui para melhorar a vazão total da informação. Enquanto dados estão sendo lidos ou escritos nos dispositivos de disco, estes são distribuídos por todos os dispositivos de disco do arranjo. Isto permite uma utilização mais eficiente da largura de banda disponível e de um *throughput* agregado mais elevado. Esta vantagem de desempenho é vista quando se integra o RAID em um ambiente onde arquivos grandes são os mais freqüentes. Aplicações tais como CAD/CAM, GIS, bases de dados e armazéns de dados beneficiam-se das características de desempenho do RAID. O desempenho para situações de pequenos arquivos é melhorado também com a utilização de RAID. Configurando o RAID corretamente, pode-se melhorar os tempos de acesso aos dados, especialmente em ambientes onde o barramento SCSI é muito ocupado. O subsistema RAID permite o acesso simultâneo às informações contidas no arranjo RAID e usa toda a largura de banda disponível para transferir muitos arquivos pequenos de

uma só vez, ao invés de um ou dois arquivos grandes. Isto pode ser considerado como acesso paralelo aos dados, uma vez que RAID é configurada para desempenho ótimo com arquivos pequenos. Subsistemas RAID integrados a operações bancárias, financeiras, contabilidade de negócios, monitoração de processos e coleta de dados são candidatos ideais para este tipo de aumento de desempenho. Os benefícios que o RAID fornece ao ambiente computacional são alta disponibilidade e elevado desempenho de dados. A administração e a confiabilidade do sistema são realçadas pelas características da redundância do RAID e pelas vantagens de desempenho que estes fornecem a um grupo de dispositivos de disco individuais. Ao usar um subsistema RAID, somente um dispositivo de disco precisa ser controlado, ao contrário da situação onde existem muitos dispositivos de disco em diversas controladoras diferentes. RAID fornece um subsistema de armazenamento virtualmente à prova de falhas. Nos próximos tópicos discorreremos sobre alguns dos níveis de RAID mais utilizados.

2.6.4.4 RAID nível 0

Definição. RAID 0 é um arranjo de discos com armazenamento em modo *striped* (em tiras) sem tolerância a falhas. Requer ao menos dois dispositivos de disco para ser implementado. Na Figura 2-21 podemos ver um arranjo de discos compondo um RAID de nível 0 e a forma como uma sequência de informação representada pelos blocos simbólicos “ABCDEF..” é armazenada no mesmo.

Características e Vantagens. RAID-0 implementa um arranjo de discos em modo *striped*, onde os dados são quebrados em blocos ou tiras (*stripes*) e cada bloco é escrito em um dispositivo de disco separado. O desempenho de I/O é extremamente melhorado ao espalhar a carga de I/O através de muitas canais e discos. O melhor desempenho é conseguido quando os dados são quebrados por múltiplos controladores com somente

um dispositivo de disco por controlador Nenhum *overhead* devido a cálculo de paridade é envolvido. Tem projeto muito simples e fácil de implementar.

Desvantagens. Não um RAID verdadeiro porque não é tolerante a falhas. A falha de apenas um dos dispositivos de disco resultará em que todos os dados do RAID sejam perdidos. Nunca deve ser aplicado em ambientes de missão crítica.

Aplicações Recomendadas. Produção de vídeo, edição de imagens e em toda aplicação que requeira largura de banda elevada.

Figura 2-21 Armazenamento da sequência de dados ABCDEF... em um arranjo RAID de nível 0 composto por quatro dispositivos de disco.

2.6.4.5 RAID nível 1

Definição. RAID 1 realiza as funções de espelhamento e duplicação dos dados. Para obter bom desempenho, o controlador deve poder ser capaz de executar duas solicitações de leitura concorrentes e separadas e também realizar escritas concorrentes em duplicata em um par de discos espelhado. O nível 1 de RAID requer um mínimo de 2 discos para ser implementado. Na Figura 2-22 podemos ver um subsistema de armazenamento composto de quatro arranjos RAID de nível 1. Podemos observar na figura a forma

como a sequência de informação representada pelos blocos simbólicos “ABCDEF..” é armazenada no primeiro arranjo RAID 1.

Características e Vantagens. É possível uma operação de escrita ou duas de leitura concorrentes por par espelhado. Consegue-se uma taxa de leitura duas vezes maior que a taxa com discos simples, porém a taxa de escrita é a mesma que para discos simples. A redundância de 100% dos dados significa que nenhuma reconstrução é necessária no caso de uma falha de disco pois o disco de reposição já está ativo e é uma cópia do disco que falhou. Sob determinadas circunstâncias, RAID 1 pode suportar falhas múltiplas e simultâneas nos discos. É o projeto mais simples de um subsistema de armazenamento RAID.

Desvantagens. Apresenta o maior *overhead* de dados dentre todos os tipos de RAID (100%), sendo portanto altamente ineficiente em termos de custo. A função de RAID é feita tipicamente pelo *software* de sistema, sobrecarregando a CPU e degradando possivelmente o *throughput* quando operando em níveis de atividade elevados. A implementação através de *hardware* é altamente recomendada. Não suporta a troca a quente de discos quando implementado por *software*.

Aplicações Recomendadas. Folha de pagamento, Contabilidade, Finanças, qualquer aplicação que requeira disponibilidade muito elevada.

Figura 2-22 um subsistema de armazenamento composto de quatro arranjos RAID de nível 1

2.6.4.6 RAID nível 0+1

Definição. RAID 0+1 visa combinar o alto desempenho do RAID nível 0 com a alta disponibilidade e confiabilidade do RAID nível 1. Requer ao menos quatro volumes de disco para ser implementado. O RAID 0+1 não deve ser confundido com o RAID 10. Na Figura 2-23 podemos observar um RAID 0+1 composto de quatro discos. O primeiro par compõe um arranjo de *data striping* e é espelhado nos dois discos do outro par. A seqüência de dados simbólica “AEBFCGDH” foi armazenada no arranjo.

Características e Vantagens. O RAID 0+1 é implementado com uma disposição espelhada, cujos segmentos são arranjos RAID 0. O RAID 0+1 tem a mesma tolerância a falhas que o nível 5 de RAID. O RAID 0+1 apresenta o mesmo *overhead* para a obtenção de tolerância a falhas que o espelhamento sozinho (RAID-1). Taxas elevadas de I/O são conseguidas graças aos múltiplos segmentos de listras (*stripes*) de dados. É a solução ideal para os ambientes que necessitam de desempenho elevado mas não necessitam de confiabilidade extrema.

Desvantagens. Uma única falha em algum dispositivo de disco fará com que todo o arranjo transforme-se, essencialmente, em um RAID de nível

0. É muito caro e tem grande *overhead*. Todas as cabeças de leitura/gravação dos discos devem mover-se em paralelo para as trilhas solicitadas, caso contrário haverá queda no desempenho. Apresenta escalabilidade muito limitada por um custo inerentemente elevado.

Aplicações Recomendadas. Servidores de Arquivo em geral. Aplicações de processamento de Imagens.

Figura 2-23 Um RAID de nível 0+1 composto de quatro discos. Ocorre *striping* e espelhamento simultâneo dos dados.

2.6.4.7 RAID nível 5

Definição. RAID 5 é um arranjo de discos independentes que opera por distribuição dos blocos de dados e de paridade. Os discos são divididos em linhas (áreas) denominadas *ranks*. Cada bloco de dados é escrito por inteiro em um dos discos de dados, em um determinado *rank*, seguindo um algoritmo de distribuição. A paridade para blocos em um mesmo *rank* é gerada durante a escrita e gravada em uma posição distribuída de mesmo *rank*. Durante as leituras, os códigos de paridade são então verificados. O RAID de nível 5 requer um mínimo de 3 discos para ser implementado. É o nível mais versátil de RAID. Na Figura 2-24 podemos observar um

RAID 5 composto de 5 discos. Note o espaço em cada disco reservado aos códigos de correção e recuperação de informação (ECC) por paridade.

Características e Vantagens. O RAID 5 oferece a mais elevada taxa de transações de dados em leituras. Oferece uma taxa média de transações de escrita de dados. Apresenta um *overhead* de 20% para a implementação do armazenamento dos códigos ECC (paridade) nos discos de dados, o que significa eficiência elevada. Apresenta boa taxa de transferência agregada. A falha de um dos discos tem um impacto médio no *throughput* do subsistema.

Desvantagens. Apresenta o mais complexo projeto de controlador RAID. É mais difícil e demorada a reconstrução de um disco no caso de uma falha de disco e reposição (em comparação ao nível RAID 1). Taxa de transferência de dados para um bloco individual é a mesma que a de um disco simples.

Aplicações Recomendadas. Servidores *Web*, *E-mail* e de Notícias. Servidores de Banco de Dados. Servidores de Aplicação. Servidores de Arquivos. Servidores para *Intranets*.

Figura 2-24 Um RAID 5 composto de 5 volumes de disco.

2.7 E/S Paralela

2.7.1 O problema de E/S em *clusters*

Uma das maiores vantagens dos *clusters* é a grande quantidade de recursos disponível no sistema (discos, memória, unidades de fitas, etc.). Tradicionalmente, em ambientes de rede de computadores convencionais, a grande quantidade de recursos não era facilmente acessível por todos os nós da rede. Somente os processos de aplicação que executavam no mesmo nó que possui um certo recurso é que eram capazes de utilizá-lo, ao menos de modo eficiente. Ainda, se houvesse algum meio pelo qual o recurso pudesse ser acessado, os passos necessários para tal não eram em geral transparentes. Por exemplo, o mecanismo utilizado para se acessar um arquivo mantido em um disco remoto podia ser o protocolo FTP, o qual pode criar uma cópia local do arquivo. Em um *cluster*, tais recursos teriam de ser acessíveis por qualquer nó do mesmo, de um modo fácil e amigável. De fato, a condição ideal é aquela em que todos os recursos são compartilhados por todos os nós, de um modo transparente, por todas as aplicações. Os usuários devem ter a sensação de estar utilizando um sistema único, composto por todos os computadores do *cluster* e seus

recursos. A partir desta premissa, o desenho ideal de um sistema de E/S e de arquivos para um *cluster* deveria ser capaz de dar acesso a todos os discos e arquivos de um modo transparente ao usuário ou aos sistemas operativos e de controle do *cluster*.

Uma vez que *clusters* acabam sendo muito similares a computadores paralelos, as mesmas espécies de aplicações destes podem ser executadas em *clusters*. O problema com que se depara quando executando tais aplicações é que elas usualmente necessitam de um sistema de E/S de alto desempenho. Tais aplicações geralmente trabalham com grandes conjuntos de dados que em princípio não poderiam ser mantidos em memória. Assim, elas precisam de um sistema de armazenamento ou de arquivos que seja capaz de escrever e ler estes dados muito rapidamente.

Por fim, quando aplicações paralelas ou distribuídas são executadas em um *cluster*, o sistema de E/S deve permitir operações cooperativas. Todas as unidades da aplicação podem desejar realizar um acesso coordenado aos dados, e essa coordenação deve ser oferecida pelo sistema de E/S. Portanto, o sistema de E/S é um candidato a implementar internamente mecanismos de coordenação e sincronização entre seus elementos distribuídos, ou, ele pode apoiar-se em uma plataforma de serviços do *cluster* que seja capaz de prover tais facilidades. Neste trabalho, estamos investigando exatamente as arquiteturas e formas de implementação de tais serviços de suporte.

2.7.2 *Clusters* e o desempenho de E/S

Havendo a necessidade de sistemas de E/S ou de arquivos de elevado desempenho para as aplicações, é preciso primeiro examinar-se a estrutura do *cluster* e dos subsistemas de E/S de seus nós a fim de poder determinar a melhor maneira de utilizá-los de modo a se obter um bom sistema de E/S e de arquivos.

A primeira e mais óbvia vantagem de um *cluster* é a abundância de recursos disponível. Os sistemas possuem muitos discos que podem ser utilizados em paralelo, concorrentemente, aumentando a largura de banda nos processos de armazenamento e recuperação de dados. Pode-se também aproveitar a grande quantidade de memória existente para construir grandes *caches* de blocos de disco ou de arquivos de modo a desacoplar o desempenho dos discos do desempenho do sistema de E/S ou do sistema de arquivos.

A segunda grande vantagem encontrada em ambientes de *cluster* é a presença de redes de interconexão de alta velocidade e desempenho que conecta os nós do *cluster*. Tais redes rápidas permitem ao sistema delegar a nós remotos a execução de muitas tarefas. Pode-se, por exemplo, utilizar a memória de um outro nó para manter os blocos de *cache* que não couberam no *cache* local. Pode-se solicitar a outros nós que coletem diversas requisições de E/S a fim de se construir requisições maiores.

Por fim, *clusters* estão cada vez mais parecidos com máquinas paralelas, o que significa que muitas das lições aprendidas no desenvolvimento de sistemas de E/S para estas podem ser aplicadas nos ambientes de *cluster*. Nas próximas sessões abordaremos algumas soluções existentes para sistemas de E/S distribuídos e de alto desempenho para *clusters*.

2.7.3 VSD

O VSD - *Virtual Shared Disk* (Disco Virtual Compartilhado) [ATTANASIO94] é fornecido pela IBM e consiste de um subsistema distribuído que permite aos programas e aplicativos que executam em nós diferentes do RS/6000 SP e aos seus produtos para *cluster* mais recentes acessar um volume lógico (LV) como se fosse um dispositivo local de cada um dos nós. O VSD fornece também a possibilidade de se utilizar o disco em uma modalidade denominada HSD - *Hashed Shared Disk* ao se

utilizar um *driver* de dispositivo específico que permite que os programas de aplicação tenham seus dados armazenados em listras (*data striping*) através dos discos físicos do VSD, reduzindo assim os gargalos de E/S e pontos quentes possibilitando, portanto, um serviço tal como o oferecido por um RAID nível 0. Além disso, VSD pode operar com espelhamento de dados em discos físicos independentes ao longo do sistema, o que lhe confere o status de RAID nível 1. Essa variante do VSD é denominada RVSD – *Recoverable VSD*, ou disco virtual compartilhado e recuperável. O RVSD fornece alta disponibilidade de serviço às aplicações de VSD ao detectar falhas nos nós detentores de volumes VSD, podendo executar procedimentos de *failover* do nó servidor em falha.

Combinadas, essas características o colocam como um subsistema de armazenamento com serviços ao nível de RAID 0+1. Porém, uma vez que essa funcionalidade é realizada de modo distribuído pelos nós do sistema, pode-se classificar o RVSD como um *Network RAID*.

2.7.3.1 Arquitetura do VSD

Na Figura 2-25 podemos ver uma simplificação da arquitetura do VSD. Ela ilustra a implementação de dois discos compartilhados virtuais (*lv_X* e *lv_Y*). Cada volume lógico é local a cada um dos nós e são chamados de nós clientes diretos. O software de VSD distribui pedidos de I/O dos outros nós, chamados nós cliente, ao nó cliente direto (isto é, o nó servidor de VSD) e retorna os resultados aos nós cliente. Na figura, o nó X é o nó cliente direto para o volume lógico VSD denominado *lv_X* e o nó Y é o nó cliente para o VSD *lv_X*. Para o VSD *lv_Y*, o nó Y é o nó cliente direto e o nó X é o nó cliente. A distribuição de I/O é feita pela *driver* de dispositivo VSD que se posiciona acima da gerência de volumes lógicos (LVM) e do subsistema de rede IP. O *driver* de dispositivo é carregado como uma extensão do *kernel* em cada nó. Dessa maneira, os volumes lógicos podem ser acessíveis globalmente. Todo nó que possua

conectividade IP pode ser usado para comunicações com discos VSD. Entretanto, o desempenho depende do tipo de rede de interconexão adotado. No IBM RS/6000 SP, a chave de alto desempenho (*SP Switch*) é utilizada para a comunicação entre clientes VSD. A API (*Application Program Interface*) para um disco VSD é um dispositivo especial “cru” (*raw special device*).

Figura 2-25 A arquitetura do VSD – Virtual Shared Disk da IBM

Cada *driver* VSD em cada nó possui um *cache buffer*, denominado *LRU Cache* (*Least Recently Used Cache*) compartilhado por todos os VSDs configurados e servidos pelo nó. Esse *cache* é usado para armazenar os blocos mais recentemente acessados dos VSDs servidos por aquele nó. O objetivo é beneficiar-se do efeito clássico do *cache*, isto é, minimizar atividade de E/S física nos discos. Se o bloco requisitado é encontrado no *cache*, ele é lido deste ao invés de ser lido do volume lógico de disco. VSD possui mecanismos que garantem a coerência de seu *cache* nos nós usuários. Porém, outros níveis de *cache* tais como os existentes no *kernel* para acesso a dispositivos de bloco não são cientes das semânticas de VSD e podem apresentar incoerência de *cache* quando modificações em blocos

do VSD ocorrerem. Por essa razão, nessas situações é aconselhável utilizar o dispositivo em modo *raw device*.

VSD é uma facilidade de compartilhamento de volumes de discos pelo *cluster*, e não é um serviço de arquivo. Por essa razão, os clientes devem coordenar seu acesso aos volumes VSD a fim de evitar conflitos de acesso. No ambiente de *clusters* IBM, os clientes VSD devem apoiar-se nos serviços dos subsistemas de suporte PSSP [CHIAKPO1996] ou RSCT [FERREIRA2002] para realizar essa coordenação. Como exemplo, sistemas de banco de dados que podem operar em modo distribuído, tais como o IBM DB2, podem utilizar volumes de dados brutos VSD para armazenamento e compartilhamento eficiente de dados, mantendo uma visão única e consistente destes com confiabilidade. Porém, necessitam utilizar os serviços desses subsistemas citados para coordenar as ações entre suas partes.

2.7.4 DRBD

DRBD – *Distributed Block Device* [REISNER2001] é um módulo de *kernel* que possibilita a implementação de dispositivos de disco virtuais a partir de discos físicos distribuídos pelos nós de um *cluster*. Na atual versão, DRBD é capaz de trabalhar com *clusters* de até dois nós sob o sistema operacional Linux, implementando um serviço de *Network RAID* de nível 1. Um dos nós do *cluster* é denominado nó *ativo* enquanto que o outro é denominado nó *em espera* (*stand-by*). DRBD suporta diferentes protocolos a fim atender uma grande variedade de necessidades e de clientes. Os dispositivos virtuais DRBD podem alcançar entre 50 % e 98 % do desempenho teórico máximo. *Clusters* baseados em DRBD estão sendo usados em produção desde junho de 2000 e tem conseguido relativo sucesso, sendo usados para espelhar bancos de dados e para implementar servidores NFS de alta disponibilidade.

DRBD é implementado através de um *driver* de dispositivo para Linux que permite que se construa sistemas em *cluster* com espelhamento de discos distribuídos (não-compartilhados), os chamados *clusters "shared nothing"*. Essa arquitetura tem não somente a vantagem de que a distância física entre as cópias dos dados pode ser maior do que com discos compartilhados convencionais, mas é também mais econômica do que estas.

2.7.4.1 Protocolos DRBD

Protocolo A. Este protocolo sinaliza à aplicação o término de um pedido de escrita assim que o nó local tiver completado a escrita do bloco no disco local e este tiver sido enviado para fora através da rede. Sinaliza-se assim a conclusão da operação de escrita sem saber com certeza se o bloco chegou ao nó contendo o disco espelho. Naturalmente este protocolo não é apropriado para qualquer aplicação, uma vez que ele pode violar o comportamento transacional do sistema. Por exemplo, um banco de dados sinaliza a seu cliente que a última transação terminou com sucesso. Porém, os blocos modificados pela transação ainda estão a caminho do nó de espera. Se o nó ativo deixar de funcionar neste momento, o nó de espera irá realizar um *roll-back* da última transação uma vez que esta estava incompleta e continuará a oferecer o serviço aos clientes. Ainda que este protocolo não sirva para se espelhar bases de dados locais, ele é bom para espelhamento de discos à longa distância, uma vez que tem baixa penalidade de performance para o sistema remetente, em especial quando enlaces de comunicação de longa distância estão sendo empregados.

Protocolo B. O protocolo B é mais adequado em situações como a do banco de dados do exemplo anterior, a fim de torná-la altamente disponível. O protocolo B considera uma operação da escrita completa assim que receber uma mensagem de reconhecimento indicando que o bloco foi recebido pelo sistema em espera. Deve-se notar, entretanto, que

o protocolo B apresenta ainda um pequeno risco residual que existe quando o gerente do *cluster* não é ciente das propriedades do protocolo B. Por exemplo: o servidor em espera falha logo após emitir um reconhecimento e antes que tenha a possibilidade escrever os blocos no disco. O servidor ativo pode sinalizar ao cliente que a transação foi bem sucedida e falha também em seguida. O gerente do *cluster* decide reparar o servidor de espera de modo a transformá-lo no novo servidor ativo. Este servidor não terminou a última transação, porém o cliente tem registro de que a transação foi bem sucedida.

O protocolo B não é bom para espelhamento de grandes massas de dados que são continuamente atualizadas porque retardará as operações do sistema ativo quando enlaces de rede longos e de alta latência forem empregados. Isto é causado também pelo limitado número de *slots* de requisições do sistema operacional Linux. Se nenhum *slot* livre existir, uma aplicação que tente emitir uma requisição adicional ficará bloqueada até que algum pedido seja atendido e um *slot* liberado. Correntemente, Linux possui uma fila de requisições de E/S de 128 entradas.

Protocolo C. O protocolo C considera uma operação da escrita completa quando um reconhecimento do tipo bloco-foi-escrito é recebido do sistema em espera. Este protocolo pode garantir a semântica de transação em todos os casos da falha.

Figura 2-26 Arquitetura de um *cluster DRBD* de dois nós realizando espelhamento dos discos.

2.7.4.2 Arquitetura do DRBD

DRBD fornece um disco compartilhado virtual que possibilita um *cluster* de alta disponibilidade. Ele necessita, porém, de software de gerenciamento do *cluster* encarregado de monitorar os nós do *cluster* e de reconfigurar os nós em caso de falha de um dos nós. Atualmente, *clusters* DRBD são implementados utilizando-se os subsistemas *heartbeat* [LVS] e similares [HAL] a fim de prover essa funcionalidade mínima necessária. Na Figura 2-26 podemos observar a arquitetura de DRBD formando um típico *cluster* de dois nós. Notar que o nó em espera não utiliza o dispositivo DRBD pois podem existir problemas de consistência de dados entre as duas visões de sistema de arquivos nos dois nós, dado o fato de que *buffers* e *caches* existem nessas camadas e de que os sistemas de arquivo não são cientes das propriedades dos dispositivos de bloco de DRBD. O dispositivo no nó em espera é utilizado apenas na eventualidade de falha do nó primário, ativo. DRBD permite portanto manter a cópia em

espera sincronizada com a cópia primária, não sendo um sistema de arquivos distribuído.

2.8 Software de Cluster

2.8.1 Sistemas de Arquivo

Dependendo do ambiente e das aplicações que estão funcionando no *cluster*, pode-se escolher uma variedade de opções de armazenamento que fornecem o desempenho e a confiabilidade que se requer, conforme já visto em (2.5). Uma outra necessidade, além do armazenamento, é a de organizar e gerenciar esses volumes de informação de modo a serem acessíveis para as aplicações e serem também gerenciáveis pelos sistemas operacionais. A função dos *sistemas de arquivo* é dar essa capacidade aos sistemas operacionais e possibilitar o acesso a dados ao nível de arquivos para as aplicações. Existem diversos tipos de sistemas de arquivo, voltados para diversos ambientes e necessidades. No caso de *clusters*, dependendo do tipo do mesmo pode-se escolher sistemas de arquivo mais adequados e que atendam aos requisitos. Para aplicações de alto desempenho, sistemas de arquivo voltados para acesso com alto desempenho, simultâneo, paralelos, são indicados. Para aplicações de alta disponibilidade, sistemas de arquivo de alta confiabilidade, redundantes, são a melhor escolha.

2.8.2 Sistemas de Arquivos Distribuídos

Todo nó de um *cluster* pode possuir dispositivos de armazenamento de dados locais e nos quais sistemas de arquivo residem. Os dados e arquivos desses discos locais podem ser necessários em muitas situações a outros processos do *cluster* que executam em outros nós, diferentes daquele que é detentor do recurso de armazenamento. A necessidade de acesso remoto a arquivos entre os nós do *cluster* requer que os *clusters* adotem um ou mais

sistemas de arquivos distribuídos. A maioria dos sistemas de arquivos distribuídos possui uma ou mais das seguintes características, as quais os tornam indistinguíveis dos sistemas de arquivos locais:

- *Transparência de rede*: arquivos remotos podem ser acessados usando-se as mesmas operações ou chamadas de sistemas que são usadas para se acessar arquivos locais.
- *Transparência de localização*: o nome de um arquivo não é vinculado a sua localização na rede. A localização de um servidor de arquivos não é parte intrínseca do nome ou caminho do arquivo.
- *Independência de localização*: quando a localização física de um arquivo muda, o seu nome não é forçado a mudar. O nome do servidor de arquivo não faz parte do nome ou caminho do arquivo.

Nas seções seguintes faremos uma breve descrição de alguns sistemas de arquivos distribuídos. Alguns deles são sistemas cuja arquitetura é aberta e cujas implementações, em particular para os sistemas operacionais típicos de *clusters* Beowulf, são facilmente encontradas. Nessa categoria, temos o NFS e Coda. Já outros, tais como o AFS, são arquiteturas proprietárias e cujas implementações estão disponíveis em geral apenas para ambientes operacionais proprietários.

2.8.2.1 NFS

O Network File System (NFS) [SUNNFS] é um sistema de arquivos distribuído e foi desenvolvido pela Sun Microsystems in 1987. Oferece um serviço de arquivos distribuídos com um padrão aberto. A versão 2 (NFSv2) foi a primeira largamente disponível e utilizada. A versão 3 do NFS (NFSv3) surgiu em 1993, apresentando melhorias no desempenho, segurança e escalabilidade. O NFS permite que as estações de trabalho compartilhem transparentemente um sistema de arquivos que exista

fisicamente em um servidor central. O cliente e o servidor são conectados tipicamente através de uma rede local. NFS tem uma especificação aberta e que é adotada por muitos fabricantes de estações de trabalho e PCs, sendo documentado em vários RFCs.

Arquitetura e Projeto NFS.

Os objetivos do NFS eram inicialmente os seguintes: (1) Transparência - os programas não podem perceber a diferença entre arquivos locais e remotos. (2) Portabilidade – de arquitetura, de plataforma de SO e de *hardware* e (3) Independência de protocolos de transporte.

Os computadores participantes podem ser clientes e servidores, permitindo diversas formas de compartilhamento dos sistemas de arquivo. Para que um cliente tenha acesso aos arquivos remotos, o servidor exporta um diretório, geralmente a raiz de um sistema de arquivos local. Esse diretório remoto é montado pelo cliente sobre um diretório local. A montagem é executada uma única vez e o acesso ao diretório remoto é transparente aos usuários, ficando o diretório local oculto devido à montagem. Máquinas cliente completamente desprovidas de discos locais podem montar um sistema de arquivos-raiz e um arquivo de troca (*swap*) através de NFS. O NFS utiliza primitivas de RPC sobre XDR ⁷ para permitir a representação de dados externos no transporte de rede e possibilitar o compartilhamento de sistemas de arquivos entre máquinas com sistemas operacionais e plataformas de *hardware* diferentes, além de permitir também compartilhar diversos tipos de arquivos. A independência de protocolo é conseguida pela implementação de NFS sobre RPC ao invés de diretamente sobre UDP ou TCP, pois RPC pode ser transportado por qualquer protocolo de transporte. O NFS foi construído originalmente com RPC sobre o protocolo UDP, pois era suposto que o ambiente típico de

⁷ XDR - *eXternal Data Representation* – um padrão para codificação e descrição de dados (RFC1014).

operação de NFS seria o de uma LAN com baixa latência e baixa taxa de perda de dados. NFS é um protocolo *stateless*, significando que cada pedido de NFS é único, independente e carrega toda a informação necessária para ser executado. Desta forma, tem-se maior facilidade para a recuperação de problemas, tais como falhas de cliente e servidores. A versão 3 de NFS permite uso de RPC sobre TCP ou UDP. Com TCP, tem-se um melhor desempenho quando clientes acessam servidores através de uma WAN. A fim de evitar latências elevadas, cada cliente tem um único soquete TCP para cada servidor NFS acessado. Todo tráfego de e para o servidor é multiplexado neste único soquete.

Desempenho do NFS

A natureza *stateless* do servidor NFS faz com que as mensagens entre servidor e cliente sejam grandes, consumindo potencialmente muita largura de banda da rede. A vantagem de um protocolo *stateless* é que o servidor não é afetado quando ocorre falha de um cliente. A implementação original de NFS para a maioria dos sistemas operacionais permitia apenas um único servidor por sistema de arquivo. Isto apresentava limitações severas em relação a escalabilidade e, portanto, ao seu uso em *clusters*. A presença de um único servidor NFS para um sistema de arquivos tem como efeito serializar os pedidos dos clientes aos arquivos criando um gargalo problemático, principalmente nos pedidos de escrita. Pedidos de leitura não seriam tão afetados porque os clientes são capazes de manter *cache* dos arquivos localmente. Nas versões mais recentes, NFS permite que mais do que um servidor atenda pedidos de um mesmo sistema de arquivos, porém apenas um deles pode oferecer serviços de escrita e os demais oferecem serviços de leitura apenas. Ainda que essa possibilidade seja útil em alguns ambientes, tais como aqueles onde muitos nós *diskless* existem e fazem uso de sistemas de arquivo em modalidade somente leitura, essa funcionalidade não atende aos requisitos

da maioria dos *clusters*, em particular daqueles onde partes de uma aplicação distribuída necessitam de acesso simultâneo aos arquivos, havendo múltiplos pedidos paralelos ao sistema de arquivos, sendo estes muito freqüentemente pedidos de escrita. A partir da versão 3, NFS possibilita que mais do que um servidor atenda a um mesmo sistema de arquivos em modalidade de escrita e leitura, porém o problema de escalabilidade ainda é uma questão. Como já foi mencionado, devido à natureza *stateless* de NFS, ocorre uma maior facilidade na recuperação de falhas, porém existem algumas penalidades de desempenho intrínsecas devido ao fato de que cada pedido é único e independente, o que o torna mais trabalhoso. Um recurso utilizado para contornar essa questão de desempenho é a utilização intensiva de *caches* tanto nos servidores quanto nos clientes. O NFS versão 3 é capaz de manter coerência de *cache* entre os servidores que oferecem serviço de um mesmo sistema de arquivos, porém não existem mecanismos para garantir coerência de *cache* entre os clientes. Assim, podem ocorrer problemas de consistência de dados em aplicações paralelas ou distribuídas que utilizarem os arquivos de um servidor NFS em modo de escrita e leitura.

2.8.2.2 AFS

AFS – *Andrew File System*, foi originalmente desenvolvido na Carnegie Mellon University em uma parceria com a IBM em meados dos anos 1980. O seu propósito inicial era de superar alguns problemas de escalação existentes em outros sistemas de arquivos distribuídos tais como o NFS. O AFS provou ser capaz de reduzir a utilização de CPU e do tráfego de rede e ainda conseguir um acesso a arquivos eficiente para grande números de clientes. Em 1989 o desenvolvimento do AFS foi transferido para a Transarc Corporation, a qual evoluiu o AFS para DFS – *Distributed File*

System – que é hoje incluído como parte do DCE ⁸. Assim, o AFS tornou-se uma tecnologia proprietária e desse modo AFS é pouco presente e representativo em *clusters* da classe Beowulf, uma vez que Linux e FreeBSD, entre outros sistemas operacionais muito utilizados nessa categoria de *clusters*, são desenvolvimentos posteriores ao AFS. Recentemente, porém, surgiram versões de sistemas de arquivos distribuídos para esses sistemas baseados na tecnologia do AFS, tais como Coda e PVFS. A inabilidade das primeiras versões de NFS em escalar a sistemas contendo da ordem de 100 ou mais clientes motivou a pesquisa e a experimentação de arquiteturas de sistemas de arquivos distribuídos mais novas e mais escaláveis.

2.8.2.3 Coda

Coda [Satyanarayanan89] é um sistema de arquivos distribuído para ambientes de computação distribuída em grande escala composto de estações de trabalho rodando Unix e variantes tais como *clusters* da classe Beowulf. O projeto de Coda foi otimizado para as questões de alta disponibilidade e desempenho, mas fornece elevado grau de consistência mesmo à luz desses objetivos. Coda apresenta resistência à falha de servidores e de rede com o uso de dois mecanismos distintos, mas complementares. O primeiro mecanismo - replicação de servidores, consiste em armazenar cópias de um arquivo em múltiplos servidores. O outro mecanismo - operação desconectada, é uma modalidade de execução em que o *cache* local do cliente suporta temporariamente o papel do local de replicação. A operação desconectada é particularmente útil para suportar estações de trabalho portáteis. Medidas realizadas em protótipos demonstram que a perda de desempenho para se fornecer elevada disponibilidade em Coda é razoavelmente pequena.

⁸ DCE - *Distributed Computing Environment*, ambiente de aplicações distribuídas, desenvolvido pelo OpenGroup (<http://www.opengroup.org/dce>).

Arquitetura de CODA

Muitos aspectos de Coda são herdados do sistema de arquivos AFS (2.8.2.2). Assim como AFS, Coda faz uma distinção entre servidores, que são relativamente poucos em número, e os clientes, que são bem mais numerosos. Os servidores são vistos como fisicamente seguros, rodando software de sistema operacional confiável e monitorados pela equipe de gerenciamento. Os clientes, por outro lado, são vistos como elementos altamente volúveis, podendo ser modificados de modo arbitrário por seus usuários, sendo também dispersos fisicamente e podendo ser desconectados ou desligados a qualquer momento e por longos períodos. Embora tanto clientes quanto servidores rodem Unix ou uma variante, os usuários podem executar programas somente nos clientes. Os clientes e os servidores comunicam-se usando mecanismos de RPC ⁹ que suportem criptografia e autenticação mútua.

O modelo de consistência oferecido por Coda é uma aproximação daquele utilizado por AFS, que por sua vez é uma aproximação do modelo oferecido por Unix. Unix permite que múltiplos processos leiam e escrevam em um arquivo simultaneamente, onde as modificações feitas por um processo são imediatamente visíveis a todos os outros processos. AFS oferece um modelo de consistência que é bastante próximo ao de Unix e compatível com a maioria das aplicações, porém sendo altamente escalável. Processos que compartilham um arquivo a partir de um mesmo local vêem a semântica exata de Unix. Processos em locais diferentes vêem modificações no arquivo com a granularidade de operações de abertura e fechamento de arquivo. Assim, vemos que Coda não apresenta um modelo de consistência estrito, podendo haver momentos em que os dados de um mesmo arquivo apresentam-se a clientes distintos de modo não consistente. Isto pode representar um problema para certos tipos de

⁹ RPC - *Remote Procedure Call*, ou chamada remota de procedimento

aplicações paralelas, podendo mesmo não possibilitar sua correta execução.

Na ausência de falhas, o modelo de consistência de Coda é idêntico ao de AFS. Na presença de falhas, Coda tenta maximizar a disponibilidade, porém pode negar acesso a uma cópia de dados se estiver absolutamente certo de que a cópia é inconsistente, mantendo esta política também na presença de particionamentos de rede. Coda utiliza um mecanismo de controle de versão para detectar essas inconsistências. Ainda, os mecanismos de alta disponibilidade de Coda podem ser suprimidos para arquivos que devem permanecer consistentes o tempo todo pois estes não garantem consistência das réplicas. Conclui-se, portanto, que Coda é interessante pelos seus aspectos de elevada disponibilidade, desempenho e escalabilidade. Deve-se entretanto limitar seu uso a aplicações paralelas que não necessitem de consistência estrita de dados.

2.8.3 Sistemas de Arquivos Paralelos

É crescente o número de sistemas baseados em *cluster* utilizados para executar aplicações que envolvem e dependem não apenas de componentes relacionados à computação, tais como o número de processadores, velocidade de *clock* de CPU, tamanhos de *cache* e latência de memória. Muitas novas aplicações para *cluster* demandam, além de grande poder computacional, grandes quantidades de operações de E/S em dispositivos de armazenamento [STE2002]. Essas aplicações consomem ou produzem grandes conjuntos de dados ou utilizam bases de dados muito grandes. Nessas situações, um *cluster* pode ser especialmente atrativo devido ao fato de que cada um dos seus nós pode incluir um subsistema de E/S completamente funcional e dotado de dispositivo de armazenamento local. Ao utilizar-se de todo o *hardware* de E/S disponível como um sistema de E/S paralelo, o *cluster* pode conseguir desempenhos de E/S consideravelmente elevados a um custo baixo. Para se conseguir

isto, é necessário *software* que possibilite que esses recursos de *hardware* sejam orquestrados e disponibilizados para uso por aplicações de alto desempenho, ou seja, *software* que permita que os discos de um *cluster* sejam usados como um recurso de armazenamento único e de alto desempenho. Tais sistemas são denominados Sistemas de Arquivo Paralelos ou *Parallel File System* (PFS).

Um sistema de arquivos paralelo é um *software* para sistemas de aplicações paralelas que prove distribuição e acesso paralelo de dados. A distribuição de dados possibilita que os dados de arquivos sejam distribuídos ao longo dos discos conectados aos diferentes nós de um *cluster*. Já o acesso paralelo possibilita acesso fácil e coordenado aos dados dos arquivos pelas múltiplas tarefas de uma aplicação paralela. O objetivo primário de um sistema de arquivos paralelo é prover alto desempenho de E/S no acesso a grandes volumes de dados para aplicações paralelas ou distribuídas.

Sistemas de arquivo paralelos podem, entretanto, não apresentar bom desempenho quando utilizados por aplicações não-paralelas ou monotarefas, particularmente se estas realizarem pequenos acessos aleatórios aos dados. Sistemas de arquivo paralelos também não são voltados a prover alta disponibilidade de dados, redundância ou outras características de segurança, sendo sua função primária prover, como já foi mencionado, acesso com desempenho elevado.

2.8.3.1 Arquitetura de um PFS

Um sistema de arquivos paralelo geralmente consiste de dois componentes de *software*: o código cliente, o qual roda nos nós de computação do *cluster*, e o código servidor, o qual roda nos nós de armazenamento. Um tarefa de uma aplicação distribuída que é cliente do PFS apresenta suas requisições de E/S para o código servidor no nó de computação onde ela

está rodando. O código cliente decompõe a requisição e a envia para cada um dos nós de E/S onde os dados afetados ou solicitados residem. O código servidor no nó de armazenamento transfere dados entre a rede e seus dispositivos de armazenamento e vice-versa. Em uma requisição de E/S típica, a comunicação entre os nós de computação e os nós de E/S segue geralmente um padrão do tipo “todos para todos”. Para uma operação de leitura, os dados dos vários nós de E/S são distribuídos (*scattered*) para os diversos nós de computação. Por sua vez, nos nós de computação, os dados oriundos dos diversos nós de E/S são coletados (*gathered*) em um único *buffer* de leitura. Uma solicitação de escrita é similar, porém o fluxo de dados é inverso.

Um outro componente importante de um PFS é o seu conjunto de metadados de arquivos. Todo sistema de arquivos precisa manter metadados de seus arquivos, os quais indicam propriedades importantes de cada arquivo, tais como seu nome, tipo, proprietários e permissões de acesso. Em um sistema de arquivos paralelo, propriedades adicionais necessitam ser mantidas, relativas à distribuição física dos dados nos discos dos nós de E/S. Inclui-se aí os números dos nós de E/S que mantêm os dados do arquivo e a forma como os dados estão distribuídos ou fragmentados ao longo dos mesmos. Ao menos um dos nós do *cluster* deve operar como nó de gerenciamento de metadados do PFS, podendo-se também distribuir esses metadados ao longo de diversos gerenciadores ao longo do sistema a fim de se obter benefícios de desempenho e talvez maior disponibilidade e confiabilidade.

A seguir faremos uma breve descrição de dois sistemas de arquivos distribuídos e paralelos. O primeiro, PVFS, nasceu de um projeto de pesquisa da Clemson University em seus laboratórios PARL, foi em grande parte financiado pelo Goddard's Earth & Space Data Computing Division da NASA e é um projeto cujo código é aberto e em constante

desenvolvimento. O segundo, GPFS, é um produto comercial da IBM e é o nome comercial do sistema de arquivos Tiger Shark¹⁰ desenvolvido pelos laboratórios da IBM em Almaden.

2.8.3.2 PVFS

Ao mesmo tempo em que os *clusters* da classe *Beowulf* cresceram em popularidade como uma plataforma de computação paralela, o mesmo aconteceu com a demanda por *software* de sistema para esta plataforma. No ambiente computacional paralelo dos *clusters* atuais, nota-se que muitos dos componentes de software essenciais do sistema estão maduros e estáveis, tais como sistemas operacionais confiáveis, sistemas de armazenamento e de passagem de mensagens. Entretanto, uma área onde faltam produtos adequados ao nível de produção para esses *clusters* é a de sistemas de E/S paralela e também sistemas de arquivos paralelos [STE2002]. O projeto do PVFS – *Parallel Virtual File System* [PVFS] ou Sistema de Arquivos Paralelo Virtual, é um esforço para fornecer um sistema de arquivos de alto desempenho e escalabilidade para esses *clusters*. PVFS é um sistema de código aberto e liberado sob licença geral pública GNU. PVFS não requer nenhum *hardware* especial ou modificações no *kernel* de Unix. PVFS fornece quatro potencialidades importantes em um único pacote: (1) um espaço de nomes consistente através de todo o *cluster*; (2) acesso transparente para as aplicações e utilitários já existentes; (3) distribuição física dos dados existentes através de múltiplos discos em múltiplos nós e (4) acesso de alto desempenho a partir do espaço do usuário.

Arquitetura de PVFS

A fim de facilitar sua utilização, PVFS prove um espaço de nomes homogêneo por todo o *cluster*. Os sistemas de arquivo PVFS podem ser

¹⁰ Para mais detalhes, consulte <http://www.almaden.ibm.com/cs/shark>

montados em todos os nós simultaneamente e no mesmo ponto de montagem, possibilitando a todos eles acessar os arquivos do PVFS através de um mesmo esquema de diretórios. Uma vez montados, os arquivos e diretórios podem ser operados com os utilitários tradicionais de *Unix*, tais como o *ls*, *cp*, e *rm*. A fim fornecer alto desempenho no acesso aos dados do sistema de arquivos para um grande número de clientes, PVFS espalha os dados através dos nós de armazenamento do *cluster*, os denominados nós de E/S. Ao espalhar os dados através dos nós de E/S, as aplicações necessitam conhecer múltiplos trajetos até os dados, seguindo diversos caminhos através da rede e acessando múltiplos discos onde os dados são armazenados. Esta técnica elimina gargalos de E/S e aumenta assim a largura de banda potencial total de dados para os clientes, a chamada largura de banda agregada. Porém, seria trabalhoso para o usuário se ele tivesse que lidar com essa complexidade de acesso e assim a biblioteca e as extensões de *kernel* de PVFS ocultam essa complexidade para o usuário, facilitando o uso e fornecendo uma interface simples e consistente ao serviço de arquivos.

Enquanto que o mecanismo tradicional de chamadas de sistema operacional para acessar arquivos é conveniente e possibilita que todas as aplicações acessem arquivos armazenados em muitos tipos diferentes de sistemas de arquivo, existe um *overhead* considerável em fazê-lo através do *kernel*. Com o PVFS, os clientes podem evitar fazer pedidos para acessar o sistema de arquivos através do *kernel*. As aplicações podem utilizar a API nativa de PVFS, implementada por uma biblioteca. Esta biblioteca executa um subconjunto das operações UNIX que contatam diretamente os servidores PVFS. Esta biblioteca pode ser utilizada por aplicações ou por outras bibliotecas, tais como a biblioteca ROMIO MPI-IO para acesso de alta velocidade ao PVFS.

Figura 2-27 A nós de um sistema de arquivos PVFS (reproduzido de [PVFS])

A Figura 2-27 mostra como os nós podem ser atribuídos para uso com PVFS. Os nós são divididos em: (a) nós de computação, onde as aplicações são executadas, (b) nó de gerência, que manipula operações com metadados, e (c) nós de E/S nos quais os dados dos arquivos do PVFS são armazenados. Os nós de gerência e de E/S podem ser usados para computação também se assim for desejado. Para pequenos *clusters*, faz sentido sobrepor estas funções a fim de manter a utilização dos recursos a mais elevada possível, enquanto que para grandes *clusters* o processamento de E/S e de metadados será melhor se recursos dedicados forem alocados a eles.

Como já anteriormente mencionado, a API nativa de PVFS fornece acesso via espaço do usuário aos arquivos PVFS através de uma biblioteca (libpvfs). Esta biblioteca manipula as operações de *scatter/gather* necessárias para mover dados entre os *buffers* do usuário e os servidores PVFS, mantendo estas operações transparentes para o usuário. A Figura 2-28 mostra fluxo de dados no sistema PVFS para operações de metadados e na Figura 2-29 vemos uma operação de acesso a dados de arquivo. Para operações de metadados, as aplicações comunicam-se através da biblioteca com o servidor de metadados. Para acesso aos dados,

o servidor de metadados é eliminado do trajeto de acesso e os servidores de E/S são contatados diretamente. Este procedimento é chave para o fornecimento de acesso em alto desempenho agregado e escalável de PVFS.

Por fim, o suporte de *kernel* provido por PVFS fornece a funcionalidade necessária para se montar sistemas de arquivo PVFS nos nós do *cluster*. Isto permite que os programas já existentes acessem arquivos PVFS sem nenhuma modificação. Este suporte não é a rigor necessário para o uso de PVFS pelas aplicações, mas fornece meios extremamente convenientes de se interagir com o sistema. A Figura 2-30 mostra o que ocorre quando o suporte de *kernel* de PVFS é usado. Esta técnica é similar ao mecanismo usado pelo sistema de arquivos Coda (2.8.2.3). As operações são passadas pelas chamadas de sistema à camada VFS¹¹ do *kernel* e são colocadas em uma fila de serviço pelo *daemon* pvfsd, que recebe as operações do *kernel* através de um dispositivo de E/S especial. Este por sua vez comunica-se com os servidores PVFS e retorna os dados através do *kernel* à aplicação. Maiores detalhes sobre PVFS podem ser encontrados em [CARNS2000].

¹¹ VFS – Virtual File System – subsistema do *kernel* Unix que cria a API para os sistemas de arquivo.

Figura 2-28 Fluxo de dados em uma operação em metadados (reproduzido de [PVFS])

Figura 2-29 Fluxo de dados no acesso a um arquivo PVFS (reproduzido de [PVFS])

Figura 2-30 Acesso a arquivo PVFS através do *kernel* (reproduzido de [PVFS])

2.8.3.3 GPFS

O GPFS – *General Parallel File System*, ou Sistema de Arquivos Paralelo Geral [GPFS] é fornecido pela IBM para os computadores que rodam o sistema operacional AIX e, mais recentemente, Linux. Fornece acesso global a arquivos nos produtos de *cluster* da IBM, tais como aqueles para as plataformas RS6000 (HACMP e SP) e para aqueles baseados em plataforma Intel, tais como o eCluster1300. Arquivos criados no sistema de arquivos GPFS podem ser acessados em qualquer nó do *cluster* que execute o código GPFS. Para nós que não rodam o código GPFS, esses arquivos podem ser acessados usando o Network File System (NFS).

GPFS é implementado como um VFS padrão de Unix, o que significa que as aplicações que usam chamadas de *kernel* Unix padrão para o sistema de arquivos funcionarão sobre GPFS sem modificações. GPFS fornece também um mecanismo de trava de arquivos com granularidade de byte, possibilitando que aplicações paralelas acessem blocos não sobrepostos concorrentemente com um mínimo de contenção. GPFS oferece escalabilidade e disponibilidade elevada, permitindo que múltiplos servidores e discos sirvam um mesmo sistema de arquivos. Quando ocorre falha de um servidor, o sistema de arquivos continua disponível desde que algum outro servidor tenha acesso aos discos que armazenam os arquivos e que ainda haja conectividade de rede entre o servidor e o cliente. Se um disco falhar, GPFS continua a oferecer acesso ao sistema de arquivos desde que os dados contidos no disco em falha não sejam metadados mas sim dados do usuário, ou, sendo metadados, desde que estes estejam replicados em outro lugar. Tanto os dados do sistema de arquivos (metadados) quanto os dados dos usuários podem ser replicados ou espelhados a fim de fornecer um ambiente de alta disponibilidade.

Arquitetura de GPFS

A arquitetura de GPFS [HEGER2001] é baseada em dois componentes chaves: infraestrutura de alta disponibilidade e discos compartilhados. Nos ambientes de *cluster* SP da IBM, esses componentes são implementados pelos subsistemas PSSP – *Parallel Systems Support Programs* [CHIAKPO1996] e pelo subsistemas VSD/RVSD [ATTANASIO94]. Nos ambientes de *cluster* de servidores, esses componentes são implementados pelos subsistemas RSCT – *Reliable and Scalable Cluster Technology* [FERREIRA2002] e na tecnologia de conexão de armazenamento SSA¹². A Figura 2-31 mostra o relacionamento entre a gerência de volumes lógicos (LVM), o PSSP, o subsistema de disco compartilhado virtual (VSD), o subsistema de disco virtual compartilhado recuperável (RVSD) e o GPFS.

Cada nó onde se executa o serviço GPFS necessita ter acesso aos discos de dados que armazenam o sistema de arquivos local. GPFS utiliza o VSD (2.7.3) para realizar isto (ou RVSD, se for necessário um ambiente de alta disponibilidade). Também, GPFS pode acessar os discos através de interfaces e conexões a discos SSA. GPFS espalha os dados, tão igualitariamente quanto possível, através dos múltiplos servidores e volumes de disco a fim de se obter um serviço de elevado desempenho, concorrente, onde múltiplos servidores atendem a muitos pedidos simultâneos de acesso a dados em um mesmo arquivo ou sistema de arquivos. GPFS pode operar com múltiplos servidores por sistema de arquivos e portanto é altamente escalável. Ainda, GPFS é um *log file system*, ou seja, ele pode registrar todas as modificações efetuadas em seus arquivos e também oferece operações com semânticas transacionais, permitindo replicação e recuperação rápida em caso de problemas, tornando-o assim um sistema de arquivos de alta disponibilidade.

¹² SSA – *System Storage Architecture*, uma tecnologia de conexão de dispositivos de armazenamento com características de SAN

Figura 2-31 Arquitetura do GPFS
(reproduzido de [GPFS])

As características de GPFS.

GPFS aparece para as aplicações como um outro sistema de arquivos qualquer. GPFS é implementado como um conjunto de componentes de software separados incluindo um módulo da *kernel* e um *daemon* GPFS (o *mmfsd*). A extensão de *kernel* de GPFS satisfaz aos pedidos dos aplicativos para acesso a arquivos de vários modos: (a) utilizando os recursos que já se encontram disponíveis naquele nó, ou (b) enviando uma mensagem ao *daemon* GPFS para dar continuidade ao pedido. O *daemon* executa a operação de E/S, assim como também faz gerenciamento dos *buffers* de *cache* do cliente GPFS. Além disso, ele inicia operações adicionais de leitura avançada (*read-ahead*) quando reconhece certos padrões de E/S, assim como também conduz escritas por trás (*write behind*) para conseguir um efeito de *pipelining* eficiente. Muitos dos serviços que são necessários para GPFS operar são fornecidos pelo *daemon* GPFS.

O *cache* de GPFS fica situado em uma área fixa e dedicada (*pinned*) de memória em cada nó. O *cache* de GPFS é chamado de *pagepool*. O *pagepool* é usado como *cache* dos dados do usuário, de blocos indiretos e metadados. É o mecanismo de *pagepool* de GPFS que permite que ele implemente as requisições de *read()* e *write()* assincronamente através dos mecanismos de leitura avançada e escrita por trás. GPFS é *multithreaded*, de modo que assim que uma aplicação faz um pedido de escrita e o seu *buffer* é copiado no *pagepool*, o pedido de *write()*, na perspectiva da aplicação, está completo. Após isso, GPFS escalona uma linha de execução para completar a requisição de *write()* até a sua conclusão, enviando chamadas de E/S aos drivers dos dispositivos de disco.

O Daemon GPFS

O *daemon* é um processo *multi-threaded* com algumas linhas de execução dedicadas a funções específicas. Assim, serviços que requerem prioridade de atenção não são obstruídos por outras linhas de execução que ficam ocupadas com serviços rotineiros. O *daemon* GPFS comunica-se com outras instâncias do *daemon* em outros nós GPFS a fim de coordenar mudanças de configuração, recuperação, e atualizações paralelas de uma mesma estrutura de dados. GPFS incorpora *caching* no lado do cliente. O *daemon* de GPFS funciona em cada nó do *cluster* que é parte de um domínio ou de um *pool* GPFS. O *daemon* fornece diversos serviços ao nó local assim como aos nós remotos. Ele fornece os seguintes serviços locais: alocação de espaço em disco a novos arquivos, gerência de metadados do sistema de arquivos (manipulação de *i-nodes* e de blocos indiretos), alocação de travas apropriadas para gerenciamento de dados e de metadados, inicialização de operações de E/S em discos e implementação de segurança e quotas de uso. Porém, todas estas tarefas devem ser realizadas em um ambiente paralelo. A fim manter a consistência e a visão única do sistema de arquivos GPFS, o *daemon*

executa diversas tarefas adicionais. O *daemon* executa estas tarefas assumindo *personalidades*. Em um ambiente GPFS, um único *daemon* pode assumir mais do que uma personalidade. As principais personalidades são:

- **Gerente de Configuração.** Um único *daemon* GPFS no pool assumirá esta personalidade, e é geralmente o primeiro *daemon* a começar no pool (o mais velho). O Gerente de Configuração é responsável por tarefas de configuração dentro do pool GPFS, tal como seleccionar o *Gerente de Grupo de Stripe* para cada sistema de arquivos.
- **Gerente de Grupo de *Stripe*.** Apenas um *daemon* por sistema de arquivo assume esta personalidade. O Gerente da Configuração decide qual nó atuará como Gerente do Grupo de *Stripe*, para cada sistema de arquivo. Esse gerente processa mudanças no estado do sistema de arquivos, incluindo: adição de discos, mudança na disponibilidade de discos, reparos no sistema de arquivos. Processos de montagem e desmontagem são executados tanto por esse gerente quanto pelo nó solicitante do serviço. Esse gerente também controla as alocações de dados, possibilitando uso eficaz do espaço e melhorando o desempenho. Finalmente, o Gerente de Grupo de *Stripe* manipula os pedidos de *token* de acesso a arquivos solicitados pelos usuários, enviando-os ao Servidor de *Token* específico daquele ponto de montagem.
- **Servidor de *Token*.** Existe apenas um Servidor de *Token* por sistema de arquivo, situado junto ao nó do Gerente de Grupo de *Stripe* daquele sistema de arquivo. Se o Gerente de Grupo de *Stripe* for movido para um outro nó, o Servidor de *Token* correspondente move-se com ele. O status de um *token* é mantido

em dois lugares: no servidor de *token* e na gerencia de *token* do nó cliente, a qual detém o *token*.

- **Gerente de Metadados.** Para cada arquivo, existe um *daemon* GPFS assumindo esta personalidade. Um *daemon* de GPFS pode ser o Gerente de Metadados de múltiplos arquivos. Em quase todos os casos, o nó que teve o arquivo aberto pelo mais longo período de tempo contínuo é o Gerente de Metadados.

2.9 Outros componentes de um *cluster*

Além dos nós físicos, redes, dispositivos de armazenamento, sistemas de arquivo e *software*, os *clusters* possuem vários outros componentes-chave que precisam ser considerados. As seguintes subseções discutem alguns desses componentes.

2.9.1 Clientes.

Um cliente é um sistema que pode acessar os nós do *cluster* para obter serviços críticos. Em alguns ambientes de alta disponibilidade, é possível existir monitoração pelo equipamento *cliente* dos recursos do *cluster*. Esse monitor pode receber informações importantes sobre a forma de acesso melhor e o estado atual dos serviços do *cluster*.

2.9.2 Servidor de Terminal

Servidores de terminal conseguem atender a uma necessidade básica do gerenciamento do *cluster*, que é a de acessar qualquer um dos nós por um canal diferente da rede local de comunicação entre os nós. Geralmente, isso é necessário em situações em que se está instalando novos nós no *cluster* ou na primeira instalação dos nós, quando estes ainda não tem todos os seus parâmetros definidos. Também, em situações de falha de

comunicação de algum nó, pode-se tentar contacta-lo através da sua porta serial RS232-C.

Uma vez que os nós estão prontos e rodando, o administrador pode usar ferramentas mais comuns tais como *telnet* ou *rsh* para acessar e comunicar-se com os nós.

ALTA DISPONIBILIDADE

3 Alta Disponibilidade

Quando se necessita de níveis altos de disponibilidade para aplicações críticas, há duas escolhas arquiteturais, a saber: (1) elevada disponibilidade (HA) e (2) sistemas tolerantes a falha (*fault-tolerant*). Cada aplicação tem um conjunto único de características técnicas, de requerimentos únicos de disponibilidade e também de um ambiente operacional específico que favorece uma solução sobre outra. Em ambientes de grandes empresas e centros de pesquisas, uma combinação de *clusters* de HA com sistemas de tolerância a falhas pode atender esses requisitos e fornecer os benefícios adicionais que não poderiam, outrossim, ser conseguidos com uma única arquitetura sozinha.

3.1 Necessidade de Alta Disponibilidade

No passado, somente algumas aplicações eram consideradas críticas o bastante para requerer alta disponibilidade. Estas eram encontradas geralmente em grandes companhias e em certos setores específicos tais como o de serviços financeiros e de telecomunicações. Mais recentemente, conforme o custo de computação veio abaixo e conforme novas tecnologias tais como a Internet emergiram, os negócios eletrônicos introduziram muitas novas aplicações que não seriam possíveis mesmo há dois ou três anos. Estas aplicações novas melhoraram a produtividade, proveram melhor informação aos responsáveis pelas tomadas de decisões e aumentaram a velocidade de transações de negócio. Mas também fizeram as companhias muito mais dependentes de seus sistemas de informação computadorizados. Hoje, muito mais do que antes, certos negócios não podem operar eficazmente – ou talvez nem possam mesmo operar - se as aplicações-chave estiverem fora do ar. Felizmente, as tecnologias para sistemas de alta disponibilidade estão melhorando e progredindo graças aos últimos anos de pesquisa na área, além do que os custos destes sistemas estão caindo.

3.2 Definindo Alta Disponibilidade

Disponibilidade é definida tipicamente como uma porcentagem do *uptime*. Baseado na operação contínua de um sistema, 24 horas por dia, 7 dias por semana, 365 dias ao ano, a Figura 3-1 relaciona porcentagens de disponibilidade à quantidade de *downtime* por ano.

Disponibilidade	Downtime por Ano
99%	87 horas, 36 minutos.
99,5%	43 horas, 48 minutos.
99,95%	4 horas, 23 minutos.

99,99%	53 minutos
99,999%	5 minutos

Figura 3-1 Disponibilidade em função do *downtime* ao ano.

A necessidade de níveis muito mais elevados de disponibilidade não é limitada aos ambientes 24x7. Muitas aplicações devem estar disponíveis durante as horas de negócio normais ou por um período de tempo crítico dentro do dia. Toda a falha de sistema dentro destes períodos críticos é inaceitável para muitos usuários.

Em relação à comunicação, alguns desafios adicionais surgem nos complexos e heterogêneos ambientes contemporâneos, os quais combinam protocolos legados, servidores com múltiplos sistemas operacionais e uma série de dispositivos especializados. Nestes ambientes, não se pode esperar que as aplicações tenham controle de todos os elementos envolvidos na comunicação, o que significa em última análise que aplicações como soluções integradas fim-a-fim são virtualmente impossíveis. No entanto, é possível haver um controle global dos pontos de comunicação da rede a fim de melhorar sua confiabilidade e disponibilidade. Esse controle teria uma visibilidade de todo o sistema e poderia ser um elo de ligação de alta disponibilidade entre as partes das aplicações.

3.2.1 Sistemas Tolerantes a Falhas

Computadores tolerantes a falha têm um projeto onde o *hardware* é quase que inteiramente replicado a fim de permitir operação ininterrupta no caso de uma falha de um componente. Não há nenhuma perda de tempo ou de desempenho para a recuperação quando uma falha ocorre. Todos os índices de memória e toda a informação baseada em disco são preservados. É utilizado *hardware* com recursos de *self-checking*, o qual garante que mesmo os erros transientes são detectados antes que

problemas relacionados à integridade de dados possam ocorrer. Características tais como diagnóstico interno, isolamento de erros de componentes que apresentam falhas, capacidades de operação remota, serviços remotos e componentes substituíveis pelo usuário permitem que estes computadores sejam usado em aplicações de telecontrole e em locais muito distantes dos escritórios locais de serviço. Tais sistemas são baseados em geral em processadores conhecidos e de comportamento largamente testado, tais como os da família Pentium da Intel e os da família de microprocessador PA-RISC da HP e rodam versões-padrão do sistema operacional. Como exemplo deste tipo de sistema, podemos citar o Stratus Continuum[®] da Stratus Computer Inc.

3.2.2 Clusters de Alta Disponibilidade

Clusters de alta disponibilidade permitem que múltiplos servidores acessem conjuntamente unidades de armazenamento de disco compartilhadas ou espelhadas e também outros tipos de dispositivos a fim de que possam se recuperar rapidamente de possíveis falhas. As falhas de *hardware* ou de *software* que afetem um componente crítico ou um sistema (nó ou subsistema) inteiro provocam o disparo de uma ação de *failover*. Essa ação ocorre de um sistema (nó) do *cluster* (o que falhou) para outro (o que vai tomar ou assumir o serviço). A integridade dos dados e o acesso aos dados são preservados, em geral utilizando-se sistemas de disco dotados de barramento SCSI com acesso compartilhado ou discos replicados por *software* por comunicação via rede. O processamento das aplicações e o acesso aos dados baseados em disco são restaurados tipicamente dentro de alguns minutos, embora os tempos da recuperação variem, dependendo das características específicas da configuração da aplicação e do sistema. A configuração do *software* deve ser feita sob medida para operar nesse ambiente, além do que o desenvolvimento e teste de certificações do *failover* são necessários para assegurar o

comportamento apropriado da ação de recuperação. A administração de um *cluster* é mais complexa do que aquela para um sistema único, uma vez que os múltiplos sistemas devem ser controlados e as informações de configuração devem ser coordenadas através dos sistemas e nós. Os melhoramentos (*upgrades*) de *software* e outras operações de manutenção de *hardware* ou de *software* podem ser feitos com tempos de interrupção mínimos ao se ter aplicações que podem migrar de um nó do *cluster* para outro.

3.3 Arquiteturas de Alta Disponibilidade

A fim de se obter alta disponibilidade, os *clusters* de HA lançam mão de algumas técnicas que visam manter seus serviços ativos a maior parte do tempo, ou ao menos a maior parte possível deste. Esse tipo de *cluster* presume que certos conjuntos de nós são capazes de executar um mesmo serviço e que essa execução ocorra de modo transparente, isto é, não deve ser perceptível aos usuários ou clientes desses serviços qual é o nó ou conjunto de nós específico que está executando o serviço naquele momento. Para chegar a tal nível de disponibilidade, os *clusters* HA tem uma arquitetura e organização interna que visa separar os diversos aspectos e níveis de complexidade envolvidos nessa tarefa. Nos tópicos a seguir teceremos algumas considerações e faremos algumas introduções sobre termos e conceitos sobre arquiteturas de *clusters* de alta disponibilidade. Para tanto, faremos uso de alguns conceitos e características da arquitetura do sistema de gerenciamento de *clusters* proposto neste trabalho, o DSMCA, objetivando a comparação e exemplificação.

3.3.1 Clusters e Nós

Um cluster de HA é definido como um conjunto de nós, possivelmente de diversos tipos, um conjunto de redes de comunicação e interconexão dos

nós, um conjunto de serviços e um conjunto de recursos. Os diversos tipos de nós de um *cluster* já foram amplamente abordados no tópico 2.3 acima e assim não faremos maiores comentários aqui. Em relação às redes de interconexão e comunicação em *clusters* de HA, são utilizados diversos tipos de rede, de acordo com a finalidade específica e adequada às características de utilização. As redes utilizadas em *clusters* são tipicamente as dos tipos já mencionados na seção 2.4 acima e são redes com finalidades diversas, tais como o transporte de mensagens de controle, mensagens de monitoração, mensagens de pedido e resposta entre os clientes e os serviços, etc. Um *cluster* em geral consiste na organização de um grupo de nós com afinidades para a realização de um certo grupo de tarefas ou serviços. Desse modo, um conjunto de nós que compõe um *cluster* recebe uma denominação que os identifica. Essa denominação recebe o nome de *identificador de cluster*, ou *ClusterId*, para o qual utilizaremos a notação $[c]$ onde c representa o número ou nome do *cluster*. Os nós de um *cluster* são também denominados a partir do *cluster* ao qual pertencem e recebem individualmente o seu *identificador de nó*, ou *ClusterNodeId*, para os quais utilizaremos a notação $[c, n]$, onde c representa o número ou nome do *cluster* e n representa o número ou nome do nó do *cluster*. As redes de comunicação que um *cluster* utiliza para interconectar seus nós recebem também uma identificação lógica no contexto de um *cluster*, denominada *ClusterNetId*, para os quais utilizaremos a notação $[c, r]$, onde c representa o número ou nome do *cluster* e r representa o número ou nome da rede do *cluster*. Esses tipos de identificadores são geralmente numéricos, mas podem também ser textuais ou ambos. Na arquitetura de *cluster* DSMCA, optou-se por se utilizar identificadores híbridos, dotados tanto de referência numérica quanto textual, havendo identificadores para *clusters*, nós, redes e serviços. Na Figura 3-2 podemos ver um *cluster* DSMCA típico com as identificações numéricas de suas redes e nós. No caso, temos o *cluster* [1], que utiliza duas redes, a rede [1.1] e a rede [1.2] (o prefixo [1.] identifica uma rede do

cluster [1]). Já os nós tem identificação similar, havendo os nós [1.1], [1.2] até [1.n].

Figura 3-2 Cluster DSMCA típico

Por sua vez, como cada nó conecta-se a duas redes, deve haver uma identificação para cada ponto de conexão de um nó a uma rede. Essa identificação é chamada de *ClusterNodeAddress* e esta é formada pelos valores que identificam o *cluster*, o nó e a rede onde este se conecta. O *ClusterNodeAddress* associa um endereço IP específico de uma rede do *cluster* a um de seus nós. A notação do identificador *ClusterNodeAddress* é uma notação tripla $[c.n.r]$ onde c é a identificação do *cluster*, n é a identificação de um nó e r é a identificação de uma rede. Assim, o endereço do nó número três do *cluster* de número um na rede número dois (Figura 3-2) pode ser expresso como *ClusterNodeAddress* [1.3.2] e cujo valor é um endereço IP.

Uma rede pode ser utilizada por diversos *clusters*, não havendo em princípio restrição quanto à exclusividade entre redes e *clusters*. As redes são consideradas canais de comunicação onde o endereçamento é administrado pela gerência do *cluster*. O tipo primário de endereço em *clusters* DSMCA é o endereço IP – *Internet Protocol Address*. Assim, quaisquer tecnologias de rede que suportem o protocolo IP podem ser utilizadas, tais como as redes já citadas em 2.4.3. Na Figura 3-3 podemos observar dois *clusters* DSMCA operando com um par de redes compartilhadas. Desde que os nós em ambos os *clusters* utilizem endereços IP distintos de todos os demais nós, inclusive dos nós de outros *clusters*, não há problemas de conflito. Uma outra possibilidade é a de que um mesmo nó ou grupo de nós participem de diversos *clusters*. O *cluster* é um elemento de configuração lógica em DSMCA e o sistema suporta a definição de diversos *clusters* superpostos em um conjunto físico de nós. A única restrição aqui, caso esses *clusters* utilizem redes compartilhadas, é que para cada *cluster* que um nó participe ele deve utilizar um endereço distinto de todos os demais que ele já utiliza e de todos os demais adaptadores de comunicação daquela rede.

Na Figura 3-4 podemos observar como ficam os identificadores para um conjunto de seis nós que compõem três *clusters* identificados por *cluster* [1], *cluster* [2] e *cluster* [3]. Podemos notar que o *cluster* [3] utiliza os nós que já fazem parte dos *clusters* [1] e [2]. Nesse caso, os identificadores de endereço de nó em cada *cluster* devem estar associados a endereços distintos.

Figura 3-3 Dois *clusters* DSMCA compartilhando redes

Identificadores contendo o sufixo de uma certa rede devem estar associados a um endereço distinto naquela rede. Assim, o identificador de endereço de nó [1.1.2] deve ter estar associado a um endereço da rede 2, da mesma maneira que o identificador de endereço de nó [3.6.2]. Ainda que estes dois identificadores sejam de um mesmo nó físico, eles devem utilizar endereços distintos no contexto da rede 2.

Figura 3-4 Seis nós compondo três clusters com superposição.

3.3.2 Clusters e Serviços

Uma das características fundamentais de um *cluster* é a sua capacidade de executar serviços em benefício de um conjunto de clientes, mantendo-os à disposição destes. A característica fundamental de um *cluster* de HA é a de sustentar a disponibilidade dos serviços que oferece, mantendo-os funcionais de modo quase ininterrupto. *Serviços* são tarefas executadas por um ou mais dos nós do *cluster*. Um *serviço* define um *domínio de nós* do serviço, que a rigor vem a ser o conjunto de nós do *cluster* nos quais o serviço está habilitado a executar. Na arquitetura DSMCA, define-se o conceito de *serviço single-instanced* e *multi-instanced*. Um serviço *single-instanced* é aquele que executa em apenas um dos nós do *cluster*. Para esses serviços, ainda que a configuração de serviço especifique que o

domínio de nós do serviço tem cardinalidade maior que um, apenas um dos nós do domínio de nós do serviço será selecionado como o nó que executará o serviço. Os demais nós do domínio ficam habilitados a executar o serviço, porém permanecem como nós de espera por algum eventual problema, tal como uma falha no serviço ou no nó que correntemente executa o serviço. Na Figura 3-5 vemos alguns exemplos de serviços *single-instanced*. Podemos observar que, ainda que os domínios dos serviços contemplem mais do que um único nó, o serviço é executado em apenas um deles. Dessa forma, como há mais nós do que serviços, um destes fica ocioso.

Figura 3-5 Um serviço *single-instanced*

Um serviço *multi-instanced* é aquele que executa em mais do que um nó do *cluster*. O número de instâncias determina quantos nós executam concorrentemente o serviço no *cluster*. Para esses serviços, geralmente o domínio de nós do serviço tem cardinalidade maior que o número de instâncias do serviço, de modo a haver margem para uma manobra caso seja necessário se desligar um dos nós que executa o serviço ou também no caso de falhas de um dos nós.

Figura 3-6 Serviços *multi-instanced*

Na Figura 3-6 podemos ver um exemplo de um *cluster* dotado de serviços *multi-instanced*. Notemos que os serviços executam em mais do que um nó, tais como os serviços 1 (*double-instanced*) e 3 (*multi-instanced*). Serviços *multi-instanced* tem a vantagem de que a sua carga de trabalho fica distribuída por diversos nós. Porém, eles tendem a ser serviços mais complexos, devido à sua distribuição. Em geral, eles são construídos como aplicações distribuídas que têm ciência da distribuição espacial de seus processos. Desse modo, utilizam algum tipo de mecanismo de comunicação entre suas partes, em geral primitivas de alguma biblioteca padronizada de passagem de mensagem, abertas ou proprietárias. Em muitos casos e em muitas situações, as partes de um serviço distribuído necessitam comunicar entre si de um modo sincronizado a fim de coordenar determinadas ações críticas para o seu funcionamento. Para tanto, algumas aplicações implementam primitivas de comunicação em grupo proprietárias para atingir esse objetivo. Uma das propostas do sistema DSMCA é prover a essas aplicações uma interface aberta que ofereça esse tipo de serviço, de modo integrado e confiável, de modo a não haver a necessidade de se reimplementar para cada aplicação certas

funcionalidades comumente necessárias em ambientes de computação e execução de serviços distribuídos.

Na Figura 3-7 temos uma visão de dois domínios de nós de serviço em um *cluster* com cinco nós. Os nós 1 e 2 fazem parte do domínio de nós do serviço 1, enquanto que os nós 3, 4 e 5 fazem parte do domínio de nós do serviço 2. Para os clientes desse *cluster*, representados na parte baixa da figura como um conjunto de estações de trabalho, deve ser indiferente e transparente o fato de um serviço ser *single* ou *multi-instanced*. Também deve ser transparente o fato de que, vez por outra, um serviço *single-instanced* migre de um nó do seu domínio para outro, o mesmo valendo para mudanças nos nós de um serviço *multi-instanced*. Essas qualidades são fundamentais para a caracterização do *cluster* como dotado de serviços de alta disponibilidade.

Outra característica de *clusters* de HA fundamental para que a alta disponibilidade seja alcançável e tenha flexibilidade é a possibilidade de superposição dos domínios de nós dos serviços. Em um *cluster* HA, um nó pode participar de vários domínios de nós de serviço distintos, de modo que um nó pode participar na execução de serviços diversos e também flexibilizar os processos de migração de serviços em situações contingenciais, tais como horários de manutenção, correções, atualizações e falhas diversas.

Figura 3-7 Um *cluster* dotado de dois domínios de nós de serviço

Na Figura 3-8 temos uma possível configuração de um *cluster* DSMCA composto por oito nós e apresentando cinco domínios de serviço entre os quais há superposição. Os domínios de nós dos serviços 1 e 2 operam com conjuntos independentes de nós. Já os domínios de nós dos serviços 3, 4 e 5 superpõem-se aos domínios 1 e 2 de um modo arbitrário. Pode-se concluir desta figura que os nós 1, 2 e 3 podem executar tanto o serviço 1 quanto o 3, enquanto que o nó 4 pode executar os serviços 1 e 4, e assim por diante.

Figura 3-8 Domínios de nós de serviço superpostos

Uma questão a ser desenvolvida ainda é a forma como o serviço é alocado a um determinado nó (ou nós) do seu domínio. Essa questão é explanada no tópico seguinte.

3.3.3 Alocação de Serviços

No tópico anterior, introduzimos o conceito de domínio de nós de serviço, que podemos referenciar abreviadamente como *domínio do serviço*. O domínio de um serviço identifica os nós do *cluster* capazes de executar o serviço. Tanto para serviços *single* ou *multi-instanced*, é preciso que haja uma regra ou política que defina qual ou quais nós do domínio de um serviço vão assumir o mesmo num determinado instante. No sistema DSMCA existem diversas maneiras de se estabelecer tais regras. A mais simples consiste em se estabelecer um número de ordenação para cada um

dos nós pertencentes ao domínio do serviço. Esse número é um valor de *precedência*. Quanto menor o valor do número de ordenação de um nó, maior a sua precedência e maiores são as chances de que ele venha a assumir o serviço em questão. De um modo simples: o nó pertencente a um domínio de serviço que tem a maior precedência é aquele que em geral assume a execução do serviço. Esta regra não é rígida, porque existem diversas formas de se alterar as políticas de alocação de serviços a nós. Por exemplo, a precedência dos nós pode ser utilizada como uma seqüência de valores que determina um escalonamento de nós rotativo para um serviço. Dessa forma, toda vez que a gerência do *cluster* necessita selecionar um novo nó no domínio de um serviço, ela pode seguir a seqüência de precedências e atribuir o serviço ao nó seguinte, ao invés de atribuir estritamente ao nó com a maior precedência. Uma outra estratégia é a alocação com precedências dinâmicas, onde o valor de precedência de cada nó varia com o tempo em função de diversas condições, tais como carga de trabalho, uso de recursos e assim por diante. Neste caso, as precedências podem ser reavaliadas periodicamente e, a cada avaliação, os serviços migram para os nós que obtiveram os melhores valores.

Na Figura 3-9 vemos um *cluster* com quatro nós e a execução de três serviços. Na mesma figura, temos a descrição dos serviços em termos de seus domínios e também as precedências (estáticas) dos nós. Com base nessas informações, podemos deduzir o porquê dos serviços 1, 2 e 3 estarem correntemente executando nos nós mostrados. Primeiramente, analisemos o serviço 1. O domínio de nós deste serviço é $\{1,2\}$ e as precedências desses nós para esse serviço são respectivamente 1 e 2. Assim, é natural a escolha do nó 1 como executor do serviço. No caso do serviço 3, cujo domínio de nós é $\{1,3,4\}$, temos as para os mesmos os valores de precedência 3, 2 e 1, respectivamente. Assim, o nó 4 é o de maior precedência e portanto o serviço 3 está sendo nele executado.

Figura 3-9 Precedências dos nós para os serviços 1, 2 e 3

Usando o mesmo raciocínio, esse *cluster* teria um estado de execução de serviços diferente caso um dos seus nós não estivesse operacional. É o caso, por exemplo, se o nó 2 estiver no momento desligado ou apresentado falha anteriormente. Na Figura 3-10 podemos observar qual seria o estado final após a avaliação das precedências com o nó 2 não-operacional. Como consequência da existência de um nó não-operacional em um domínio de serviço, esse nó não é considerado na avaliação do nó que deve assumir o serviço. Desse modo, o serviço 1 fica em execução no nó 1 pois ele é o único nó que restou no domínio desse serviço. Já o serviço 2, que antes era executado pelo nó 2 (vide Figura 3-8) agora está rodando no nó 3, pois, ainda que este nó tenha precedência menor que o nó 2 para este serviço, ele será o nó selecionado uma vez que a precedência do nó 2 não é considerada. O serviço 3 continua rodando no nó 4, que é o nó de maior precedência para esse serviço. O nó 2 não faz parte do domínio desse serviço e portanto não afeta a escolha do nó de execução desse serviço.

Figura 3-10 Nó não-operacional (2) em um domínio de serviço

Uma outra situação-exemplo em que um nó pertencente a domínios de serviço vem a ficar não-operacional seria a mostrada na Figura 3-11. Agora, temos uma hipotética situação em que o nó 1 não está operacional, possivelmente devido a falha ou desligamento. O único serviço afetado foi o serviço 1, que migrou para o nó número 2.

Figura 3-11 Uma variação do caso anterior, em que o nó 1 não está operacional

Notemos neste último caso o possível não-balanceamento da carga de trabalho entre os nós operacionais. O nó número 3, apesar de operacional, não está executando nenhum dos serviços enquanto que o nó 2 assumiu a incumbência de executar dois dos serviços e o nó 4 executa apenas um deles. Essa é uma situação típica que ocorre com a atribuição estática de serviços a nós, onde o não-balanceamento da carga de trabalho pode ocorrer de modo relativamente fácil. Apesar disso, esse *cluster* suportaria a falha de um segundo e até de um terceiro nó e mesmo assim em alguns casos todos os serviços continuariam a ser executados. Na Figura 3-12 vemos o que acontece caso o nó 4 vir a falhar e ficar não-operacional. O nó 3 assume o serviço 3 pois ele passa a ser o nó operacional com a maior precedência dentre os restantes do domínio daquele serviço.

Figura 3-12 Falha de um segundo nó concomitantemente à falha do primeiro.

Mesmo neste caso, a distribuição dos serviços aos dois nós sobreviventes pode não ser a mais adequada. Se por exemplo a soma das cargas de

trabalho referentes aos serviços 1 e 2 for muito maior que a carga de trabalho imposta pelo serviço 3, então temos um possível desbalanceamento de carga. Por essa razão, a possibilidade de distribuição de serviços com base em outros parâmetros é altamente desejável.

Como exemplo, o sistema DSMCA prevê que os serviços podem ser distribuídos aos nós com base em diversas informações. Existem dois conjuntos de parâmetros que determinam a distribuição: parâmetros estáticos e parâmetros dinâmicos. Dentre os parâmetros estáticos, temos:

- Número do nó
- Endereço do nó
- Prioridade do nó
- Valor de precedência

O valor da prioridade de um nó é um parâmetro estático, atribuído a cada nó de um *cluster* e é independente do serviço ou serviços de que o nó participa. Esse parâmetro pode ser ajustado, por exemplo, quando os nós diferem muito em relação às suas capacidades e desempenho. Pode-se dar um valor de prioridade mais alta para, por exemplo, nós que tenham processadores mais rápidos e maior quantidade de memória, enquanto que para os nós que tem processadores mais lentos ou menor quantidade de memória, uma prioridade menor. O valor de precedência é também um valor estático que determina a precedência primária do nó. A precedência de um nó é uma média ponderada onde cada um desses parâmetros possui um peso. Para um mesmo serviço, esses pesos são iguais para todos os nós no seu domínio. Dessa forma, pode-se obter precedências baseados em diversos parâmetros do nó.

Dentre os parâmetros dinâmicos, temos:

- Percentual de utilização do processador
- Taxa média de operações de E/S
- Taxa média de operações de rede
- Taxa média de ocupação da RAM
- Taxa média de utilização de *swap* / memória virtual
- Quantidade média de arquivos abertos
- Quantidade média de conexões de rede

Para cada um deles, podemos especificar um peso específico e realizar uma soma ponderada de todos esses parâmetros. Parâmetros com peso zero não influenciam no valor final obtido. A média ponderada obtida dos parâmetros dinâmicos é usada como um parâmetro para o cálculo da precedência final juntamente com os parâmetros estáticos, cujo peso específico pode ser especificado. A média ponderada estática, M , é dada por:

$$M = \frac{(\sum p.P) + w.M_d}{(\sum p) + w}, \text{ onde } p \text{ é o peso de um parâmetro estático e } P \text{ é o}$$

valor do parâmetro estático; w é o peso de M_d , uma média ponderada dos parâmetros dinâmicos, dada por:

$$M_d = \frac{\sum d.D_d}{\sum d}, \text{ onde } d \text{ é o peso de um parâmetro dinâmico e } D_d \text{ é o}$$

valor corrente do parâmetro dinâmico. Dessa forma, se utilizarmos M como um valor de precedência para a escolha do nó que irá assumir um serviço, vemos que os parâmetros estáticos podem ser facilmente colocados como aqueles mais determinantes na escolha, ficando os valores

dos parâmetros dinâmicos como uma forma de *interferência* no processo de escolha caso o valor de M_d seja significativamente diferente de um nó para outro. Porém, pode-se controlar quanto os parâmetros dinâmicos influenciam no processo de escolha ao atribuir-se valores apropriados para w , o peso de M_d no valor do M final.

3.3.4 Serviços e Recursos

No tópico anterior, introduzimos o conceito de serviço, que vem a ser a unidade de maior granularidade administrada pela gerência do *cluster*. Um serviço é uma tarefa em execução por um nó ou conjunto de nós do *cluster*. Para executar um serviço, é necessário que alguns pré-requisitos sejam atendidos de antemão, caso contrário o serviço não conseguirá cumprir sua função ou nem mesmo conseguirá ser ativado. No sistema DSMCA, adotou-se o conceito de *recurso* como sendo a unidade funcional que permite ou colabora para que um serviço possa ser executado. Um serviço pode, portanto, ser definido como um conjunto de recursos. Executar um serviço significa então ativar os recursos que o compõe e que são necessários ao funcionamento do mesmo. A ativação desses recursos deve dar-se, na maioria dos casos, em uma seqüência pré-definida e adequada. Recursos podem ser entidades de *hardware* ou de *software*. Clusters são ambientes ricos em tipos e quantidade de recursos e sendo assim uma larga gama de serviços pode ser executada nos mesmos. Na Figura 3-13 podemos ver algumas das espécies mais comuns de recurso presentes nos ambientes de *cluster*.

No sistema DSMCA, recursos são entidades fortemente *tipadas*, de acordo com sua natureza, funcionalidade e características. O sistema DSMCA, por utilizar uma abordagem totalmente orientada a objetos distribuídos, visualiza os recursos como sendo um conjunto de objetos CORBA dotados de interfaces específicas que possibilitam seu gerenciamento e administração. Objetos de gerenciamento de recurso são implementados e

ativados no contexto de aplicações específicas denominadas *Gerentes de Recurso*, que por sua vez são programas residentes nos nós do *cluster* e que são ativados sob demanda sempre que um determinado tipo de recurso precisa ser ativado e/ou gerenciado.

Figura 3-13 Recursos típicos de ambientes de cluster

Em um *cluster*, é necessário haver um gerenciamento dos recursos de modo que os serviços que eles implementam possam ser ativados corretamente. O *cluster* tem em sua configuração um conjunto de recursos, denominados *recursos do cluster*. Em um *cluster*, somente poderão ser executados os serviços cujos recursos estiverem contidos no conjunto de recursos do cluster. A razão disto é que somente os recursos desse conjunto estão sob gerenciamento do *cluster* e têm definido um

gerenciador de recurso adequado. A não existência de um gerenciador de recurso para um determinado tipo de recurso impede que este seja incluso no conjunto de recursos de qualquer *cluster*. Na Figura 3-14 podemos ver a distribuição de um conjunto de *recursos de cluster* entre diversos serviços. Notemos que alguns dos recursos são distribuídos a mais de um serviço. Isto significa que determinados recursos são *compartilháveis*, podendo ser utilizados em vários serviços concorrentemente. É o caso, por exemplo, de dispositivos de disco, sistemas de arquivo, adaptadores de comunicação e outros. Quando um recurso tem esta capacidade e características, dizemos que o recurso tem o atributo de ser compartilhado, ou seja, seu atributo *Shared* é verdadeiro. É o caso, por exemplo, de um sistema de arquivos que pode ser recurso para diversos serviços. Um outro caso típico seria o de um adaptador de rede. Não é necessário, ainda que possível, que esses recursos sejam de uso exclusivo de um único serviço. Quando um recurso deve ser de uso exclusivo de apenas um serviço, ele deve ser marcado como *não-compartilhável*. Desse modo, a configuração de serviços do *cluster* impede que ele seja associado além do serviço que o utiliza. Um outro atributo de recursos diz respeito ao fato deste existir em todos os nós do *cluster* ou em todos os nós dos domínios de serviço que necessitam utilizar-se do mesmo. Quando isto é verdade, dizemos que o recurso tem valor não-verdadeiro para o atributo *Unique*. De outra forma: se um recurso tem o atributo *Unique* verdadeiro, significa que existe apenas uma única instância ou cópia do mesmo no *cluster*. Quando um recurso é *não-único*, significa que o serviço pode migrar para qualquer um dos nós do domínio do serviço e lá encontrar o recurso disponível para si, seja ele uma réplica ou um equivalente. Quando um recurso é *único*, significa que o serviço pode migrar para qualquer um dos nós do domínio do serviço, porém, para ter acesso ao recurso, ele provavelmente necessitará tomar algumas ações especiais de modo a ter visibilidade do recurso. Este é o caso de recursos que não existem replicados ao longo dos nós do *cluster*, tais como discos físicos acessados através de barramentos

de E/S compartilhados. Esses recursos literalmente podem *migrar* para mais do que um nó, porém existe apenas *uma* instância física do mesmo. O caso mais complexo em termos de atributos de recurso com certeza é quando o recurso tem como verdadeiros os atributos *Unique* e *Shared*. Em geral, apenas um grupo seletivo de recursos pode ter tais atributos.

Figura 3-14 Distribuição dos recursos do cluster aos serviços.

Exceto em casos especiais, onde os serviços utilizam esses recursos de modos específicos, tais como limitações nos modos de acesso e/ou coordenação de acesso, recursos únicos devem em geral ser marcados como não-compartilháveis.

Na definição de um serviço, os recursos que o compõem são especificados e uma *relação de dependência* é estabelecida. Essa relação de dependência serve para que a gerência de serviços saiba qual é a ordem mais adequada para se ativar os recursos. Na Figura 3-15 vemos uma relação de dependência entre os recursos que compõem um serviço típico. Neste exemplo, a primeira relação de dependência é aquela que define o domínio de nós para a execução do serviço. Em seguida, temos os primeiros níveis de dependência entre os recursos, onde dispositivos físicos tais como unidades de disco rígido e adaptadores de rede são especificadas. Neste serviço, portanto, esses são os primeiros recursos a serem verificados, testados e ativados no processo de ativação do serviço em questão. Havendo sucesso nessa primeira etapa, segue-se para a verificação e ativação dos recursos no próximo nível, os quais dependem dos anteriores. Neste exemplo, partições de disco rígido, possivelmente alocadas nos discos físicos ativados na etapa anterior, são localizadas, verificadas e tem seu acesso habilitado.

Figura 3-15 Relação de dependência entre os recursos de um serviço.

Trata-se da ativação de um conjunto de recursos lógicos, independentes. Outra categoria de recurso que normalmente é ativada nesta etapa é a dos recursos de endereço de rede lógico de adaptadores, tais como endereços IP. Em seguida, um outro nível de recursos é ativado, tais como subsistemas de *software*, serviços de *kernel* e outros. Dentre estes, podemos destacar: sistemas de arquivo, protocolos de rede, etc. Neste exemplo, temos que os recursos desse nível são dependentes dos demais ativados na etapa anterior. Por fim, uma aplicação, é ativada. No exemplo, trata-se de um servidor *web*, o qual necessita de acesso a alguns sistemas de arquivos hospedados em algumas partições de discos rígidos e também de alguns endereços de rede e adaptadores de comunicação para poder oferecer os seus serviços. Estando satisfeitas essas necessidades, o aplicativo pode executar e o serviço está pronto. A desativação de um serviço segue um processo similar, porém reverso, ou seja, primeiramente o aplicativo é terminado, os endereços de rede configurados são liberados dos adaptadores, os sistemas de arquivo são desmontados, as partições de disco tem seu acesso desabilitado e por fim os discos físicos são desalocados, assim como os adaptadores físicos de comunicação que foram alocados na etapa de ativação. Maiores detalhes sobre recursos serão abordados nos próximos tópicos.

3.3.5 Migração de Serviços

Como já vimos, serviços devem ter a capacidade de migrar para qualquer um dos nós do domínio do mesmo. Essa capacidade deve existir por diversos motivos, dentre eles:

- Possibilidade de migrar o serviço para um nó de reserva enquanto o nó de execução preferencial do serviço está em processo de manutenção ou de atualização;
- Migração para um nó com menor carga de trabalho e, portanto, mais eficiente na sua execução;
- Migração para um nó de reserva no caso de haver falha continuada do serviço no nó preferencial ou corrente do serviço;
- Migração para um nó de reserva no caso de haver falha completa do nó preferencial ou corrente do serviço.

Nos dois primeiros casos, diz-se que o serviço migrou por uma ação administrativa e por razões possivelmente também administrativas ou talvez técnicas. Nos dois últimos casos, diz-se que o serviço migrou forçosamente para um outro nó por motivo de falha. Neste caso, costuma-se denominar a ação migratória de *failover* (uma abreviação de *take-over-due-to-fail*, ou tomada do serviço devido a falha). Nos casos em que não houve motivação de falha para a migração, deve haver antes da tomada do serviço por um dos nós um processo de *liberação* (*release*) do serviço, levado a cabo pelo nó que correntemente o executa. Na Figura 3-16 vemos uma situação em que o nó executor de um serviço entra em um processo de falha (nó #4). Em última análise, essa falha do nó #4 deve desencadear um processo de *failover* do serviço 3 que nele está sendo executado. De acordo com a precedência de nós do serviço 3, o nó seguinte na seqüência de precedências é o nó número 3 (precedência=2) e assim este deverá ser o nó a tomar o serviço logo adiante. De um certo modo, o raciocínio é simples: o serviço 3 deverá ser ativado no nó #3. Não é necessário haver etapa de liberação, até porque o nó #4 falhou e não está mais operacional. Entretanto, algumas complicações existem por trás deste raciocínio simples.

Os maiores problemas do gerenciamento de serviços de um *cluster* estão nas etapas que envolvem a migração de um recurso de um nó para outro. Mesmo quando a migração ocorre por vias administrativas, tais quando o gerente do *cluster* emite um comando que solicita a migração de um serviço e todo o processo ocorre em uma seqüência lógica, em princípio poderiam haver problemas de conflito caso algumas ações especiais não fossem tomadas. Esses problemas podem ocorrer porque o gerenciador do *cluster* é um processo distribuído, e dessa forma suas partes devem estar em perfeita sincronia e coordenadas de modo a que nenhuma delas tome medidas ou execute ações que venham a conflitar com medidas ou ações tomadas por seus parceiros em outros nós.

Figura 3-16 Falha de um nó executor de serviço

Em situações como essa, é necessário que haja um serviço mediador entre as partes, que saiba realizar alguma espécie de decisão conjunta entre os envolvidos e interessados. De um certo modo, esse serviço deve ser capaz de realizar alguma espécie de *eleição* entre os interessados no serviço e eleger aquele que irá tomar e executar o serviço. Os interessados, no caso, são os nós do domínio do serviço 3 sobreviventes. Na Figura 3-17 vemos,

simplicadamente, o processo de votação em andamento. Ambos os interessados recebem uma solicitação para manifestarem-se sobre uma determinada proposta. A proposta, no caso, será a de que um determinado nó assuma o serviço. Caso o voto de todos os interessados for positivo, o mediador comunica a todos sobre o sucesso da eleição e a partir deste momento todos respeitam o resultado.

Figura 3-17 O processo de votação sobre o destino do serviço 3, cujo nó falhou (Figura 3-16)

O nó eleito assume o serviço e os demais não tentam assumi-lo. O mediador pode ser programado para tentar realizar a eleição um certo número de vezes, até que consiga o consenso. O mediador deve também estar ciente das políticas relativas a eleições naquele domínio, caso contrário pode manter propostas incompatíveis ou ilegais, que serão sistematicamente rejeitadas. Para evitar este tipo de problemas, em muitos sistemas o mediador é apenas um intermediário que faz o papel de levar a cabo a eleição, mas não faz ele mesmo as propostas. Um dos membros interessados é que deve, a partir das políticas vigentes, elaborar uma proposta de transição do serviço que seja compatível e facilmente aceita pelos demais membros. Desse modo, evita-se situações indesejáveis e talvez não-consensuais. Serviços especializados em realizar tarefas de intermediação de eleições baseadas em propostas são normalmente parte de um *serviço de comunicação entre grupos de processos*, também chamados de *Serviços de Comunicação em Grupo* ou simplesmente *Serviços de Grupo*. Esses serviços garantem a participação de todos os interessados nas votações de uma proposta ou, se for permitido, que algum critério de maioria seja definido. Eles garantem que apenas um processo de votação de proposta ocorra no contexto de um grupo a cada instante, de modo a que não haja conflitos entre propostas distintas. Caso duas eleições sejam solicitadas em um intervalo curto de tempo, uma delas é realizada por completo em primeiro lugar e só depois disto é que a segunda eleição é realizada. Os *Serviços de Grupo* fazem parte do conjunto de serviços essenciais de qualquer sistema de gerenciamento de *cluster*. Por essa razão, DSMCA prevê a existência de um serviço dessa categoria, o qual atende não apenas ao gerenciamento do *cluster*, mas tem interfaces abertas para que qualquer aplicação, distribuída ou não, possa se associar ao serviço e formar grupos de interesse, quer seja para realização de mera

comunicação (*broadcast* de mensagens de interesse do grupo), quer seja para coordenação e sincronização de suas partes e ações. Desse modo, certos serviços de *cluster*, tais como sistemas de arquivo distribuídos como os mencionados em (2.7), (2.8.2) e (2.8.3) podem se beneficiar da existência desses serviços pois não necessitariam implementar suas próprias soluções internamente.

Figura 3-18 O *cluster* após a eleição do nó 3.

3.3.6 Votação de Propostas

Como já mencionamos no tópico anterior, a gerência do *cluster* deve ser dotada de serviços que realizem eleições de propostas. As propostas são elaboradas por algum dos membros componentes da gerência do *cluster*. Esse membro pode solicitar a eleição da proposta por vários motivos. Dentre esses motivos, podemos citar:

- Uma ferramenta de gerenciamento do *cluster* está conectada àquele membro da gerência do *cluster* e solicitou, através de um comando administrativo, uma ação que envolve todos os demais membros.

- Aquela particular instância da gerência do *cluster* recebeu uma notificação de uma condição anormal no *cluster*, tal como a falha de um dos serviços que estão em execução no seu nó local.

Figura 3-19 Envio da proposta por um membro do grupo

Figura 3-20 Solicitação de apreciação da proposta

Em qualquer um desses casos, aquela instância da gerência do *cluster* irá solicitar ao serviço de comunicação em grupo que leve a cabo uma eleição sobre uma proposta por ela elaborada. Essa proposta pode ser, por

exemplo, de migração de um serviço de um nó para outro. Logicamente, esta proposta deve ser condizente com as políticas vigentes no *cluster* e naquele serviço.

Na Figura 3-19 vemos o momento em que o membro da gerência do *cluster* do nó #4 envia ao serviço de grupo a proposta para ser colocada em votação. Em seguida (Figura 3-20), o serviço de grupo solicita a cada um dos membros (inclusive o próprio membro que fez a proposta) que a avalie e manifeste seu voto a respeito. Com essa etapa, conclui-se a *primeira fase* do processo de votação. Na sequência, cada um dos membros manifesta sua opinião sobre a proposta enviando ao serviço de grupo o seu *voto* (Figura 3-21). Com isto, encerra-se a segunda fase do processo de eleição e aprovação/rejeição da proposta, quando então o resultado geral é comunicado a todos os membros (Figura 3-22).

Esse tipo de processo é comumente denominado de *two-phase-commit*, devido ao fato de que ele ocorre em duas fases e também porque, como já mencionado, caso alguma mudança de estado tenha sido causada pela apresentação da proposta, esta mudança é desfeita caso a proposta seja rejeitada ou se a eleição for cancelada.

3.3.7 Votação de Propostas no DSMCA

Daremos a seguir uma visão do processo de votação de propostas tal como ele é realizado no contexto do sistema DSMCA através do seu serviço de comunicação em grupo, ou simplesmente, *Serviço de Grupo*. Para tanto, utilizaremos como cliente-exemplo do serviço de grupo o *Cluster Manager*, o qual é o serviço DSMCA responsável por coordenar a distribuição e migração de serviços ao longo dos nós do *cluster*.

Figura 3-21 Envio do voto por cada membro do grupo

Figura 3-22 Comunicação do resultado

Na Figura 3-23 temos o desencadear de um processo de apreciação de proposta, iniciado pelo envio do nó #4 de uma solicitação de votação ao Serviço de Grupo.

Figura 3-23 Proposta feita pelo nó #4, contendo 2 itens, em 2 fases.

No sistema DSMCA, um cliente do serviço de grupo faz parte de um grupo pré-estabelecido, do qual fazem parte todos os seus interessados. No caso, o grupo em questão é um *grupo de serviço* denominado [*grupo de serviço.2*] (pois é um grupo de membros que lidam com serviços, do *cluster*, especificamente o serviço número 2), do qual fazem parte todos os nós que constituem o domínio de nós do serviço. A rigor, os componentes desse grupo são todos os *Cluster Managers* dos respectivos nós constituintes desse domínio de serviço. O serviço de grupo, ao receber a solicitação do membro do nó #4:

- a) Gera uma identificação única para a proposta (no exemplo, 4.35, que pode significar que o membro #4 enviou sua 35ª proposta), e
- b) determina o número de itens que constitui a proposta (2).

O DSMCA considera cada item da proposta como uma proposição individual. A proposta foi especificada como sendo em duas fases,

portanto cada proposição será votada em duas fases. Inicialmente, o serviço de grupo envia o primeiro item da proposta para os três membros do grupo em questão, que são as gerências de *cluster* nos nós 2, 3 e 4 (Figura 3-24). A seguir, dá-se início à votação por cada um dos membros. Os membros analisam o item da proposta e posteriormente retornam um voto a respeito dele.

Figura 3-24 Recebimento do item 1 da proposta

Nas seqüências da figura Figura 3-25 podemos acompanhar o desenrolar da votação da proposta de dois itens. No final, nas figuras (h) e (i), temos uma seqüência final alternativa, que mostra o que poderia ocorrer caso um dos membros rejeitasse um dos itens da proposta. Deve ocorrer, nesses casos, um retorno ao estado original do serviço, num processo denominado *rollback*. A semântica da votação é portanto transacional e atômica, isto é, caso a proposta seja rejeitada, o estado final é como se a votação nunca tivesse ocorrido. Sobre o retorno posterior do voto por cada um dos membros, cabem aqui algumas considerações. Nesse tipo de serviço, onde

ocorrem diversas interações entre vários objetos distintos, é altamente desejável que o modelo de comunicação utilizado não obrigue o retorno de um valor imediatamente após o envio de uma mensagem. A razão disto está no fato de que o serviço de grupos, que coordena e realiza a eleição, pode ficar a cargo de se comunicar com um grande número de membros que possivelmente apresentam tempos de resposta diversos. Realizar o envio das propostas e receber os votos serialmente pode ser, portanto, muito demorado, além do que a chamada obriga o membro a enviar uma resposta imediata. Um problema aqui seria o de lidar com membros sobrecarregados, lentos ou em processo de falha. Uma alternativa de eficiência e flexibilidade é utilizar algum serviço de mensagem do tipo *datagrama*. Quando se pensa na implementação de tal serviço utilizando objetos CORBA e interfaces IDL, podemos imaginar a especificação de métodos de objeto sem valor de retorno (*void*) e com chamada unidirecional (*oneway*). Porém, segundo alguns especialistas em implementações de CORBA¹³, o uso desta diretiva e do tipo *void* IDL para os métodos não é garantia de que o ORB apresentará uma semântica de *datagrama* para as invocações de métodos de objeto. Uma das razões, segundo se alega é que o protocolo inter-ORB (IOP) utilizado como padrão em CORBA, o IIOP, é um protocolo orientado a conexão e com reconhecimento e que utiliza o transporte do protocolo TCP, o que o descaracteriza como um protocolo de *datagrama*.

¹³ Consulte [VINOSKI1999] no capítulo 4, seção 4.12. e [ANSACOM95] em 3.1.2.7.

Figura 3-25 Votação da proposta de dois itens

De acordo com [VINOSKI1999] - seção 4.2,

“Não elabore projetos que assumam um comportamento não-bloqueante ou assíncrono das chamadas aos métodos apenas pelo uso da declaração oneway para as operações. O comportamento real em run-time de tais chamadas depende do ORB em questão e, tipicamente, depende também de questões de concorrência e de localidade do cliente e do servidor”.

Também, segundo [ANSACOM95] – seção 3.1.2.7,

“Um chamada oneway existe para prover um estilo de interação baseado em datagramas não-confiáveis, tal como aquele oferecido pelo protocolo UDP em redes IP. Uma vez que o datagrama tenha sido composto e enviado pela rede, o enviante não possui mais controle algum sobre o processo de entrega, nem confirmação de sua entrega. Caso o destinatário estiver recebendo uma tempestade de chamadas oneway de muitas fontes distintas,, é provável que ele esgotará seus recursos para processá-las e terá que descartar algumas delas. Por esta razão, a semântica da chamada oneway é denominada de best-effort (melhor esforço, não uma promessa)”.

Do primeiro comentário, conclui-se que é prudente ter informações precisas sobre o comportamento de um ORB antes de apostar nessa solução. Do segundo, percebe-se que as semânticas de *oneway* são aquelas que podem trazer flexibilidade na implementação do serviço de grupos em discussão. Tem-se, portanto, que a simples utilização da diretiva *oneway* pode ser não significativa caso o protocolo de transporte do ORB não for de *datagrama*. Atualmente já existem implementações de CORBA que permitem a utilização de protocolos de IOP baseados em *datagrama*¹⁴ tais como o UDP, os quais complementam a semântica de *oneway* e desse

¹⁴ A implementação de[ORBacus] é uma delas, e será discutida no capítulo de comparações.

modo podem viabilizar a existência de um serviço de grupo eficiente no sistema DSMCA ao apoiar-se nessa solução. Ao se utilizar um protocolo de *datagrama*, o próprio objeto que implementa o serviço deve tratar o não-retorno de respostas. Isto porque o não-retorno de votos pelos membros de um grupo pode ser considerado um fato normal em certos casos. Uma outra alternativa seria a de se utilizar algum serviço específico para o envio de mensagens através de *datagramas*. O serviço de grupo não teria a incumbência de cuidar da comunicação entre seus membros, mas lançaria mão de um serviço de mensagens baseado em *datagramas*, o qual agrega alguns elementos de confiabilidade. Como comentário final deste capítulo, adianta-se que a arquitetura DSMCA apóia-se nesta segunda alternativa.

DSMCA

A Distributed System for Support and Management of Clusters And Applications

4 DSMCA

O DSMCA - *Distributed System for Support and Management of Clusters and Applications* – ou Sistema Distribuído para o Suporte e Gerenciamento de Clusters e Aplicações, consiste de um conjunto de programas baseados em classes de objetos C++ e objetos CORBA cuja finalidade é prover um ambiente eficiente, fácil e flexível para o gerenciamento de sistemas em *cluster* e também como plataforma de desenvolvimento para aplicações de *cluster*. O sistema DSMCA é composto por diversos serviços, gerências e monitores interconectados por um ORB CORBA, cujas interfaces são especificadas em IDL – *Interface*

Definition Language, e são documentadas em HTML – *Hypertext Markup Language*. Na Figura 4-1 podemos ver um esboço geral do sistema e seus principais componentes. Dentre eles, podemos destacar:

- *Cluster Manager*- a gerência de *clusters* e serviços de *cluster*;
- *Cluster Group Service* – um serviço de comunicação em grupo;
- *Cluster Message Service* – um serviço de datagrama confiável;
- *Cluster Node Monitor* – um serviço de monitoria de nós;
- *Cluster Adapter Monitor* – monitoria de adaptadores de comunicação;
- *Service Manager* – o gerente local de serviços de um nó;
- *Resource Manager* – o gerente local de recursos de um nó;
- *Resource Action Selector* – um serviço de correlação entre eventos e ações;
- *Resource Action Executor* – um serviço de execução de ações;
- *Resource Locator & Activator* – o serviço local de ativação e localização de recursos e gerenciadores de recurso;
- *Resource Event Selector* – um serviço de coleta e distribuição seletiva de eventos diversos do sistema;
- *Resource Manager* – implementador e/ou gerenciador de um ou mais recursos dos serviços;

- *Resource DB Manager* – o serviço de armazenamento e distribuição de dados diversos do sistema.

Na Figura 4-1, os componentes do sistema são representados pelas figuras circulares. Não representam exatamente programas, mas sim certas áreas, funções e/ou serviços do sistema, sendo que cada qual pode ser composto por mais de um programa, por diversos objetos C++ e CORBA e cujos serviços podem ser acessíveis e controláveis por diversas interfaces IDL. As setas que unem os diversos elementos representam fluxos de mensagens, solicitações, notificações, eventos e informações. Em geral, as setas com sentido para baixo representam solicitações, enquanto que as setas com sentido para cima representam em geral fluxos de informações, eventos e notificações. Juntamente com cada seta há uma legenda com a indicação da espécie de fluxo que ela representa.

A seguir faremos uma breve explanação do funcionamento geral do sistema e outra para cada elemento da arquitetura do DSMCA.

Figura 4-1 Arquitetura do DSMCA

4.1 DSMCA – Visão Geral

O sistema DSMCA visa dar suporte para o gerenciamento de *clusters* de servidores e atuar como *middleware* de suporte das aplicações que executam em *clusters*, permitindo a estas terem características de alta disponibilidade e grande desempenho. DSMCA é composto de diversos módulos de serviço implementados como serviços CORBA e disponíveis através de interfaces CORBA IDL e visa essencialmente tirar proveito da arquitetura de *hardware* dos *clusters* tais como a presença de módulos de comunicação, de armazenamento, de recursos e processamento independentes, concorrentes e redundantes, alcançando com isso percentuais de *downtime* mínimos e elevadas taxas de transações por segundo. Diversos tipos e classes de aplicações podem ser construídas na plataforma DSMCA, mas dentre elas destacam-se as aplicações servidoras. Servidores *web*, servidores de arquivo e servidores de banco de dados estão entre as aplicações possíveis e indicadas para o sistema, sendo que todas elas podem ser construídas como aplicações distribuídas dotadas de características de alta disponibilidade e desempenho, uma vez que as características do *hardware* do *cluster* e as características funcionais de DSMCA podem ser somadas. Uma outra classe de aplicações possível e necessária seria a das ferramentas de administração de *clusters*, as quais constituiriam um grupo de ferramentas de gerenciamento para o próprio sistema DSMCA. Essas ferramentas administrativas poderiam ser baseadas em uma interface gráfica (GUI) e interagiriam com o DSMCA através de uma conexão ao serviço de gerenciamento de *clusters* (*Cluster Manager*) e a outros de seus serviços em algum nó qualquer do *cluster*. A interface do serviço *Cluster Manager* permite o gerenciamento de *clusters* através de ações de monitoria, configuração e controle.

4.1.1 Aplicações Distribuídas em DSMCA

O sistema DSMCA possibilita que as aplicações apresentem características desejáveis na execução distribuída de suas tarefas, tais como a consistência de estado, coordenação na execução de tarefas e serialização de procedimentos críticos. DSMCA provê para as aplicações um serviço de comunicação e coordenação de grupos de processos de aplicações distribuídas - o *Cluster Group Service* – o qual apresenta aos seus clientes uma interface de comunicação simples baseada em ações de junção ao grupo e na chamada de métodos para envio de propostas e recebimento de resultados.

Figura 4-2 Modelo de interconexão bidirecional de objetos CORBA utilizado por DSMCA

O serviço do *Cluster Group Service*, como a maioria dos serviços de DSMCA, é baseado em interfaces de solicitação do usuário e em interfaces de *callback* do serviço, constituindo um canal bidirecional de comunicação entre os objetos provedores do serviço e os objetos clientes do serviço. Tal modelo de interconexão de objetos e interfaces CORBA está ilustrado na Figura 4-2. O serviço de comunicação em grupo possibilita que grupos de processos compondo uma aplicação distribuída

possam manter os seus dados de estado consistentes entre todas as instâncias da aplicação, as quais operam em espaços de endereçamento totalmente separados (a característica *shared nothing* comum dos *clusters* da classe *Beowulf*.)

Processos ou objetos utilizam o serviço de grupo como um mecanismo de coordenação para modificações do seu estado. Para tanto, eles submetem ao grupo *propostas* de alteração de estado e outros tipos de propostas, as quais são atômicamente distribuídas a todos os membros do grupo (todas as instâncias dos processos ou objetos da aplicação) através de um canal de comunicação confiável do DSMCA implementado pelo *Cluster Message Service*. Cada membro retorna votos contendo um valor de aceite ou rejeição dos itens da proposta. O serviço faz a apuração e, ao final, caso a proposta tenha sido aprovada, os membros são convocados a tomar a ação cabível naquela situação particular (*commit*), ou voltar atrás em todas as ações previamente tomadas durante as fases de votação da proposta (*rollback*). Com isso, o serviço permite que a aplicação apresente atômica na alteração de seu estado e na propagação de outros tipos de mensagens significativas para os processos da aplicação.

4.1.2 Comunicação Confiável em DSMCA

Para que o serviço de grupos possa operar com confiabilidade, ele utiliza o serviço de comunicação confiável do *cluster* DSMCA – o *Cluster Message Service*. Trata-se de um serviço de mensagens (datagramas) confiável baseado em chamadas de interface CORBA IDL através de um ORB CORBA operando com protocolo de comunicação de datagramas UDP/IOP. O serviço deve ser capaz de transmitir qualquer tipo de mensagem ou dado definido pelo cliente através da linguagem IDL, seja ele um valor numérico escalar, uma *string* de caracteres ou um tipo de dado definido pelo objeto-cliente, tais como estruturas, uniões e seqüências. Para atingir tal objetivo o serviço define o tipo de dado

Datagram como a sua unidade de dados de serviço, o qual por sua vez é baseado no tipo CORBA::Any. Para garantir sua confiabilidade, o serviço define internamente chamadas na interface do serviço que visam a confirmação (*acknowledgement*) de recebimento e de entrega da mensagem ao cliente, além de verificações de possibilidade de entrega baseada na topologia atual da rede e no estado de sobrevivência das interfaces (adaptadores) físicos de comunicação dos nós do *cluster*. Desse modo, o serviço de mensagens tem previamente boas indicações de que o nó destinatário está operacional, se há um caminho de comunicação possível e se este possui o cliente-destinatário ativo antes de tentar o envio da mensagem. O serviço pode, a pedido do cliente, operar em modo não-confiável, atuando portanto como um serviço simples de datagrama sem confirmação (*unacknowledged*). Para conseguir atingir tais objetivos, o *Cluster Message Service* constrói internamente um mapa topológico das redes ativas do *cluster*, listas de estado dos nós e dos objetos-clientes ativos em cada nó. Tais informações são construídas a partir de outras informações obtidas do serviço de monitoria de nós do *cluster* DSMCA – o *Cluster Node Monitor*. Este serviço fornece aos seus clientes listas de nós do *cluster* que apresentam indícios seguros de que estão operando com capacidades de intercomunicação entre si. Estas listas são chamadas de *NodeMembership*. Um *NodeMembership* compõe-se de um grupo de nós identificados pelos seus *Id*'s (número do nó no contexto de um *cluster*), a identificação da rede onde seus adaptadores estão operacionais (o *NetworkId* da rede) e a identificação do líder do grupo (*leader*).

O conjunto de *NodeMemberships* reflete a situação topológica e operacional dos nós de um *cluster* e de suas redes de comunicação em um dado instante. Eles permitem calcular caminhos de comunicação entre quaisquer nós do *cluster* desde que estes pertençam a *NodeMemberships* com conectividade topológica entre si. O serviço de mensagens opera então com o conceito de encaminhamento de mensagens (*forwarding*) por

nós intermediários. No serviço *Cluster Message Service*, nós intermediários são aqueles onde o serviço de mensagens está preparado para passar adiante uma mensagem recebida e que não é destinada ao nó onde reside. Ele faz isto baseado na sua visão topológica da rede e no melhor caminho até o destino final, podendo inclusive ser através de um novo intermediário. O conceito é similar ao roteamento de pacotes realizado pelo protocolo IP – *Internet Protocol*, porém mais complexo e permite operar em redes IP particionadas – situações nas quais o roteamento IP puro não seria capaz de realizar a entrega ao destino. Na Figura 4-3 vemos uma hipotética situação em que ocorreram particionamentos nas redes 1 e 2 de um *cluster* de cinco nós. O roteamento IP não teria capacidade de prover comunicação continuada entre os nós 1 e 4. Porém, utilizando o serviço de mensagens do *cluster* é possível a comunicação entre quaisquer serviços DSMCA presentes nos nós, haja visto que o serviço de mensagens do nó 3 perfaz o papel de encaminhador entre os segmentos particionados das redes. Os nós com capacidade de encaminhamento são sempre líderes de algum *NodeMembership* pois são eles que anunciam aos demais líderes sobre o seu grupo de nós.

4.1.3 Monitoria de Estado de Nós em DSMCA

O serviço de monitoria de nós de DSMCA – o *Cluster Node Monitor* – opera a partir de informações obtidas a partir de um serviço de nível mais baixo DSMCA denominado *Cluster Adapter Monitor* – o serviço de monitoria de adaptadores de comunicação do *cluster*.

Figura 4-3 Comunicação entre nós via Cluster Message Service utilizando nós encaminhadores intermediários em redes particionadas.

Esse serviço é especializado nas características de comunicação dos adaptadores presentes, visando estabelecer entre eles os chamados *anéis de heartbeat* - ou *heartbeat rings*. Trata-se de um mecanismo de duas finalidades, a saber: (1) estabelecer um mecanismo de monitoria de estado operacional dos adaptadores de comunicação dos nós do *cluster* e das redes de comunicação aos quais estes conectam-se e (2) criar um serviço de informações que forneça as identificações dos adaptadores operacionais em uma mesma rede de modo contínuo. Esses dois mecanismos fornecem, em conjunto, subsídios para que o serviço de monitoria de nós possa realizar seus serviços, identificando os conjuntos de nós operacionais do *cluster* e quais os caminhos de comunicação possíveis. Na Figura 4-4 vemos um anel lógico de heartbeat operando em uma das redes de um *cluster* onde cinco nós tem seus adaptadores operacionais. O identificador de um adaptador de comunicação é composto pelo identificador da rede na qual o adaptador se conecta seguido do identificador do nó do *cluster*. O conjunto de identificadores de adaptadores operacionais em uma rede, mais o identificador dessa rede, juntamente com a identificação do líder desse conjunto de adaptadores é denominado *AdapterMembership*. Um

AdapterMembership é um grupo de adaptadores de comunicação conectados a uma rede do cluster e que se encontram em estado operacional normal. O líder do *AdapterMembership* é o adaptador do grupo cujo identificador numérico é o de valor mais alto. Ele tem a função de coordenar o grupo e criar as listas de *AdapterMembership*. O serviço de monitoria de adaptadores opera como um serviço distribuído coordenado utilizando um mecanismo simples de consenso e votação para manter a consistência do estado do grupo, e para tanto utiliza um protocolo simples baseado em chamadas de uma interface de serviço CORBA IDL dos objetos monitores. Esse protocolo opera em um ambiente onde as falhas de comunicação são esperadas. Desse modo, o protocolo de transporte de chamadas de objetos CORBA utilizado é o protocolo de datagrama mais simples, o IOP/UDP, que não provê confirmação e não possibilita retornos na chamada. Essas características tornam as chamadas de métodos de interface do serviço não bloqueantes, mas que necessitam ser monitoradas quanto a problemas de comunicação através de temporizadores e de mensagens de reconhecimento de entrega de modo a haver a detecção segura de problemas de comunicação entre os adaptadores. Os objetos monitores trocam continuamente entre si mensagens denominadas *heartbeat*. A mensagem *heartbeat* é recebida quando um membro invoca o método *heartbeat()* na interface do seu nó monitor. Ela visa dar ao nó seguinte (o nó monitor) uma indicação do estado operacional do membro enviante (o nó monitorado). A lógica interna do serviço de monitoria de adaptadores é elaborada de tal modo que um anel lógico entre os adaptadores é estabelecido. Um nó envia mensagens de *heartbeat* apenas para o nó seguinte a ele no anel, de modo a manter o volume de mensagens na rede a níveis baixos e aceitáveis.

Figura 4-4 Um *heartbeat ring* de adaptadores composto de cinco nós

Desse modo, todo nó é monitor do nó que o precede. Existe um limite de tempo máximo em que um nó pode ficar sem enviar mensagens de *heartbeat* ao seu monitor. Quando um nó excede esse tempo, o seu nó monitor envia ao líder do *AdapterManager* uma notificação da possível morte do adaptador do nó monitorado. O líder deve verificar tal evento e, se for o caso, reconfigurar o anel através de uma transação atômica utilizando primitivas *prepare_to_commit* seguida de *commit* ou *abort*, entre todos os membros sobreviventes. Na Figura 4-5 podemos ver a infraestrutura completa de serviços de comunicação de *cluster* do sistema DSMCA, a qual oferece serviços para os demais módulos de DSMCA e para quaisquer outros serviços e objetos interessados. Os serviços são:

comunicação em grupo com semânticas diversas, comunicação confiável e robusta, confirmada ou não.

4.1.4 Conexão aos Serviços de Comunicação

Objetos-cliente interessados no serviço de mensagens devem obter uma referência à administração do serviço e solicitar um objeto e interface *proxy* do serviço, de modo a poder conectar-se ao serviço especificando uma referência à sua interface de *callback* durante a chamada do método *connect()*. Da mesma forma, os objetos-cliente interessados no serviço de comunicação em grupo devem obter referências à interface de administração do serviço de grupos e solicitar um objeto e interface de administração do grupo desejado. Após isso, o cliente deve solicitar um objeto *proxy* do grupo e em seguida conectar a sua interface de *callback* ao mesmo através da chamada do método *connect()*. No serviço de grupos de DSMCA, um grupo é criado quando um objeto de administração de grupo é solicitado e criado pela primeira vez em qualquer um dos nós do *cluster*. O nome de um grupo é um identificador global e único no contexto de um *cluster*. A partir do momento em que um grupo é criado, qualquer outra solicitação a um objeto de administração desse grupo, em qualquer outro nó, efetivamente estará afetando o grupo já existente. Assim, uma solicitação para junção ao grupo feita por um objeto em um outro nó do *cluster* ocasionará a junção do solicitante aos membros já existentes nesse grupo, presentes em quaisquer outros nós do *cluster*. Sempre que um novo membro solicita junção ao grupo (*join*) ou um membro solicita deixar o grupo (*leave*) o serviço notifica a todos os demais membros existentes sobre o evento e aguarda o parecer de cada um. Se o parecer for favorável pela maioria, o membro é aceito e ocorre uma notificação de alteração no *membership* do grupo a todos os membros.

Figura 4-5 A infra-estrutura de serviços de comunicação do sistema DSMCA

O serviço de grupos apresenta a característica de unir objetos em um grupo global e distribuído, possibilitando comunicação coordenada e sincronizada através do envio de propostas. Propostas podem ser vistas como mensagens simples ou solicitações diversas, apresentando também a

característica de sincronia virtual devido ao fato de que sempre é o nó líder do grupo que encaminha e recebe as respostas de todas as mensagens para os membros do grupo, fazendo isso de modo serializado. Na Figura 4-6 podemos observar a situação de dois grupos de objetos existentes ao longo de dois dos nós de um *cluster* DSMCA. Os grupos são identificados pelos números 1 (*Grupo 1*) e 2 (*Grupo 2*). Os dois nós exibidos são os identificados pelos números 3 (*Node 3*) e 4 (*Node 4*). O *Grupo 1* possui dois membros, um deles residente no nó 3 e o outro no nó 4. O *Grupo 2* possui três membros, dois deles residentes no nó 3 e um deles no nó 4.

Para os clientes do serviço de grupo, ou seja, os membros dos grupos, a distribuição espacial dos membros do grupo ao longo dos nós do *cluster* é transparente, ficando a cargo do serviço de grupos realizar a comunicação de propostas e o recebimento de resultados entre todos os membros em todos os nós. Para a comunicação de mensagens entre os objetos distribuídos do serviço de grupos, utiliza-se o serviço de mensagens confiável que opera com as informações de conectividade e de estado operacional de nós e adaptadores fornecida em tempo real pelos serviços de monitoria de nós e de adaptadores. O serviço guarda em todas as suas instâncias uma relação dos grupos e dos membros existentes, relação essa que constitui o estado do serviço de grupos.

4.1.5 Exemplo de Aplicação Distribuída

Na Figura 4-7 podemos observar o desenho de uma aplicação distribuída ao longo de três nós do *cluster* operando com o suporte dos serviços de grupo e comunicação de DSMCA. Trata-se de um hipotético servidor de banco de dados no qual o armazenamento encontra-se distribuído pelos discos dos três nós, podendo haver replicação dos dados como forma de se aumentar a confiabilidade e disponibilidade dos dados e de se obter distribuição espacial como forma de se agilizar e melhorar os tempos de resposta de acesso aos dados.

Figura 4-6 Grupos de objetos coordenados pelo Serviço de Comunicação em Grupo de DSMCA, compondo possíveis aplicações distribuídas.

Porém, tais configurações de um banco de dados exigem que as suas instâncias (processos ou objetos) estejam em perfeita sincronia e que realizem as ações de atualização ou modificação do banco de dados de modo coordenado e com semânticas transacionais. Isto equivale a dizer que o estado do sistema distribuído deve ser sempre consistente. Nestes casos, DSMCA oferece o suporte necessário para atingir esses objetivos. O serviço de grupo não somente oferece serviços de votação e consenso para que a aplicação realize alterações no seu estado de modo transacional e atômico como também oferece benefícios tais como manter todas as instâncias da aplicação sempre cientes do número e identificação das demais instâncias ativas e operacionais. O serviço de grupo notifica a todas as partes da aplicação a respeito da chegada e partida de novos membros (instâncias da aplicação) de modo que elas possam se

reorganizar na realização da tarefa que executam. Por exemplo, o banco de dados pode inicialmente estar operando com somente uma instância de processamento em um dos nós, o qual responde por 100% das solicitações dos clientes do banco de dados. Quando um segundo nó é ativado e uma segunda instância é ativada no mesmo, ela perfaz localmente uma chamada *join* para o grupo do banco de dados, cujo serviço já conhece a existência da primeira instância no nó inicial. Ambas recebem a notificação de que o grupo é composto de dois processos e assim elas podem se (re)configurar de um modo pré-estabelecido pela lógica de distribuição de carga de trabalho do banco de dados. Assim, por exemplo, cada instância pode assumir 50% da carga de trabalho do banco, algo como cada uma delas processar 50% de cada *query* às tabelas. Quando o terceiro nó é ativado, ocorre novamente a notificação pelo serviço de grupo da chegada do novo membro e então as instâncias de processo do banco reconfiguram-se novamente de modo a que cada uma delas responda por 1/3 da carga de trabalho de cada *query* solicitada. Não estamos aqui tecendo maiores comentários a respeito da sincronia de dados entre as várias instâncias no momento da junção de novos membros, mas assumimos que o serviço de grupo está também oferecendo meios para que uma nova instância atualize seus dados até que estes estejam perfeitamente sincronizados com a(s) instância(s) ativas. Em casos assim, as instâncias antigas não dividem sua carga de trabalho com a nova instância até que esta operação seja completada.

Figura 4-7 Um servidor de banco de dados distribuído operando em três nós com o suporte dos serviços de DSMCA

Quando, por algum motivo qualquer uma das instâncias do banco de dados é desativada (por exemplo, por uma ação gerencial local no nó onde ela reside), o serviço de grupo naquele nó recebe a solicitação da instância para deixar (*leave*) o grupo. O serviço de grupos notifica a todas as demais sobre esse evento, de modo a que eles tomem as ações necessárias condizentes com a situação que está por vir. Os dois nós restantes podem decidir nesse momento sobre uma nova configuração de distribuição de carga de trabalho tal como ela era antes, ou seja, 50% da carga para cada um deles. Um outro detalhe a respeito da chegada e partida de membros é que essas etapas são sempre passíveis de votação pelos membros atuais, ou seja, eles são notificados desses eventos e podem votar a respeito da entrada e partida de membros do grupo. Assim, a entrada de um novo

membro pode ser vetada durante um certo tempo caso os demais membros assim decidirem, tal como, por exemplo, em uma situação em que a solicitação de entrada do novo membro está ocorrendo em um momento inadequado ou inoportuno. Também, um membro pode ter sua saída retardada caso os demais membros necessitem de mais tempo antes de permitir sua partida. Esse caso ilustra uma partida condicional de um membro, onde ele próprio solicita e acata a decisão da maioria dos membros. Porém, o membro pode solicitar partida e desligar-se do grupo incondicionalmente. É o caso, por exemplo, quando o membro recebe um pedido de *shutdown* da administração local do nó onde se encontra, e ele deve terminar suas atividades incondicionalmente o mais rapidamente possível. Uma outra questão ligada à partida de membros de um grupo diz respeito a problemas que podem ocorrer no processo ou objeto que implementa um membro e que podem leva-lo a um término anormal. Nessas condições, o membro provavelmente não terá tempo ou condições de notificar o serviço de grupos sobre sua partida e dessa forma é o serviço de grupo que deverá detectar a partida por falha do membro em questão. A interconexão de objetos CORBA prevê situações como essa, de modo que o código do serviço de grupos perceberá o erguimento de uma exceção CORBA específica ao tentar utilizar sua referência ao objeto que implementa a interface de *callback* do membro que terminou anormalmente, pois esta terá tornado-se inválida devido à destruição do objeto. Uma outra situação mais complexa é aquela em que o código de um membro torna-se não-responsivo, provavelmente devido a condições excepcionais tais como *bugs* e *deadlocks*. Nesses casos, o objeto que implementa a interface de *callback* do membro com problemas mantém-se e sua referência permanece válida. Nesses casos, a interconexão de objetos CORBA não tem condições de detectar essa situação. É possível instruir uma invocação de um método de objeto CORBA para que ela erga uma exceção por *timeout* caso o tempo para a chegada da resposta exceda um certo valor. Porém a detecção de problemas por *timeout* durante uma

rodada de negociação entre todos os membros de um grupo pode ser altamente contraproducente. Sendo assim, o serviço de grupos de DSMCA exige que as interfaces de *callback* dos membros do serviço implementem o método *heartbeat()*, o qual serve para que o serviço verifique periodicamente o estado de responsividade de todos os seus membros e possa ter de antemão informações mais precisas sobre os seus estados. Em qualquer caso, sempre que um membro apresentar-se não responsivo ou que a referência à sua interface de *callback* não for mais válida, o serviço de grupo notifica esse evento a todos os membros restantes através de uma mensagem de falha de membro *member_failure_detected()*. Os membros devem agir de modo similar ao caso de uma saída voluntária do membro, respondendo ao serviço. Em todos os casos envolvendo chegada e partida de membros, o serviço de grupos faz uma notificação final aos membros sobre a nova composição do grupo (o *membership*) através de uma chamada de evento *membership_change()*. Notemos que uma mudança na composição do grupo serve para que o próprio grupo inicie um processo interno de mudança de estado através de uma solicitação de alteração dos dados de estado interno do grupo ou pelo envio de propostas e, portanto, um dos membros deve ter a incumbência de tomar essa iniciativa. A lógica interna de funcionamento do grupo nos casos de mudança no *membership* é dependente portanto de sua implementação.

4.1.6 Inter-relação entre Monitoria de Nós e Grupos

Observando novamente a Figura 4-6 podemos notar que existe uma inter-relação entre o serviço de grupos (*Cluster Group Services*) e o serviço de monitoria de nós (*Cluster Node Monitor*). O serviço de grupos na verdade é cliente do serviço de monitoria de nós e utiliza as informações de *NodeMembership* que obtém deste para criar um grupo interno ao *cluster* denominado *NodeMembership*. Esse grupo mantém como valor interno de estado uma lista de identificadores de nós do cluster (*ClusterNodeList*) que

encontram-se operacionais e com conectividade lógica para comunicação. Sempre que o serviço de monitoria de nós notifica uma alteração em um *NodeMembership*, o serviço de grupos no nó que recebeu a notificação reconstrói a lista de nós operacionais do *cluster* em questão e submete o novo valor como proposta para votação por todos os membros, de modo a haver consenso sobre a mudança. Caso haja consenso, uma notificação sobre a aceitação da proposta é enviada e assim a visão de *NodeMembership* é consistente por todo o *cluster*. Os processos ou objetos interessados em obter informações sobre nós através desse grupo interno normalmente fazem apenas uma subscrição ao grupo, de modo a terem apenas notificações sobre mudanças consensuais. Caso haja tentativa de junção como membro, o serviço pode ou não aceitar tal tentativa uma vez que desse modo o membro participará das votações do grupo. A decisão de aceitar ou não membros externos para esses grupos internos é dependente da implementação do serviço. A inter-relação entre o serviço de monitoria de nós e o serviço de grupos é importante porque ela permite que o serviço de grupos possa tomar as ações necessárias para notificar grupos que perderam membros devido à falha completa de um nó. O serviço notifica os membros sobreviventes dos grupos afetados para que se manifestem sobre a saída por falha dos membros que naquele nó residiam e em seguida notifica as alterações sofridas no *membership* do grupo.

4.1.7 Clientes de uma Aplicação Distribuída

De volta ao exemplo da Figura 4-7, os clientes da aplicação servidora do banco de dados podem ou não ter ciência da distribuição espacial e de carga de trabalho das instâncias do banco de dados. Se eles devem ter essa ciência, então os clientes precisam possuir código ciente do estado do banco e de sua configuração atual, o que equivale dizer que o código-cliente deve realizar junção ao grupo de banco de dados e participar, ao menos como um *Subscriber*, das decisões de reconfiguração de modo a

conhecer a distribuição da carga de trabalho no banco. Esse esquema é interessante uma vez que delega ao cliente uma parte da função de distribuição de carga de trabalho. Se os clientes não devem ter ciência da distribuição espacial do banco, o seu código é mais simples e não necessita portanto ser membro ou *Subscriber* do grupo. Ele necessita conhecer apenas alguns parâmetros para conexão ao banco, tais como endereço IP e porta TCP do nó que atende os pedidos dos clientes. Este nó servidor realiza a distribuição de tarefa entre os demais nós servidores do banco, coleta os resultados e envia o resultado completo ao cliente. É comum que qualquer um dos nós servidores possa realizar a função de atender os clientes, e nesses casos pode haver distribuição de carga de trabalho entre eles baseada em um equipamento de redireção de endereços IP (tais como o LVS¹⁵) ou simplesmente através de distribuição rotativa de endereços IP pelo sistema de resolução de nomes, o que pode ser feita por um RR-DNS¹⁶.

4.1.8 Recursos e Serviços de *Cluster*

O sistema DSMCA possui algumas gerências cujos serviços são voltados à administração de *Serviços de Cluster* e dos recursos necessários para a execução destes. Um Serviço de Cluster é uma tarefa a ser executada pelo *cluster* e que atende a alguma necessidade dos computadores e sistemas clientes do *cluster*. Como exemplo de serviços de *cluster* podemos citar: serviço de dados, serviço de arquivos, serviços Web, serviços Internet, dentre outros. Um Serviço de Cluster é ativado e está pronto para atender pedidos de seus clientes no momento em que todos os seus recursos estão prontos e ativados. Os recursos de um Serviço de Cluster podem ser somente aqueles denominados como Recursos de Cluster, que são os registrados como de propriedade de um *cluster*. Porém, uma vez que um

¹⁵ Linux Virtual Server – maiores detalhes na seção 5.1

¹⁶ RR – DNS, ou *Round-Robin DNS* – consulte o RFC 1794 em <http://www.ietf.org/rfc/rfc1794.txt>

Recurso de Cluster está atribuído a um serviço, ele passa a ser chamado Recurso de Serviço.

4.1.8.1 Recursos de Cluster

Para o gerenciamento dos Recursos de Cluster o sistema DSMCA conta com algumas áreas administrativas distintas, a saber: a *Gerência de Recursos*, o *Ativador e Localizador de Recursos* e o *Gerenciador de Recursos*. A Gerência de Recursos é um serviço do sistema DSMCA que pode receber pedidos para iniciar, habilitar, desabilitar e terminar o uso de um certo recurso de serviço com a finalidade de prepara-lo para que um certo Serviço de Cluster possa entrar em operação naquele nó do *cluster*. Uma outra característica da Gerência de Recursos é ser um serviço local de cada nó do *cluster* e não é um serviço distribuído. Ela preocupa-se apenas com as questões locais dos recursos e assume que questões relativas a recursos distribuídos estão sendo tratadas em outros serviços. Para que recursos sejam gerenciados por essa gerência, o cliente de seus serviços necessita possuir um identificador do recurso que deseja manipular, o chamado *ResourceId*. O identificador de recurso pode ser obtido ao se especificar os detalhes do recurso de interesse na interface *ResourceIdFactory* do serviço. No sistema DSMCA os recursos são vistos em dois níveis abstratos: o *Gerenciador de Recurso* e o *Recurso Real*. O primeiro consiste de um programa executável que implementa as interfaces do módulo IDL *ResourceLocalAdministration* do sistema DSMCA. Esse conjunto de interfaces proporciona ao sistema as capacidades de gerenciar um recurso através de chamadas padronizadas e homogêneas, além de prover também a possibilidade de haver interfaces específicas para cada recurso particular. O *Gerenciador de Recurso* pode ser tanto um representante quanto o próprio *Recurso Real*. O recurso real pode ser qualquer elemento capaz de oferecer funcionalidade para a execução do serviço e de seus processos, tais como: dispositivos físicos e lógicos, partições e volumes de disco, sistemas de arquivo, adaptadores e

endereços de rede, filas diversas e programas e aplicativos diversos. Quando o *Recurso Real* está implementado no mesmo programa executável que o seu *Gerenciador de Recurso* então o gerenciador é o próprio recurso. A fim de desvincular a Gerência de Recursos da tarefa de lidar com esses detalhes de cada recurso, o sistema implementa o chamado *Ativador e Localizador de Recursos*. Este serviço implementa dois conjuntos de interfaces, a saber: as interfaces definidas no módulo *ResourceLocatorAdministration* e as interfaces definidas no módulo *ResourceProxyAdministration*. O primeiro define um conjunto de interfaces para um serviço de pedidos de ativação ou localização de um certo recurso. Este é na verdade um serviço de ativação de Gerenciadores de Recurso, os quais como já dissemos representam um certo recurso e apresenta interfaces conhecidas para sua administração. O segundo grupo define um conjunto de interfaces similares às interfaces presentes em um Gerenciador de Recurso e que efetivamente podem ser utilizadas no lugar destas de modo funcionalmente igual, pois são representantes funcionais (proxies) destas. Em outras palavras, o Gerenciador de Recursos de DSMCA não necessita acessar diretamente os gerenciadores de recurso dos recursos que administra. Ao invés disto, ele acessa as interfaces de *ResourceProxyAdministration* presentes nos objetos instanciados no *Ativador e Localizador de Recursos* e que proporcionam a mesma funcionalidade. Isto é conseguido porque o ativador e localizador anexa os objetos e interfaces dos gerenciadores de recurso aos seus objetos *proxies* de gerenciadores de recurso durante a ativação do gerenciador. Um dos efeitos desta técnica é que o ativador e localizador passa a ser um ponto de concentração administrativa e gerencial de todos os recursos do nó, e daí ele ser também chamado de *hub* de administração de recursos.

Figura 4-8 Os recursos e gerenciadores de recurso da aplicação Servidor de Banco de Dados Distribuído

Na Figura 4-8 podemos observar a configuração do serviço do banco de dados distribuído que estivemos utilizando até aqui como exemplo de uma aplicação em execução em um *cluster* DSMCA, porém por questões de simplificação estaremos considerando que o serviço está no momento em execução com apenas duas instâncias, em dois nós do *cluster*. Notemos que o serviço opera com a presença e ativação de um conjunto de quatro recursos, a saber: um volume de disco (*DB Storage*), um adaptador de comunicação em rede (*Net Adapter*), um endereço de comunicação em rede (*Net Address*) e um programa aplicativo servidor (*DB Distributed Application*). O segundo e o terceiro recurso compõem na verdade um único: uma interface de comunicação em rede, necessária para que os

clientes do banco de dados possam acessá-lo através da rede. O primeiro recurso (volume de disco) é necessário para o armazenamento persistente dos dados e metadados do banco. O último recurso corresponde ao aplicativo servidor do banco de dados. Notemos que esses recursos devem ser ativados em uma ordem particular, e assim na figura nota-se a presença dos números 4,3,2 e 1 ao lado dos recursos, indicando a ordem de ativação e desativação correta ou necessária dos recursos a fim de que o serviço inicie sua operação. Notemos também que, para cada recurso, um objeto de gerenciador de recurso (*Resource Manager*) está presente representando o recurso para o sistema, ao mesmo tempo que suas interfaces tem *proxies* no “hub”. Alguns destes gerenciadores não são os recursos reais, como é o caso dos gerenciadores de disco e de elementos de comunicação em rede. Já o gerenciador do aplicativo servidor poderia ser implementado integrado ao mesmo, compondo um único programa executável. Notar também as setas com a legenda “*Uses*”, indicando que o aplicativo servidor utiliza ou acessa os recursos indicados para poder operar.

4.1.8.2 Serviços de Cluster

Para o gerenciamento dos Serviços de Cluster o sistema DSMCA conta com uma gerência de serviços, o *Service Manager*. Trata-se de um serviço local de cada nó do *cluster*, ficando encarregado de iniciar e terminar a execução de serviços no âmbito do nó onde reside. Para realizar a tarefa de iniciar um serviço, o *Service Manager* obtém uma relação dos recursos necessários para a execução daquele serviço bem como uma relação de interdependência e de ordenação desses recursos. Após isso, o *Service Manager* inicia solicitações à gerência de recursos locais do nó (o *Resource Management*) para a ativação de cada recurso necessário. Os recursos são ativados, portanto, em uma ordem tal em que não ocorrem conflitos. Cabe ressaltar aqui que a gerência de recursos local não tem subsídios para ativar grupos de recursos em ordem de interdependência ou

em seqüência. Ela trata os recursos individualmente e lida apenas com as questões relativas à ativação e desativação pontual destes. É portanto responsabilidade do gerenciador de serviços cuidar desse aspecto, enviando solicitações à gerência de recursos em uma ordem adequada, seja para a ativação ou desativação de serviços.

Figura 4-9 Dois serviços e seus respectivos recursos em execução em um nó de cluster, sob gerenciamento do *Service Manager*

Sendo assim, para realizar a tarefa de terminar um serviço, o *Service Manager* obtém a relação dos recursos necessários para a execução daquele serviço e, a partir da relação de interdependência e de ordenação dos mesmos calcula a ordem de desativação, normalmente uma ordem inversa à de ativação. Após isso, o *Service Manager* inicia solicitações à gerência de recursos locais do nó (o *Resource Management*) para a

desativação de cada recurso. Na Figura 4-9 podemos observar como seria o conjunto de recursos ativos em um nó onde dois serviços foram iniciados pelo gerenciador de serviços. A gerência de recursos foi solicitada a ativar todos os recursos necessários em uma ordem adequada. Temos dois serviços ativos: o Serviço 1 (*DB Service*) é um serviço de banco de dados distribuído (multi-instâncias) e o seu recurso servidor (*DB Distributed Application*) utiliza os serviços de grupo de DSMCA para operar corretamente. Ele tem como recursos necessários, além do servidor, um adaptador de rede, um endereço de rede IP e um volume de armazenamento. O segundo serviço ativo, o Serviço 2 (*Web Service*) é um servidor *Apache web* de única instância, e que portanto não necessita dos serviços de grupo para operar. Ele tem como recursos: o servidor *web Apache*, um sistema de arquivos (onde as páginas e aplicativos do *site* são armazenados), um adaptador de rede, um endereço de rede IP e um volume de disco onde o sistema de arquivos reside. Notemos que cada serviço necessita utilizar um adaptador de comunicação separado. Esse adaptador de comunicação faz parte da interface de comunicação do serviço para permitir que os clientes possam acessá-lo. Em geral, os nós de um *cluster* não terão à disposição um adaptador físico independente para cada serviço do nó. Por esta razão, DSMCA define o conceito de *recurso compartilhado*. Um recurso que tem essa característica pode ser atribuído a diversos serviços distintos. A única exigência nesse caso é que esse recurso exista em todos os nós onde os serviços que o utiliza poderão ser executados. Na Figura 4-10 podemos observar uma situação onde os serviços do exemplo anterior (Figura 4-9) estão agora compartilhando o recurso Adaptador de Rede (*Net Adapter*). Porém, notemos que os recursos tipo *Net Address* continuam sendo recursos exclusivos de cada serviço, o que significa que cada serviço é acessível através de um endereço IP diferente no nó. Uma consideração aqui seria o porque de não compartilhar-se o mesmo endereço IP para os dois serviços, uma vez que o que os mesmos podem ser distinguidos no nó através do endereço da

porta de serviço TCP ou UDP que estes utilizam para atender seus clientes. A razão é simples: os serviços DSMCA compõem um conjunto de elementos gerenciados pelo sistema que devem possuir a capacidade de *migrar* de um nó para outro. Desse modo, se é necessário migrar um serviço, ele leva consigo para o novo nó todos os seus recursos, exceto aqueles que são compartilhados (e que, portanto, deve existir em réplica no novo nó).

Figura 4-10 Dois serviços compartilhando o recurso Adaptador de Rede em um nó do cluster

Se o recurso de endereço fosse compartilhado, ele não poderia migrar para o novo nó, pois pode estar em uso por outro serviço. Uma outra forma de ver tal situação seria a seguinte: recursos compartilhados devem existir em réplica em todos os nós do domínio do *serviço*. No caso de um endereço

de rede, ele só pode existir em um único nó a cada momento. Assim, sendo um recurso exclusivo, ele existirá apenas em um nó a cada momento e não haverá conflitos. Logicamente, migrar um endereço IP para outro nó significa ter suporte no sistema operacional e nas implementações dos protocolos de comunicação para que os adaptadores de rede possam ser configurados simultaneamente com mais do que um único endereço. Até a alguns anos atrás eram incomuns implementações de protocolo IP que permitiam tais configurações de endereçamento. Atualmente, porém, essa característica é comum na maioria delas, de qualquer fabricante.

A gerência de recursos deve ser capaz de notificar o gerenciador de serviços sobre quaisquer eventos relevantes relativos aos recursos de um serviço. Desse modo, a conexão entre os objetos da gerência de recursos e o gerenciador de serviços é bidirecional, onde o gerenciador de serviços implementa uma interface de *callback* como cliente da gerência de recursos. Essa interface permite que alterações de estado e condições de falha sejam informadas ao gerenciador de serviços. A gerência de recursos opera com informações relativas à política de falhas dos recursos, tendo freqüentemente a incumbência de reativa-los quando estas ocorrem porém as políticas podem impor um número máximo de reativações num certo intervalo de tempo, quando então a falha é comunicada ao cliente do recurso. O gerenciador de serviços tem outras políticas relativas ao serviço e é este que em última análise decide o que fazer quando um recurso continua em falha mesmo após as políticas de recurso já terem sido aplicadas pela gerência de recursos. O gerenciador de serviços também recebe em sua interface de *callback* notificações sobre comandos enviados aos recursos, tais como comandos de ativação e desativação, habilitação e desabilitação e outros, tais como comandos de suspender e resumir o comando. Isto porque o gerenciador de serviços é em geral o único, porém não necessariamente, cliente da gerência de recursos. A gerência de

recursos é um serviço de interface aberta, permitindo diversos clientes simultâneos, o que abre a possibilidade de que conflitos no controle dos recursos ocorram. Desse modo, é responsabilidade dos implementadores dos clientes desse serviço estabelecerem mecanismos para evitar esses conflitos. Uma possibilidade é utilizar comunicação em grupo ou outros mecanismos de coordenação. O sistema DSMCA apresenta um único cliente local para a gerência de recursos de um nó - o *Service Manager* - e este não implementa mecanismos de coordenação. Porém, uma vez que serviços são o principal foco nas tarefas de gerenciamento de um *cluster*, os quais deve ter a capacidade de migrar de nó para nó ao longo do *cluster*, estes devem ser globalmente administrados através de uma gerência especializada. Assim, uma gerência exclusiva de serviços de *cluster* a nível global é definida, o que é visto na próxima seção.

4.1.8.3 Migração de Serviços no Cluster

Para o gerenciamento global dos Serviços de Cluster o sistema DSMCA conta com o serviço *Cluster Manager*, o qual é a rigor um gerenciador distribuído de serviços, complementando as capacidades dos gerenciadores locais de serviço. Por ser um serviço distribuído o *Cluster Manager* é cliente do serviço de grupos de DSMCA a fim de coordenar as ações entre suas diversas instâncias existente em todos os nós do *cluster*. As atividades do *Cluster Manager* são essencialmente aquelas ligadas à determinação do nó ou nós mais adequados para a execução de um serviço e a solicitação, no nó adequado, da inicialização ou término do serviço. Localmente, o serviço de inicialização e término de um serviço é feita pelo *Service Manager* daquele nó.

Figura 4-11 A gerência global de serviços de *cluster* do serviço *Cluster Manager* em um *cluster* DSMCA com cinco serviços ativos.

Na Figura 4-11 podemos ver a situação de um *cluster* DSMCA onde cinco serviços estão ativos ao longo de dois de seus nós. No primeiro nó está em execução o serviço 1 (multi-instâncias), o serviço 2 e o serviço 3. No segundo nó, está em execução a segunda instância do serviço 1, o serviço 4 e o serviço 5. Os retângulos de linhas tracejadas junto ao localizador e ativador de recursos simbolizam os serviços e os seus recursos. O *Cluster Manager* aparece ao alto como um serviço distribuído ao longo do *cluster*, tendo total visão de todos os serviços do *cluster*, seus estados e localizações. O *Cluster Manager* apresenta uma interface para clientes, de modo que estes possam comandá-lo. Esta interface é mais apropriada na verdade para que uma aplicação de gerenciamento de *cluster* baseada em uma interface gráfica possa conectar-se e comandar o funcionamento do

cluster. As interfaces são definidas no módulo IDL *ClusterManager* de DSMCA e é um conjunto extenso de interfaces e métodos, permitindo operações de *boot* de *clusters*, obtenção de informações e dados diversos de um *cluster*, existindo também interfaces dedicadas ao comando de serviços, tais como inicialização, reinicialização, migração, rotação, e término de serviços, além de permitir obter-se dados e informações diversas sobre quaisquer serviço em execução no *cluster*. O *ClusterManager* é um serviço global e pode portanto ser acessado em qualquer nó do *cluster*, apresentando aos clientes visão consistente do estado do *cluster* e seus serviços, além de coordenar as diversas ações gerenciais tanto internas quanto aquelas devidas a comandos oriundos de uma aplicação de gerenciamento do *cluster*. Por oferecer a possibilidade de interfaces de *callback* para os clientes, o *ClusterManager* notifica aos mesmos quaisquer alterações e ações em andamento ou completadas no *cluster*. Os clientes não participam de decisões, pois os mesmos não formam um grupo de gerenciamento. Somente as instâncias de *ClusterManager* é que formam um grupo coordenado pelo serviço de grupos de DSMCA para coordenar ações e evitar conflitos e problemas. Uma situação bastante clara dessa necessidade é durante a migração de um serviço de um nó para outro, operação essa que envolve a desativação de um serviço em um nó, a migração dos recursos para o novo nó e a ativação do serviço no novo nó. Todas as ações de *ClusterManager* são portanto baseadas em votação e consenso de modo que as ações locais de cada uma de suas instâncias não sejam conflitantes. Também, ele é membro dos grupos internos do serviço de grupos tais como o *NodeMembership*, de modo a receber rapidamente notificações sobre a morte de um nó do *cluster* e iniciar imediatamente ações no sentido de reiniciar os serviços que desapareceram nos nós sobreviventes do *cluster*. Para realizar estas tarefas, *ClusterManager* baseia-se em políticas de falha e de tomada de serviços (*Failover* e *Takeover Service Policies*). Estas políticas estabelecem a forma como o serviço deve escolher, migrar e

reiniciar serviços ao longo dos nós do *cluster*. São políticas globais mas a decisão final sobre essas ações sempre passa por votação e consenso entre os membros sobreviventes de falhas. No caso de não-falhas, tais como um nó do *cluster* sofrendo um processo de *shutdown*, o *ClusterManager* daquele nó simplesmente pára os seus serviços e notifica ao grupo de gerência do *cluster* que está deixando o mesmo, de modo a deixar livre aos restantes a tarefa de decidir quem assumirá os serviços que este está desligando. Esta seria uma situação de *shutdown* imediato ou emergencial. Uma outra situação seria a de um *shutdown* suave, onde o *ClusterManager* do nó sendo desligado pode ele próprio iniciar um processo de migração paulatina de seus serviços para outros nós (sempre havendo votação e consenso para tais ações) e, somente após ter migrado todos os seus serviços é que ele se desliga do grupo e termina suas atividades.

4.1.8.4 Recursos e Eventos

Para que o sistema DSMCA possa ter ciência da situação dos recursos dos serviços de *cluster* que administra, o gerenciamento de recursos pode apoiar-se nas chamadas aos métodos das interfaces dos objetos gerenciadores de recurso de modo a obter a informação desejada. Este tipo de abordagem é eficiente quando as informações que se deseja sobre os recursos são as informações correntes, tais como elas se apresentam naquele momento. Trata-se de uma fonte de informação síncrona, que é requisitada sob demanda. Porém, quando é necessário ter-se informações atualizadas sobre os recursos e seus estados, no instante imediato a um evento que causou uma alteração no estado dos mesmos, essa abordagem mostra ser uma solução pobre. Alterações no estado de um recurso são eventos assíncronos e, portanto, a melhor maneira de ter ciência dos mesmos e permitir o seu tratamento adequado por elementos competentes é implementando um sistema de captura, encaminhamento e seleção de eventos relacionados aos recursos, que permita também a qualquer serviço

ou objeto interessado nos mesmos registrar seu interesse e recebe-los de modo assíncrono. No sistema DSMCA, a gerência de recursos é o principal interessado nos eventos de recursos, porém é possível que outras aplicações ou serviços possam também registrar o interesse em eventos e recebe-los. Para que esta arquitetura de eventos seja possível, é necessário que os interessados em eventos implementem interfaces *callback* de objetos, de modo a adequar-se à sua natureza assíncrona. No sistema DSMCA, as interfaces relativas a eventos são definidas no módulo *Event*. Um objeto ou serviço interessado em receber eventos assincronamente deve implementar uma interface *Event::Interested*, ao mesmo tempo que deve relacionar-se com uma interface *Event::Selector*, a qual permite ao interessado especificar a característica dos eventos em que está interessado. O sistema DSMCA define o serviço *Resource Event Selector*, o qual é dotado de interfaces apropriadas para o recebimento e processamento de eventos de diversas origens. Porém, dado que DSMCA possui o conceito de *hub* de recursos, os eventos podem ser todos de lá recebidos, uma vez que este é *proxy* de todos os gerenciadores de recurso e os eventos por eles gerados são para lá direcionados. Em princípio, o serviço seletor de eventos conecta-se apenas ao *hub*, o que lhe dá a capacidade de receber os eventos de todos os gerenciadores, porém é possível que outras fontes geradoras de eventos conectem-se diretamente ao serviço de eventos e a ele os envie quando necessário.

Figura 4-12 Recursos encaminham seus eventos ao "hub", que por sua vez os encaminha ao serviço de eventos de DSMCA

Na Figura 4-12 podemos observar como o serviço de eventos está relacionado ao *hub* de recursos e dessa maneira é capaz de suprir seus

clientes com informações sobre eventos destes, de modo assíncrono. O módulo *Event* de DSMCA define interfaces básicas para objetos geradores, processadores e encaminhadores de eventos (*Event::Generator*, *Event::Forwarder*, *Event::Processor*) respectivamente para aqueles objetos que geram e necessitam tornar o sistema ciente de seus eventos, para aqueles objetos que são os elementos finais na cadeia de eventos, ou seja, os processadores de eventos, e finalmente para aqueles que necessitam tanto gerar e/ou processar eventos ou que simplesmente encaminham eventos para outros objetos. Os gerenciadores de recurso são programas dotados de objetos diversos, dentre eles alguns baseados na interface *Event::Generator* associados aos recursos que gerencia. Desse modo, os recursos gerenciados são vistos como entidades geradoras de eventos. A interface *Event::Generator* permite que se ajuste as condições nas quais o recurso deve gerar eventos, e desse modo entidades externas ao gerenciador podem controlar os modos de geração de eventos do recurso, modificando ou alterando o tipo e as condições em que eventos são gerados. O gerenciador de recurso é dotado também de alguns objetos da classe *Event::Forwarder* pois internamente o gerenciador de recursos deve encaminhar os seus eventos internos por diversos níveis gerenciais do recurso até que este atinja um encaminhador final que leva seus eventos ao *hub* externo.

4.1.8.5 Recursos e Ações

Vimos na seção anterior que o sistema DSMCA permite que eventos relativos aos recursos que administra podem ser recebidos seletivamente por qualquer elemento interessado nos mesmos. Adiantamos na referida seção que a gerência de recursos (Resource Management) é a principal área interessada nos eventos gerados pelos recursos, uma vez que ela é a gestora dos mesmos e deve estar ciente dos eventos uma vez que estes trazem importantes informações sobre os recursos, em particular sobre alterações de estado, exceções, taxas, quantidades e volumes que marcam

os parâmetros operacionais do mesmo. Certos eventos são caracterizados pelo fato de requererem uma reação em resposta à ocorrência dos mesmos. É o caso quando o evento reporta uma exceção no recurso, a qual deve ser tratada através de uma *ação* sobre os parâmetros do próprio recurso ou sobre recursos relacionados direta ou indiretamente a ele. Em DSMCA define-se o conceito de *Ação de Recurso (Resource Action)* como uma série de chamadas aos métodos de um gerenciador de recurso com a finalidade de reajustar os parâmetros e estado operacional do mesmo. Uma *Ação de Recurso* possui uma identificação (*ActionId*) global no sistema, e a sua finalidade é a de controlar, configurar, corrigir, recuperar e/ou estabilizar o mesmo através de comandos. Uma *Ação de Recurso* desdobra-se em uma sequência de comandos denominada *Seqüência de Ação*. Cada item de uma sequência de ação pode ser tanto uma chamada a um método de uma interface específica de gerenciamento do recurso ou uma chamada para modificação de uma *propriedade* do recurso. Uma sequência de ação é composta de *passos (steps)*, onde cada passo envolve a determinação de um nome de método de controle do recurso ou um nome de uma propriedade do recurso, um nome de parâmetro do método, seu tipo e o correspondente valor a ser utilizado para o parâmetro ou propriedade. O valor utilizado em um passo de sequência, assim como todas as seqüências da uma ação, são previamente configurados para cada tipo de recurso do sistema. Os valores utilizados são especificados como *metadados*, e podem ser valores obtidos de diversas áreas tanto do sistema DSMCA quanto do sistema operacional. No sistema DSMCA, existem serviços exclusivos ligados à execução de ações de recursos.

Figura 4-13 Os objetos e serviços relacionados a Ações de Recurso e seus relacionamentos com o serviço de Eventos em DSMCA

Na Figura 4-13 podemos observar os objetos e serviços relacionados a eventos de DSMCA. Associado como cliente do serviço de seleção de eventos (*Resource Event Selector*), o qual seleciona eventos oriundos dos

recursos do sistema, temos o serviço Seletor de Ações de Recursos (*Resource Action Selector*). Esse, que é o primeiro serviço da área de ações, tem por finalidade básica tornar-se cliente interessado em eventos de recursos, especificamente daqueles recursos que estão sob administração da gerência de recursos (*Resource Management*). A gerência de recursos solicita ao *Resource Action Selector* que: (a) torne-se cliente interessado em eventos de um determinado recurso ao solicitar uma interface *Interested* no serviço seletor de eventos e nela especificar a identificação do recurso de seu interesse e que (b) sempre que um evento daquele recurso lhe for enviado pelo serviço, selecione a identificação da ação adequada para aquele evento, se houver, e notifique a gerência de recursos.

O *Resource Action Selector* obtém uma lista de relacionamentos entre eventos e ações do recurso em questão e desse modo passa a ser capaz de selecionar qual é a ação relacionada a um evento particular oriundo daquele recurso. Sempre que um evento ocorrer no referido recurso, o serviço de eventos seleciona o *Resource Action Selector* como destinatário do mesmo (pois este registrou interesse) e o envia ao mesmo. Este, por sua vez, verifica qual é o evento específico e seleciona o *Id* da ação específica para o evento particular. DSMCA permite que os recursos tenham associados a si listas de eventos e ações relacionadas, de modo a permitir que os serviços de ações e eventos possam trabalhar em conjunto. Notar que na Figura 4-13 a proposta do sistema DSMCA é a de que as ações não sejam incondicionalmente executadas sempre que um evento relacionado à uma ação ocorrer. No sistema DSMCA existe uma área de serviço específica para a execução de Ações de Recurso: a *Resource Action Executor*. Para que uma ação seja executada em um determinado recurso, o cliente solicita em sua interface o *Id* do recurso e o *Id* da ação desejada. A execução é realizada pelo executor de modo incondicional. Porém, podemos verificar pela figura que a gerência de recursos fica como um

intermediário entre o seletor de ações e o executor de ações. Esse arranjo da arquitetura é proposital, com a seguinte justificativa: é a gerência de recursos que é detentora das políticas de todos os recursos utilizados no sistema, e desse modo é ela quem decide, em última análise, se a seleção de uma ação devido ao surgimento de um evento deve se desdobrar na execução de suas seqüências. A gerência de recursos pode determinar, por exemplo, de acordo com as políticas vigentes, se o número de execuções daquela ação já foi realizado em numero excessivo dentro de um intervalo de tempo pré-determinado e configurado, numa situação típica em que a ação é corretiva devido ao surgimento de eventos que reportam problemas no recurso.

Para realizar ações sobre um recurso, o executor utiliza-se apenas das interfaces *proxy* que localiza no *hub* gerenciador (*Resource Locator/Activator*), as quais são funcionalmente equivalentes às aquelas existentes nos gerenciadores locais reais de cada recurso. A finalidade do *hub* aqui torna-se mais clara, livrando o executor de localizar os objetos reais locais de cada recurso e respectivo gerenciador, ficando a cargo da lógica interna do *hub* manter referências aos objetos reais e oferecer objetos *proxy* representantes dos mesmos e com funcionalidade idêntica.

4.1.8.6 Monitoração Remota de Recursos

Vimos na seção anterior que o sistema DSMCA permite que eventos sejam gerados por recursos. Esses eventos podem ser gerados por mudanças corriqueiras de estado dos recursos, tais como aquelas devidas ao envio de comandos aos recursos, mas também podem ser gerados por ajustes específicos em objetos de evento associados aos gerenciadores do recurso (citados em 4.1.8.4). Esses objetos, que tem interfaces derivadas de *Event::Generator*, tem a capacidade de gerar eventos baseados em predicados, mudanças e operações específicas no recurso, utilizando amostragem discreta a intervalos de tempo pré-estabelecidos ou contínua.

São os objetos de interfaces *Event::Discriminator*, *Event::Change* e *Event::Operation* respectivamente. O *Remote Monitoring Service* funciona como um canal de eventos remoto multi-cliente baseado no serviço de grupos. Ele permite que todos os membros associados ao grupo de um recurso recebam notificações sobre os eventos gerados pelo recurso de modo consistente. Para se associar ao grupo de eventos de um recurso, o cliente deve solicitar ao serviço de monitoração remota que proceda à localização do nó do recurso e que lá faça a solicitação para que eventos do recurso sejam notificados ao serviço de monitoria remota naquele nó. Sempre que um evento relativo ao recurso for gerado e recebido pelo serviço, o grupo é notificado em todos os nós de modo que todos os interessados no evento sejam notificados.

A razão da presença de um serviço de monitoração de recursos baseado no serviço de grupos se deve ao fato de que ele oferece um serviço de notificação de eventos de recursos a grupos distribuído, de modo consistente e sincronizado, permitindo uma visão única dos eventos que ocorrem em um determinado recurso no *cluster*. Uma outra razão para o *Remote Monitoring Service* estar apoiado no serviço de grupos tem a ver com o propósito de torna-lo um serviço de confiabilidade, uma vez que o serviço de grupos é um serviço que tem por premissa ser robusto e utiliza-se de um serviço de entrega e difusão de notificações confiável, além de ser um serviço de difusão distribuída. Pode-se, portanto, haver um número qualquer de clientes interessados em eventos de um certo recurso específico, distribuídos ao longo do *cluster*, onde todos recebem os mesmos eventos sobre o recurso em uma ordem estrita.

Figura 4-14 O serviço de monitoração remota de recursos de DSMCA

O *Remote Monitoring Service* tem também a características de tornar a localidade do recurso transparente para todos os interessados. Na Figura 4-14 podemos ver a inter-relação entre o serviço de monitoria remota de recursos de DSMCA e os demais serviços e módulos correlatos.

4.1.8.7 O Serviço de Dados do Cluster

Até o momento vimos que o sistema DSMCA possui muitas áreas e serviços, os quais necessitam obter informações diversas para poder operar. Muitas das informações necessárias para a realização de uma operação por parte de um objeto provedor de serviços são fornecidas explicitamente pelos clientes durante as chamadas que fazem ao serviço na forma de parâmetros para os métodos que representam as primitivas do serviço. Porém, restringe-se a um mínimo a quantidade dessas informações através das interfaces de serviço de modo a minimizar o volume de dados nas chamadas, o que se traduz em mensagens menores em transito através do ORB CORBA que interconecta os objetos do sistema. Em geral, a maior parte das informações que um provedor de serviço necessita para operar e responder às solicitações que lhe são feitas são obtidas pelo próprio provedor no serviço de dados do *cluster*, o chamado *Cluster DB Manager*. Trata-se na verdade de mais um serviço implementado através de objetos CORBA visando prover a todo o sistema DSMCA uma interface padronizada para o banco de dados de configurações gerais de *cluster*. O *Cluster DB Manager* é um serviço presente em todos os nós do *cluster* e opera como um serviço distribuído apoiado no serviço de grupos de DSMCA. A finalidade do serviço de grupo é manter a consistência do banco de dados por todos os nós do *cluster* e implementar semântica transacional para as alterações no mesmo. O serviço de dados é um serviço *multiclient* e possibilita aos seus clientes acesso unidirecional ou bidirecional. O acesso unidirecional é feito através de chamadas explícitas para a obtenção dos dados desejados pelo cliente. Já o acesso bidirecional utiliza uma interface de *callback* no cliente, permitindo que os dados sejam notificados aos clientes imediatamente após alterações, inclusões ou exclusões no banco de dados.

Figura 4-15 O serviço de dados de DSMCA
(Cluster DB Manager) e seus clientes no
sistema.

Para serviços que necessitam de auto-reconfiguração após alterações significativas ou importantes no banco de dados, é recomendável que elas

utilizem o acesso bidirecional ao serviço de dados. Na Figura 4-15 podemos observar a interconexão de clientes do serviço de dados em DSMCA. Notar a interconexão do mesmo ao serviço de grupos de DSMCA, o qual é provedor de serviços para ele. O serviço de grupos não necessita dos dados do *Cluster DB Manager* para iniciar sua operação, assim como alguns outros poucos serviços do sistema.

4.2 DSMCA – Visão Estrutural

Nesta seção fazemos uma breve descrição da estrutura dos módulos da arquitetura DSMCA, propondo formas de construção e as propriedades e capacidades de cada um deles.

4.2.1 Resource Managers

Um *Resource Manager* (gerenciador de recursos) é um elemento do sistema DSMCA que visa representar os recursos que o *cluster* irá manipular a fim de executar seus serviços. Ele opera nos níveis conceituais mais baixos do sistema. O sistema DSMCA não lida diretamente com os recursos existentes nos nós do *cluster*, mas ele precisa de objetos CORBA que representem esses recursos. O gerenciador de recursos é um tipo de programa que implementa e pode ativar esses tipos de objetos, dando desse modo aos serviços DSMCA o poder de administrar e gerenciar os recursos de seus serviços. Para tanto, os gerenciadores de recurso necessitam ter interfaces bem definidas e conhecidas, visando possibilitar a sua utilização pelas demais partes do sistema. Porém, certos recursos tem características e propriedades muito específicas para serem administrados por interfaces padronizadas. Sendo assim, os gerenciadores de recurso permitem que objetos de gerenciamento específicos, com interfaces específicas, sejam embutidos nos mesmos. Esses objetos são ativados no gerenciador de recursos sob demanda e solicitação. Existem interfaces para a manipulação de

propriedades de recursos e também interfaces específicas, com métodos específicos, não documentados ou conhecidos por DSMCA. As interfaces dessas classes de objetos, porém, podem ser descritas no serviço de dados (*Cluster DB Manager*) e com isso permite-se que chamadas aos seus métodos possam ser construídas dinamicamente utilizando as facilidades DII (*Dynamic Interface Invocation*) de CORBA e que ações gerenciais (*actions*) possam ser executadas nas mesmas.

4.2.2 Tipagem e Identificação de Recursos

Como já anteriormente mencionado, os recursos manipulados pelo sistema DSMCA são fortemente tipados, havendo uma classificação extensa para cada espécie. Em DSMCA, um recurso pode ser um elemento de qualquer nível. Existem duas formas de identificação de recursos em DSMCA: o *ResourceInfo*, que identifica e informa as características particulares e específica de um recurso, podendo ser encarado como uma referência absoluta ao recurso, e o *ResourceId*, que identifica um recurso de modo mais genérico, podendo ser visto como uma referência relativa ao recurso. Ambos podem representar um recurso de qualquer nível e espécie. Em DSMCA, recursos não são necessariamente dispositivos físicos, tais como periféricos e interfaces. Recursos podem ser também entidades lógicas, tais como endereços, sistemas de arquivo e programas aplicativos. Ainda, em DSMCA os níveis de classificação de recurso também são vistos como recursos. Um dispositivo físico, tal como um *hard disk* por exemplo, pode ter várias de suas partes e áreas funcionais identificadas por vários valores de *ResourceInfo* e *ResourceId* distintos. Dessa forma, uma certa *função* ou divisão lógica de um *hard disk* pode ser representada como um recurso e possuir os seus identificadores próprios. Na Figura 4-16 podemos observar a denominação de recursos de DSMCA utilizando um *ResourceInfo*.

Figura 4-16 Níveis de um ResourceInfo

A classificação envolve diversos níveis, onde cada um revela um aspecto sobre o recurso. Um *ResourceInfo* inicia a identificação de um recurso a partir do gerenciador que o implementa. Um *ResourceId* suprime a especificação do gerenciador, mantendo todos os demais identificadores. Desse modo, um *ResourceId* identifica um recurso independentemente do gerenciador que o implementa. Essa característica é importante quando se está especificando recursos em níveis do sistema que desconhecem a figura dos gerenciadores de recurso. Um *ResourceId* identifica um recurso de forma relativa a um *cluster* ao qual o recurso pertence. Um *cluster* pode ter diversas *categorias* de recursos, de diversos *tipos*. Diversos *dispositivos* podem ser de um certo tipo e categoria. Um dispositivo pode ser gerenciado por uma ou mais *áreas* gerenciais, e em cada uma dessas áreas pode haver uma ou mais *classes* de *objetos* de gerenciamento.

Tanto o *ResourceInfo* quanto o *ResourceId* são modelados como tipos de dados do usuário no módulo IDL *Resource::* do sistema DSMCA. Conforme já citado, tanto um como o outro pode representar um recurso de qualquer nível, e assim estes tipos de dados são formados por uma união (*union*) complexa de dados chaveada por um tipo *ResourceType* e utilizando um estilo de *wrapping* para cada possível tipo que representa. Assim por exemplo, um *ResourceId* pode ser, de acordo com o valor que o chaveia, um *ClusterResourceId* (nome e número de um *cluster*) ou um *CategoryResourceId* (um *wrapping* de um *ClusterResourceId* e de um nome e número de categoria) ou um *TypeResourceId* (um *wrapping* de um *CategoryResourceId* e de um nome e número de um tipo) e assim por diante. Abaixo, vemos a definição IDL do tipo enumerado *ResourceIdLevel* que serve para identificar o nível de um *ResourceId* e também para chavear a *union* que este compõe.

```

enum ResourceIdLevel {
    RI_Any,
    RI_Cluster,
    RI_Category,
    RI_Type,
    RI_Device,
    RI_MgmtArea,
    RI_MgmtClass,
    RI_MgmtObject
};

struct ClusterResourceId {
    ResourceNumber          cluster_number;
    ResourceName            cluster_name;
};

struct CategoryResourceId {
    ClusterResourceId       category_cluster; // wrapped
    ResourceNumber          category_number;
    ResourceName            category_name;
};

struct TypeResourceId {
    CategoryResourceId      type_category;    // wrapped
    ResourceNumber          type_number;
    ResourceName            type_name;
};

.
.
.
// Definições de DeviceResourceId, AreaResourceId,...
.
.
union ResourceId switch (ResourceIdLevel) {
    case RI_Cluster:
        ClusterResourceId      cluster_id;
    case RI_Category:
        CategoryResourceId     category_id;
    case RI_Type:
        TypeResourceId          type_id;
    case RI_Device:
        DeviceResourceId        device_id;
    case RI_MgmtArea:
        AreaResourceId          mgmtarea_id;
    case RI_MgmtClass:
        ClassResourceId         class_id;
    case RI_MgmtObject:
        ObjectResourceId        object_id;
};

```

4.2.3 Estrutura de um Gerenciador de Recursos

A fim de acomodar a grande diversidade de tipos e classes de recursos que DSMCA deve suportar, os gerenciadores de recurso tem uma estrutura interna que reflete esse fato. Essa estrutura acompanha de perto as características de um identificador *ResourceInfo* e os níveis de classificação de recurso que o mesmo incorpora. A estrutura de um gerenciador de recursos compõe-se de uma árvore (*tree*) de adaptadores de objetos portáveis CORBA – POAs – *Portable Object Adapters* – cuja natureza e característica permite uma construção e inter-relação hierárquica em árvore. Esses adaptadores serão os hospedeiros de todos os objetos de gerenciamento de recursos que o gerenciador vier a implementar e instanciar. Internamente, o gerenciador de recursos possui diversas espécies de grupos de objeto. Um destes grupos, cujas interfaces são definidas pelo módulo IDL de DSMCA denominado *ResourceManagementTree*, lidam apenas com o nascimento e morte de ramos da árvore de recursos. São os objetos de interface *Builder*. Um outro grupo, também definidos nesse módulo IDL, lida com a administração estrutural da árvore, e tem interfaces diferentes de acordo com o nível da árvore onde se encontram. São os objetos de interfaces *TopAdmin*, *Admin* e *SubAdmin*, todas derivadas da interface *BaseAdmin*. Abaixo, podemos observar as linhas iniciais do código desse módulo.

```
/* $Revision: 1.1.0.0 $ */  
  
#pragma prefix "dsmca.org/DSMCA"  
  
#ifndef _MANAGEMENT_TREE  
#define _MANAGEMENT_TREE  
  
#include "Resource.idl"  
#include "POAManager.idl"  
  
module ResourceManagementTree {  
  
    interface Builder;  
    interface BaseAdmin;
```

```

interface TopAdmin;
interface Admin;
interface SubAdmin;
.
.
.
};

```

Outros grupos de objetos, cujas interfaces são definidas a partir do módulo-base IDL *ResourceBaseAdministration*, vão lidar com a administração de ramos inteiros da árvore e também cuidar do gerenciamento local de um nó da árvore. Nas linhas abaixo podemos observar as linhas iniciais desse módulo e as interfaces que define.

```

/* $Revision: 1.1.0.0 $ */

#ifndef _RESOURCE_BASE_ADMINISTRATION
#define _RESOURCE_BASE_ADMINISTRATION

#include "Types.idl"
#include "ResourceManagementTree.idl"
#include "Event3.idl"

// Interfaces basicas de gerenciamento de recursos
// -----

module ResourceBaseAdministration {

    // Interfaces definidas neste módulo

    interface ResourceBaseForManagement;
    interface ResourceBaseForAccountedMgmt;
    interface ResourceBaseForAdministration;

    interface ClusterManagement;
    interface CategoryManagement;
    interface TypeManagement;
    interface DeviceManagement;
    interface AreaManagement;
    interface ClassManagement;
    interface ObjectManagement;

    interface ClusterAdmin;
    interface CategoryAdmin;
    interface TypeAdmin;
    interface DeviceAdmin;
    interface AreaAdmin;
    interface ClassAdmin;
    interface ObjectAdmin;

```

.
.
.
};

A interface cujo sufixo é Management pertence ao grupo daquelas voltadas a gerenciar um nó específico da árvore, o qual representa um determinado recurso. A interface cujo sufixo é Admin pertence ao grupo daquelas voltadas a administrar a criação de um novo ramo de recursos da árvore, bem como destruí-lo. Os objetos cujas interfaces tem o sufixo Management são os primeiros representantes dos recursos no sistema e são suas interfaces que são invocadas quando ações gerenciais nos mesmos são necessárias.

A estrutura em árvore de POAs e objetos de um gerenciador de recursos facilita a administração de uma gama consideravelmente rica de categorias e tipos diferentes de recursos, evitando possíveis conflitos entre nomes similares para tipos distintos de recursos e também entre os objetos padronizados de gerenciamento que devem existir para cada recurso e compartilham nomes iguais. Na Figura 4-17 podemos observar um possível arranjo da árvore de POAs em um gerenciador de recursos de DSMCA.

Ainda, a estrutura em árvore permite uma fácil localização dos objetos no gerenciador, uma vez que o registro de objetos nos POAs leva em geral à um registro de *binding* desses objetos e suas referências nos serviços de localização de objetos de CORBA baseada em uma sintaxe simples de caminho, tal como um caminho para um nome de arquivo em uma estrutura complexa de diretórios em um sistema de arquivos.

Figura 4-17 Estrutura interna primária de um gerenciador de recurso

4.2.3.1 Construção da Árvore de Recursos

Um gerenciador de recursos DSMCA é um servidor ORB CORBA onde uma estrutura em árvore hierárquica de objetos de gerenciamento de recursos são instanciados. Um objeto de gerenciamento representa um recurso do sistema e as chamadas aos métodos de suas interfaces tem efeitos diretos sobre o recurso. Dado que um gerenciador deve ter a capacidade de gerenciar diversas instâncias de um determinado tipo de recurso e também que, como requisito da arquitetura, deve ser capacitado a gerenciar mais de um tipo e/ou categoria de recurso, a estrutura interna de objetos deve ser tal que facilite a lógica e construção do mesmo. A árvore de objetos de gerenciamento de recursos é uma das técnicas adotadas para tal. Para que um recurso possa ser gerenciado, todos os níveis do recurso devem ser ativados no gerenciador, o que equivale dizer que a árvore interna de POAs e os objetos administrativos e de gerenciamento correspondentes a todos os níveis do recurso devem ser instanciados e ativados. Em um nível estrutural mais baixo, a árvore é construída de objetos POA e de objetos de interface *Builder*. Um objeto dessa classe pode construir um novo nó da árvore sob requisição. Ao fazer isto, ele anexa o novo POA como filho do POA onde está hospedado. Em seguida, um novo objeto *Builder* é instanciado e registrado no novo POA. Os objetos *Builder* são locais a cada nó e tem a capacidade apenas de criar novos nós na árvore apenas como filhos diretos do nó onde está registrado. A fim de compor uma estrutura administrativa da árvore administrativa, com visibilidade maior e proporcionando uma interface de solicitação de criação de ramos da árvore, uma classe de objetos de administrativos é utilizada. Esses objetos, que tem interfaces descendentes de *Admin*, residem ao longo dos nós (POAs) da árvore e administram o nascimento e destruição dos ramos da árvore.

Figura 4-18 O primeiro e o segundo passo na criação de um ramo da árvore administrativa de recursos em um gerenciador

Esses objetos não são cientes das funções de gerenciamento de recursos e são na verdade objetos de administração da estrutura de objetos do gerenciador. Na Figura 4-18 (*passos 1 e 2*) podemos observar as

seqüências de solicitação que um objeto *Admin* recebe para que um novo ramo administrativo da árvore de um gerenciador seja criado.

Figura 4-19 Dois ramos administrativos são criados a partir da administração-topo da árvore do gerenciador

Na Figura 4-19 vemos o estado da árvore administrativa após uma seqüência de duas solicitações de criação de (sub)administrações da árvore no objeto *TopAdmin*. Os objetos da classe *Admin* criados sempre mantêm referência aos objetos-pai, assim como os objetos-pai mantêm uma relação de todas as sub-administrações filhas deste. Um objeto administrativo mantém também referência ao objeto *Builder* que atua no nó comum onde residem. Além disso, os POAs são objetos CORBA que naturalmente mantêm uma relação pai-filho.

Figura 4-20 As visões estrutural e administrativa da árvore do gerenciador

O resultado dessas referências múltiplas entre administrações e objetos pais e filhos é mostrado na Figura 4-20 onde nota-se que duas visões da árvore são construídas: uma visão administrativa da árvore e uma visão estrutural. A visão administrativa compõe um conjunto de objetos em níveis distintos da árvore administrando seus ramos. A visão estrutural compõe um esqueleto de hospedeiros (POAs) de objetos diversos (dentre eles, os objetos *Admin*) e de objetos-constructores que tem por finalidade construir novos nós em novos níveis da árvore.

4.2.3.2 Administração de Recursos

Uma vez que uma estrutura de suporte para os objetos de administração de recursos é definida, passamos agora a descrever a estrutura de administração de recursos e objetos de gerenciamento de recursos, objetos esses que também são hospedados nos nós da árvore administrativa. Um objeto de administração de recursos é responsável por criar um ramo da árvore especializado em gerenciar um certo tipo específico de recurso. Assim, existem objetos administrativos de recurso para criar ramos da árvore para cada um dos *clusters* sendo administrados, para cada *categoria* de recurso administrada em um *cluster*, e assim por diante. Ao criar um ramo da árvore, o objeto administrativo cria automaticamente um objeto de gerenciamento de recurso no nó que dá origem a aquele ramo. Um nó da árvore é, portanto, um ponto identificado com um certo recurso, e desse modo tem-se que em cada nó da árvore existe um objeto de *gerenciamento* de recurso, especializado no recurso identificado com o ponto onde ele se encontra na árvore. Se um objeto de gerenciamento de recurso encontra-se, por exemplo, no nó “/webcluster/disk/volume”, esse objeto é especializado no gerenciamento de volumes de disco do *cluster* “webcluster”. Se um objeto de gerenciamento de recurso encontra-se, por exemplo, no nó “/webcluster/network/adaptor/eth0”, esse objeto é especializado no gerenciamento do adaptador de rede denominado “eth0” do *cluster* “webcluster”.

Um gerenciador de recurso inicia com alguns poucos ramos e objetos em sua árvore. Em particular, ele inicia-se com um objeto de administração de *clusters* de modo que ele possa ser solicitado a gerenciar recursos de *clusters* distintos. Para cada *cluster*, um ramo distinto da árvore é criado. No nó correspondente à um *cluster*, um objeto de gerenciamento de um recurso *cluster* é criado.

Figura 4-21 Os objetos de administração e gerenciamento de recursos nos primeiros níveis do gerenciador MGR1

Esse objeto representa um recurso tipado como “cluster” no sistema. Cada *cluster* pode ter diversas categorias de recurso, e assim um objeto administrador de categorias de recursos é criado nos nós correspondentes aos *clusters*. Quando um certo recurso de um *cluster* de uma determinada categoria necessita ser ativado e gerenciado, solicita-se à administração do

cluster que crie um ramo de categoria na árvore, cujo nome é o mesmo da categoria do recurso que se deseja gerenciar.

Figura 4-22 O dispositivo Dev1 gerenciado pelo gerenciador MGR1 e as áreas e classes gerenciais ativas

Na Figura 4-21 podemos ver a árvore de recursos de um gerenciador que encontra-se gerenciando recursos para três *clusters* simultaneamente (*clusters* A, B e C). O *cluster* B tem ativos no momento recursos de duas categorias: C1 e C2. Na categoria C2, existem recursos de dois tipos ativos

no momento: T3 e T4. Na vemos que existe um recurso do tipo T4 ativo denominado Dev1. Esse recurso está ao nível de dispositivo (*device*) e encontra-se sendo gerenciado em três áreas distintas: controle, monitoria e configuração. Na sua área de monitoria, duas classes de monitoração estão sendo realizadas: *DataRate* e *RequestRate*. A primeira classe é uma classe voltada a monitoração de taxas de dados transferidos de e para o dispositivo. A segunda classe é voltada para a monitoração de taxas de requisição de operações no dispositivo. Na classe *RequestRate* estão sendo utilizados dois objetos específicos para monitoração de algum valor ligado a taxas de solicitação ao dispositivo Dev1 - *ReadReqstsSecnd* e *WriteReqstsSecnd*, que indicam as taxas correntes de leitura e escrita no dispositivo, respectivamente. Notemos que objetos de monitoração de recursos com semântica de valor escalar de quantidade (*QuantityMonitor*) são instanciados naquele último nó (*folha*) da árvore. Trata-se de objetos de gerenciamento específicos e simples para a monitoração de um valor específico do recurso. Esses objetos tem portanto uma similaridade com o conceito de *ResourceVariable*, um conceito utilizado pelo sistema de gerenciamento de *clusters* da IBM – que será visto em maiores detalhes em (5.4). Associados a estes objetos, outros podem ser instanciados sob demanda a fim de que eventos sejam gerados quando o valor que o objeto representa atingir uma certa condição ou valor. São objetos geradores de evento específicos. Os eventos gerados por eles são encaminhados pelo gerenciador de recursos ao *hub* para que possam ser coletados e processador.

4.2.3.3 Eventos de Recursos

Os eventos gerados pelos recursos do sistema DSMCA são oriundos de objetos associados ao objeto de gerenciamento do recurso presente no gerenciador.

mostrado na Figura 4-23 onde vemos a interconexão de encaminhadores de evento dos níveis mais alto de um gerenciador de recursos e assim podemos observar o encaminhador-raíz, presente no nó-raiz da árvore de administração de recursos do gerenciador. O objeto de gerenciamento de recurso de cada nó da árvore, que em última análise representa um recurso, possui uma referência ao seu objeto encaminhador de recursos. Essa referência pode ser obtida inclusive através de um método da interface do objeto de gerenciamento de recurso. Na Figura 4-24 podemos ver os objetos de encaminhamento de eventos nos níveis mais baixos do mesmo gerenciador mostrado na figura anterior. Nessa figura, podemos ver que os encaminhadores de evento existem em todos os níveis e em todos os nós, oferecendo o serviço de encaminhamento de eventos a todos os recursos que o gerenciador implementa. No nível mais baixo, vemos que os objetos monitores de valor (*QuantityMonitor*) possuem também seu próprio encaminhador de eventos. Porém, diferentemente de outros objetos de gerenciamento de outros níveis, esses objetos de quantidade não são geradores de eventos. Para que eventos sejam gerados a partir de objetos de quantidade no nível mais baixo, entre outros que podem ali existir, objetos especiais de geração de evento devem ser solicitados ao monitor. Objetos tais como *Event::Discriminator* ou *Event::DiscriminatorSampler* são candidatos a gerador de eventos para esses monitores, pois eles permitem a especificação de *predicados* que podem ser avaliados periodicamente ou continuamente. Quando um predicado do objeto discriminador se torna verdade, ele gera um evento que é passado ao encaminhador local. Esses geradores de evento nascem acoplados ao monitor de quantidade e avaliam seus predicados implicitamente em relação ao valor do mesmo. Na Figura 4-24 podemos ver a presença desses objetos de geração de evento acoplados aos monitores de quantidade.

Figura 4-25 A estrutura em árvore de encaminhadores de evento e a presença de geradores de evento em qualquer nível.

Uma vez que o sistema DSMCA possui um *hub* de gerenciamento de recursos, é natural que todo o fluxo de eventos, proveniente de qualquer gerenciador de recurso, seja para ele direcionado. O *hub* possui internamente uma estrutura e organização similar à de um gerenciador de

recurso. Possui também uma árvore de objetos de hospedagem, administração de árvore, administração de recursos, gerenciamento de recurso e encaminhadores de evento. Na Figura 4-26 podemos ver a situação de interconexão entre os encaminhadores de evento raiz em dois gerenciadores de recurso e o encaminhador de eventos raiz do *hub*. O sistema DSMCA possui um serviço de seleção de eventos, já examinado em (4.1.8.4). Esse serviço deve receber todos os eventos gerados por todos os gerenciadores locais de um nó. A maneira mais eficiente de se conseguir isto é conectar o serviço de seleção de eventos diretamente à raiz encaminhadora de eventos do *hub*. Tal situação também é mostrada na Figura 4-26. O serviço global de seleção de eventos implementa uma interface *Event::Processor* a fim de receber eventos como elo final na cadeia. Porém, processadores locais e específicos de eventos podem existir, não interrompendo o fluxo dos mesmos. Esses processadores podem ser elementos específicos, voltados para tomar ações sobre eventos específicos, de acordo com exigências locais específicas.

4.2.4 Módulos do Sistema DSMCA

4.2.4.1 Localizador e Ativador de Recursos

Um dos problemas da necessidade de se utilizar diversos gerenciadores distintos para os recursos é o fato de que o gerenciamento dos mesmos fica espalhado entre diversos processos. A fim de integrar o gerenciamento de recursos em uma árvore única, estabeleceu-se o conceito de *Hub* de recursos – uma árvore de objetos de gerenciamento similar à mostrada na Figura 4-17 porém consistindo da união de todos os objetos existentes em todas as árvores de gerenciamento locais individuais dos gerenciadores.

Figura 4-26 Os encaminhadores de evento na raiz dos gerenciadores enviam seus eventos para o *hub*. Processadores locais de evento podem estar conectados em qualquer ponto.

Uma vez que a gerência de dados de configuração do cluster não permite que os recursos tenham nomes conflitantes, essa árvore reflete e apresenta de modo unívoco qualquer objeto de gerenciamento, representando assim qualquer recurso. Na implementação de DSMCA, os objetos do hub são

proxies para os objetos locais de gerenciamento, que efetivamente realizam as tarefas de gerenciamento dos recursos. O hub funciona como um ponto de localização facilitada de recursos e de ativação de recursos.

Uma vez solicitado ao hub uma referência à um objeto de gerenciamento de um determinado recurso ainda não ativo, o hub determina qual é o programa gerenciador desse recurso, ativa-o e reflete em sua árvore os objetos do mesmo, de modo sincronizado. Mais alguns detalhes sobre o hub estão na Figura 4-27. As interfaces dos serviços no *hub* estão definidas no módulo IDL ResourceLocatorAdministration do sistema DSMCA, cujas linhas iniciais estão mostradas abaixo.

```
/* $Revision: 1.1.0.0 $ */

#ifndef _RESOURCE_HUB
#define _RESOURCE_HUB

#include "ResourceProxyAdministration.idl"

module ResourceLocatorAdministration {

// Este módulo define as interfaces do serviço
// Resource Locator and Activator (Hub)

interface ResourceRootLocator;

interface ResourceInfoLocator;
interface ResourceIdLocator;
interface ResourceInfoActivator;
interface ResourceIdActivator;
interface ResourceInfoDeactivator;
interface ResourceIdDeactivator;

interface Admin;
```

O *hub* implementa também interfaces *proxy* das interfaces administrativas dos gerenciadores de recurso, de modo a poder exercer a representação dos mesmos. Essas interfaces *proxy* de administração de recursos são definidas no módulo IDL ResourceProxyAdministration de DSMCA, cujas linhas iniciais estão mostradas abaixo.

```

/* $Revision: 1.1.0.0 $ */

#ifndef _RESOURCE_PROXY_ADMINISTRATION
#define _RESOURCE_PROXY_ADMINISTRATION

#include "Types.idl"
#include "ResourceLocalAdministration.idl"

// Este módulo especializa as interfaces de ResourceLocalAdministration
// a fim de torna-las aptas a operar em um Gerenciador de Recursos Proxy
// No sistemas DSMCA, o proxy de gerenciamento é o serviço Resource
// Locator and Activator (Hub). As interfaces aqui definidas são portanto
// utilizadas para a implementação dos objetos de gerenciamento de
// recursos da árvore do Hub.

module ResourceProxyAdministration {

// Interfaces aqui definidas

// Interfaces-base
interface AdministrationProxyBase;
interface AdministrationProxyAdminBase;
interface AdministrationProxyMgmtBase;

// Interfaces para proxies de administração de ramos de recursos
interface ManagerAdmin;
interface ClusterAdmin;
interface CategoryAdmin;
interface TypeAdmin;
interface DeviceAdmin;
interface ClassAdmin;
interface ObjectAdmin;

// Interfaces para proxies de gerenciamento de recursos
interface ManagerManagement;
interface ClusterManagement;
interface CategoryManagement;
interface TypeManagement;
interface DeviceManagement;
interface ClassManagement;
interface ObjectManagement;

```

Figura 4-27 Localizador e Ativador de Recursos de DSMCA

DSMCA Functional Description	Claudio R de Vincenzi	10/11/2002
------------------------------------	-----------------------	------------

O **Resource Locator and Activator (Hub)** é um serviço do DSMCA que visa criar um ponto de administração único dos recursos do sistema. Cada gerenciador de recursos está em princípio conectado ao hub através de seu Event::Forwarder - raiz. O hub tem uma estrutura interna de POAs, gerenciadores e administradores cujo conjunto é equivalente à união dos conjuntos equivalentes nos gerenciadores de recursos. Os gerenciadores e administradores do hub tem a característica de serem *proxies* dos seus pares nos gerenciadores locais. O gerenciamento portanto pode ser realizado com ações no *hub*. O *hub* apresenta interfaces adicionais que permitem a fácil ativação localização e ativação de qualquer recurso do sistema. Sendo que um recurso é representado pelo seu gerenciador e objetos, o *hub* ativa os gerenciadores sempre que um recurso é solicitado.

4.2.5 Seletor de Eventos

Um dos problemas relativos aos gerenciadores de recurso é a captação de eventos geradas nos seus múltiplos níveis e áreas. Eventos são mensagens geradas por qualquer *recurso*. Eventos devem ser tratados de acordo com as informações nele contidas. Para lidar com essa questão, DSMCA define que as árvores de gerenciamento devem ser dotadas de objetos *encaminhadores de evento* (*Event::Forwarders*), os quais são conectados através da árvore de gerenciamento e levam todo e qualquer evento até o seu topo (vide 4.2.3.3). Quando o *hub* ativa um gerenciador, ele conecta o encaminhador do topo da árvore do mesmo a si, de modo que todo o fluxo de eventos, de qualquer gerenciador, dirige-se sempre ao *hub*. Porém, o *hub* é apenas um *proxy* para os recursos e não processa seus eventos. Para facilitar a tarefa de se receber eventos, objetos interessados nos mesmos podem tornar-se clientes do Seletor de Eventos, uma situação já mostrada na Figura 4-26. Trata-se de um serviço que permite que os interessados em eventos solicitem a criação de *seletores de evento* nos quais diversas formas de seleção de eventos podem ser especificadas, fazendo com que o cliente receba apenas os eventos dos tipos que desejar. Esse serviço tem muitas semelhanças com o *CORBA Notification Service* [ORBNotify], especificado pelo OMG. Abaixo, podemos observar as linhas iniciais do código do módulo IDL *Event*, onde as definições das interfaces relativas a serviços de eventos são definidas.

```
/* $Revision: 1.1.0.0 $ */

#ifndef _EVENT
#define _EVENT

#include "Types.idl"
#include "Resource.idl"

module Event {

    interface Handle;
    interface Generator;
    interface Forwarder;
```



```

interface Discriminator;
interface Processor;
interface Sampler;
interface DiscriminatorSampler;
interface Change;
interface ChangeSampler;
interface Operation;
interface Processor;
interface Interested;
interface Selector;
interface ProcessorAdmin;
interface SelectorAdmin;
interface Channel;

```

É possível, por exemplo, que o interessado configure o seletor para que lhe sejam enviados eventos seletivamente de acordo com regras ou expressões. Para tanto, o interessado especifica em chamadas específicas ao seletor estruturas tipo *Selection*, cuja sintaxe é mostrada abaixo em um trecho do código do módulo *Event::* de DSMCA. Pode-se notar que é possível especificar conjuntos de recursos, áreas, níveis e razões de evento em um *Selection*. Um objeto *Selector* permite que vários *Selections* sejam adicionados à seleção de eventos que realiza.

```

// Listas de estruturas de seleção de eventos

typedef sequence<ResourceSelector> ResourceSelectorList;
typedef sequence<AreaSelector> AreaSelectorList;
typedef sequence<LevelSelector> LevelSelectorList;
typedef sequence<ReasonSelector> ReasonSelectorList;

// Estrutura de seleção de eventos
struct Selection {

    // Um Selection tem nome ...
    SelectionName name;

    // ... e listas de seletores de eventos.

    ResourceSelectorList rsrc_sl;
    AreaSelectorList area_sl;
    LevelSelectorList levl_sl;
    ReasonSelectorList resn_sl;

};

```

Mais informações sobre o seletor de eventos podem ser vistas na Figura 4-28.

Figura 4-28 Seletor de Eventos do DSMCA

4.2.6 Seletor de Ações

Uma outra dificuldade associada ao gerenciamento de recursos é a correlação entre eventos e ações gerenciais necessárias para reagir aos eventos. Em DSMCA, existe um serviço específico com essa finalidade, o *Action Selector*. Com base em informações obtidas do serviço de dados de *cluster* e também com base nas solicitações da gerência de recursos de *cluster*, o seletor de ações fica atento a eventos dos recursos que estão sendo administrados. O seletor de eventos filtra apenas os eventos solicitados. Quando um evento é recebido, analisa o mesmo e determina-se se há o registro de uma ação correspondente. Uma ação é uma seqüência de chamadas que visam reagir ao evento. As chamadas de uma seqüência são invocações de métodos em interfaces IDL/CORBA de objetos de gerenciamento levando valores obtidos do sistema ou previamente especificados. Podem também ser chamadas em interfaces de propriedades de recursos levando novos ajustes. Ao se determinar qual é a ação adequada, o seletor de ações comunica o evento e a identificação da ação ao gerenciador de recursos do cluster. Este, que é ciente das políticas de recursos, determina se a ação deve ser levada a cabo ou não. Se for conveniente, ele chama o executor de ações e solicita a execução de toda a seqüência da ação. Na Figura 4-29 temos mais detalhes sobre o seletor de ações. Abaixo, podemos observar as linhas iniciais do código do módulo IDL *Action* de DSMCA.

```
* $Revision: 1.1.0.0 $ */

#ifndef _ACTION
#define _ACTION

#include "Types.idl"
#include "Resource.idl"
#include "Event3.idl"
#include "ResourceLocatorAdministration.idl"

module Action {

    interface Connected;
```

```
interface Connectable;

interface Selector;
interface Executor;
interface Forwarder;

interface SelectorProxy;
interface ExecutorProxy;

interface SelectorAdmin;
interface ExecutorAdmin;
interface LocatorAdmin;
interface EventAdmin;
```

O grande número de interfaces definidas aqui se deve ao fato de que este módulo IDL, assim como os demais módulos IDL de DSMCA, define conjuntos de interfaces utilizadas por diversos serviços, ou diversas partes ou módulos de um serviço. O serviço de ações é dividido em ao menos três partes distintas: seleção, encaminhamento e execução de ações. Além disso, o serviço opera em modalidade conectada com seus clientes, e assim interfaces representantes do serviço (*SelectorProxy*, *ExecutorProxy*) são implementadas por especialização múltipla de seletores, executores e interfaces de conexão. Também, a interface de intermediários entre seleção e execução de ação (*Forwarder*) é definida. Essa interface é implementada em DSMCA pela gerência de recursos. Além disso, interfaces administrativas diversas são usadas para que os clientes do serviço obtenham seus *proxies* (*SelectorAdmin*, *ExecutorAdmin*) e para que a administração do sistema possa conectar os serviços de ações a outros serviços necessários a ela, tais como os serviços de evento e o localizador e ativador de recursos (interfaces *EventAdmin* e *LocatorAdmin*).

Figura 4-29 Seletor de Ações do DSMCA

4.2.7 Gerenciador de Recursos de Cluster

Este gerenciador é capaz de ativar e desativar qualquer recurso especificado na base de dados. Porém, ele não faz isto diretamente, solicitando para tal os serviços do *hub*. Uma vez ativado um recurso, ele passa a monitorá-lo e observa as notificações sobre o mesmo, recebidas através do seletor de ações, o qual fica como encarregado de avisar sobre mensagens oriundas dos recursos ativos. Quando um recurso está sendo ativado, ele registra o mesmo no seletor de ações para que este fique atento a eventos daquele particular recurso. Quando o recurso deve ser desativado, ele solicita que o seletor de ações deixe de observar tais eventos. O gerenciador de recursos é o provedor de serviços para a gerencia de serviços e trabalha com interfaces de *callback* com ele, de modo a poder enviar eficientemente notificações diretas e específicas sobre problemas com recursos. O gerenciador de recursos é local a um nó, e não é um serviço distribuído. Na Figura 4-30 temos mais detalhes sobre o mesmo. Abaixo, as linhas iniciais e a definição de parte da interface *Management* do módulo IDL *ResourceManager* de DSMCA. É esta a interface onde os clientes da gerencia de recursos solicitam operações sobre os recursos definidos no sistema.

```
/* $Revision: 1.1.0.0 $ */

#ifndef _RESOURCE_MANAGER
#define _RESOURCE_MANAGER

#include "Types.idl"
#include "ResourceLocatorAdministration.idl"

module ResourceManager {

    // Interfaces definidas neste módulo
    // -----

    interface Connected;
    interface Management;
    interface Client;
    interface Proxy;
    interface Admin;
```

```

// Exceções erguidas nos métodos deste módulo
// -----

exception UnableToActivate    {};

exception ResourceNotActive   {};
exception ResourceNotStarted  {};

exception FailedStarting      {};
exception FailedStopping      {};

exception ResourceNotFailed    {};
exception FailedToReset       {};

exception NotAllowed          {};
exception NotSupported         {};

exception InvalidLevelOfId     {};

exception AlreadyConnected     {};
exception InvalidClient        {};

// Base para interfaces callback e proxy
// -----

interface Connected {

    boolean  is_connected();

    void     disconnect();
    boolean  confirm_disconnect();

};

// Gerenciamento de recursos: necessita do
// ResourceIdent do recurso
// -----

interface Management {

    // Comandos de gerenciamento de recursos
    // -----

    // Ativar recurso
    void    activate_resource(in ResourceIdent resid)
                                raises(NoSuchResource,                UnableToActivate,
                                    NoNodeTypeAssociation);

    // Desativar recurso
    void    deactivate_resource(in ResourceIdent resid)
                                raises(NoSuchResource, ResourceNotActive, FailedStopping);

    // Iniciar recurso
    void    start_resource(in ResourceIdent resid)

```

```

        raises(NoSuchResource, FailedStarting);

// Parar recurso
void    stop_resource(in ResourceIdent resid)
        raises(NoSuchResource, FailedStopping);

// Reiniciar recurso
void    restart_resource(in ResourceIdent resid)
        raises(NoSuchResource, FailedStopping, FailedStarting);

// Desabilitar recurso
void    disable_resource(in ResourceIdent resid)
        raises(NoSuchResource, ResourceNotActive, FailedStarting,
               NotSupported);

// Habilitar recurso
void    enable_resource(in ResourceIdent resid)
        raises(NoSuchResource, ResourceNotStarted);

// Suspender recurso
void    suspend_resource(in ResourceIdent resid)
        raises(NoSuchResource, ResourceNotActive, FailedStarting,
               NotSupported);

// Recomeçar recurso
void    resume_resource(in ResourceIdent resid)
        raises(NoSuchResource, ResourceNotStarted);

// Restaurar recurso
void    reset_resource(in ResourceIdent resid)
        raises(NoSuchResource, ResourceNotFailed, FailedToReset);

```


Figura 4-30 Gerenciador de Recursos de Cluster de DSMCA

4.2.8 Executor de Ações

Este serviço é complementar ao serviço seletor de ações. Uma vez solicitado a executar uma ação, busca na base de dados de configuração do

cluster todos os detalhes referentes à seqüência da ação a ser executada, coleta os dados necessários do sistema, avalia as funções solicitadas e inicia as chamadas especificadas. Uma ação pode ter um comprimento arbitrário, ou seja, pode ser uma seqüência de qualquer comprimento. O comprimento de uma seqüência é dado em *passos (steps)* e uma seqüência pode ter um número de passos arbitrário. A montagem de uma seqüência de ação se dá pela realização de seus passos. Uma seqüência pode ser, por exemplo, uma chamada a um método de interface, na qual um número arbitrário de parâmetros pode ser necessário. Nesse caso, cada passo seria a descrição da ação necessária para se determinar o nome, o tipo e o valor de um parâmetro do método em questão. Para a determinação do valor do parâmetro, existem formas de especificação de valores através de *metadados*, os quais podem indicar que o valor depende de uma consulta ou busca a dados do sistema DSMCA, tanto nas suas variáveis e objetos de estado quanto no banco de dados de configuração, ou mesmo uma busca a dados no sistema operacional. Assim, a montagem de uma seqüência pode se estender por um tempo relativamente longo. Uma vez completada a ação, o gerenciador de recursos do *cluster* é notificado. Tanto o seletor de ações quanto o executor trabalham com o gerenciador de recursos de *cluster* com interfaces de *callback*, de modo que a notificação de ações pode ocorrer a qualquer momento, de modo assíncrono. Na Figura 4-31 temos mais detalhes sobre este serviço.

Figura 4-31 Executor de Ações do DSMCA

4.2.9 Gerenciador de Serviços

Este gerenciador é capaz de ativar e desativar qualquer serviço de *cluster* no âmbito de um nó. Trata-se de um gerenciador de serviços local do nó,

não sendo, portanto, um serviço distribuído. Quando solicitado a ativar um serviço, ele busca na base de dados detalhes sobre o serviço e determina assim a identificação dos recursos necessários ao mesmo, a ordem, seqüência e interdependência entre os mesmos. Desse modo, pode solicitar à gerência de recursos a ativação dos mesmos na ordem apropriada e assim levantar o serviço. Para desativar um serviço, segue um procedimento reverso, desativando primeiro os recursos que foram os último a serem ativados e por fim aqueles que foram os primeiros a serem ativados. Isso garante uma ordem de desativação tal que as dependências são satisfeitas. Maiores detalhes sobre esse serviço podem ser obtidos na Figura 4-32. Podemos observar abaixo as linhas iniciais do código do módulo IDL ServiceManager, o qual define as interfaces, tipos de dados e exceções do serviço. As interfaces DefaultClusterManagement e DefaultClusterManagement permitem a administração de serviços. A primeira possibilita que um *cluster-default* seja definido, já que o serviço pode administrar recursos de qualquer um dos *clusters* definido no serviço de dados de configuração de DSMCA. A segunda exige que os serviços sejam identificados juntamente com o *cluster* onde são definidos.

```
/* $Revision: 1.1.0.0 $ */

#ifndef _SERVICE_MANAGER
#define _SERVICE_MANAGER

#include "Types.idl"
#include "Service.idl"
#include "ResourceManager.idl"

module ServiceManager {

    // Interfaces definidas neste módulo
    // -----

    interface Connected;
    interface DefaultClusterManagement;
    interface Management;
    interface ServiceIdManagement;
    interface ServiceMgmtInfo;
    interface Client;
    interface Proxy;
```

```

interface Admin;

// Exceções que podem ser erguidas nos métodos
// -----

exception NoSuchService    { ServiceIdIndex index; };
exception NotActiveService { ServiceIdIndex index; };

exception UnknownService   {};
exception UnknownCluster   {};

exception FailedStarting   {};
exception FailedStopping   {};

exception ServiceNotStarted {};
exception ServiceNotActive {};
exception ServiceRunning   {};
exception UnableToHalt     {};

exception UnknownServiceType {};

exception NotAllowed        {};
exception NotSupported      {};

// Caso a configuração (BD) mostre que este
// serviço não tem associação para este nó...

exception NoNodeTypeAssociation {};

exception AlreadyConnected   {};
exception InvalidClient     {};

exception NoDefaultCluster  {};

// Base para interfaces callback e proxy
// -----

interface Connected {

    boolean    is_connected();

    void    disconnect();
    boolean    confirm_disconnect();

};

// Gerenciamento de serviços baseado em cluster "default"
// -----

// A interface para gerenciamento de serviços através
// de ServiceName/Number

interface DefaultClusterManagement {

    // Ações para estabelecer/obter o cluster "default"

```

```

// -----

// Ajustar o cluster-default pelo nome
void set_default_cluster_name(in ClusterName cluster)
    raises(UnknownCluster, NotAllowed, NotSupported);

// Ajustar o cluster-default pelo número
void set_default_cluster_number(in ClusterNumber clsnum)
    raises(UnknownCluster, NotAllowed, NotSupported);

// Obter o nome do cluster-default atual
ClusterName default_cluster_name()
    raises(NoDefaultCluster);

// Obter o número do cluster-default atual
ClusterNumber default_cluster_number()
    raises(NoDefaultCluster);

// Ações de iniciar/parar/encerrar serviços
// -----

// Iniciar um serviço referenciando seu nome
void start_service(in ServiceName svcname)
    raises( NoDefaultCluster,
            UnknownService,
            FailedStarting,
            NoNodeTypeAssociation);

// Parar um serviço referenciando seu nome
void stop_service(in ServiceName svcname)
    raises( NoDefaultCluster,
            UnknownService,
            ServiceNotStarted,
            FailedStopping);

// Encerrar um serviço referenciando seu nome
void halt_service(in ServiceName svcname)
    raises( NoDefaultCluster,
            UnknownService,
            ServiceRunning,
            UnableToHalt);

// Iniciar um serviço referenciando seu número
void start_service_number(in ServiceNumber svcnum)
    raises( NoDefaultCluster,
            UnknownService,
            FailedStarting,
            NoNodeTypeAssociation);

// Parar um serviço referenciando seu número
void stop_service_number(in ServiceNumber svcnum)
    raises( NoDefaultCluster,
            UnknownService,
            ServiceNotStarted,
            FailedStopping);

```

```

// Encerrar um serviço referenciando seu número
void halt_service_number(in ServiceNumber svcnum)
    raises( NoDefaultCluster,
            UnknownService,
            ServiceRunning,
            UnableToHalt);

// Obter Id de serviços a partir de seu nome ou número
// -----

// Obter o ServiceId do serviço de nome svcname
ServiceId get_service_id_for_name(in ServiceName svc)
    raises( NoDefaultCluster,
            UnknownService);

// Obter o ServiceId do serviço de número svcnumber
ServiceId get_service_id_for_number(in ServiceNumber svc)
    raises( NoDefaultCluster,
            UnknownService);

};

```

Figura 4-32 Gerenciador de Serviços locais de DSMCA

DSMCA Functional Description	Claudio R de Vincenzi	10/11/2002
------------------------------------	-----------------------	------------

O **Service Manager** é um serviço DSMCA que cuida de iniciar serviços localmente nos nós do cluster. Um serviço é uma coletânea de recursos interdependentes que quando iniciados tornam o cluster operacional. O Service Manager utiliza os serviços do Resource Manager para ativar e gerenciar os recursos necessários, tendo este uma política de recuperação de falhas locais de recursos. O Service Manager recebe notificações sobre falhas mais graves nos recursos e pode dar início a procedimentos de recuperação do serviço, seguindo uma política pré-estabelecida de recuperação de falhas.

4.2.10 Gerenciador de Serviços de Cluster

Este gerenciador é capaz de ativar e desativar qualquer serviço de *cluster* em qualquer nó. Trata-se de um gerenciador de serviços distribuído e para que haja coordenação entre suas partes, utiliza os serviços de grupo. Quando solicitado a ativar um serviço, ocorre o envio de proposições aos membros do grupo de modo a que haja consenso sobre a forma e o local onde o serviço deve ser ativado. Os procedimentos adotados por essa gerência já foram explanados em 3.3.6 e 3.3.7. Mais detalhes sobre ele podem ser vistos na

Figura 4-33. Abaixo, podemos observar parte do código do módulo IDL ClusterManager onde o gerenciamento de *clusters* e de seus serviços podem ser realizados pela aplicação de gerenciamento do *cluster*. A interface Management permite iniciar as atividades de um certo *cluster* e é também a *factory* de interfaces ClusterServiceManagement, as quais permitem gerenciar os serviços de *cluster*. A interface DefaultClusterManagement permite serviços similares à Management porém permite estabelecer um *cluster-default*. É também *factory* de interfaces DefaultClusterServiceManagement, a qual permite gerenciar os serviços do *cluster* iniciado. O gerenciamento de serviços inclui ações tais como iniciar, parar, reiniciar, rodar e migrar.

```
/* $Revision: 1.1.0.0 $ */

#ifndef _CLUSTER_MANAGER
#define _CLUSTER_MANAGER

#include "Types.idl"
#include "ServiceManager.idl"
#include "ClusterService.idl"

module ClusterManager {

    // Interfaces definidas neste módulo
    // -----

    interface Connected;
    interface Client;
```

```

interface ServiceClient;
interface Proxy;
interface Admin;
interface Management;
interface DefaultClusterManagement;
interface ClusterServiceManagement;
interface DefaultClusterServiceManagement;

// Gerenciamento de clusters
// -----

interface Management {

    // Ative um cluster pelo nome
    void boot_up_cluster (in ClusterName cluster)
        raises (UnknownCluster);
    // Desative um cluster pelo nome
    void shut_down_cluster (in ClusterName cluster)
        raises (UnknownCluster);
    // Ative um cluster pelo numero
    void boot_up_cluster_number (in ClusterNumber clustnum)
        raises (UnknownCluster);
    // Desative um cluster pelo numero
    void shut_down_cluster_number (in ClusterNumber clustnum)
        raises (UnknownCluster);

    // Obtenha a gerencia de serviços
    ClusterServiceManagement get_service_management()
        raises (NoDefaultCluster, ClusterNotOperational);

};

// Gerenciamento de cluster baseado em cluster "default"
// -----

interface DefaultClusterManagement {

    // Especificação do nome do cluster "default"
    void set_default_cluster_name(in ClusterName cluster)
        raises (UnknownCluster, NotAllowed, NotSupported);
    // Especificação do nro. do cluster "default"
    void set_default_cluster_number(in ClusterNumber clnum)
        raises (UnknownCluster, NotAllowed, NotSupported);

    // Nome do cluster "default"
    ClusterName default_cluster_name()
        raises (NoDefaultCluster);

    // Nro. do cluster "default"
    ClusterNumber default_cluster_number()
        raises (NoDefaultCluster);

```

```

// Ative o cluster
void boot_up_cluster ()
    raises(NoDefaultCluster);
// Desative o cluster
void shut_down_cluster ()
    raises(NoDefaultCluster);

// Obtenha a gerencia de serviços
DefaultClusterServiceManagement get_service_management()
    raises(NoDefaultCluster, ClusterNotOperational);

// Obtenha informações de certo cluster
ClusterInfo get_cluster_info ()
    raises(NoDefaultCluster);

// Obtenha informações do tipo do cluster
ClusterTyping get_cluster_type ()
    raises(NoDefaultCluster);

// Obtenha informações de estado do cluster
ClusterStatus get_cluster_status ()
    raises(NoDefaultCluster);

// Obtenha conjunto de nós do cluster
ClusterNodeList get_cluster_node_list ()
    raises(NoDefaultCluster);

// Obtenha informações de tipos de nós do cluster
ClusterNodeTypeDomain get_cluster_node_type_domain ()
    raises(NoDefaultCluster);

// Obtenha informações dos nós do cluster
ClusterNodeDomain get_cluster_node_domain()
    raises(NoDefaultCluster);

// Obtenha informações de um nó do cluster
ClusterNodeInfo get_cluster_node_info(in NodeId node )
    raises( NoDefaultCluster,
            OutOfClusterNodeDomain);

// Obtenha informações dos endereços dos nós do cluster
ClusterNodeAddressList get_cluster_node_address_list ()
    raises(NoDefaultCluster);

// Obtenha conjunto de redes do cluster
ClusterNetList get_cluster_net_list ()
    raises(NoDefaultCluster);

// Obtenha informações sobre as redes do cluster
ClusterNetInfoList get_cluster_net_info_list ()
    raises(NoDefaultCluster);

// Obtenha informações gerais/agregadas sobre o cluster
ClusterData get_cluster_data ()

```

```

        raises(NoDefaultCluster);

};

// Gerenciamento de serviços de cluster
// baseado em cluster "default"
// -----

interface DefaultClusterServiceManagement {

    // Conectar esta interface à interface de gerencia
    // de serviços do cliente

    void connect_service_client(in ServiceClient sclnt)
        raises(AlreadyConnected, InvalidClient);

    // Inicie serviço - nó = politica de takeover + eleição
    void start_service_name (in ServiceName svcname)
        raises(UnknownService);

    // Parada administrativa do serviço
    void stop_service_name (in ServiceName svcname)
        raises(UnknownService,
            ServiceNotStarted);

    // Reinicie administrativamente o serviço
    void restart_service_name (in ServiceName svcname)
        raises(UnknownService,
            ServiceNotStarted);

    // Inicie serviço
    void start_service_number (in ServiceNumber svcnum)
        raises(UnknownService);

    // Parada administrativa do serviço
    void stop_service_number (in ServiceNumber svcnum)
        raises(UnknownService,
            ServiceNotStarted);

    // Reinicie administrativamente o serviço
    void restart_service_number (in ServiceNumber svcnum)
        raises(UnknownService,
            ServiceNotStarted);

    // Inicie serviço - nó = especificado + eleição
    void start_service_on_node_name (in ServiceName svcname,
                                     in NodeName node)
        raises(UnknownService,
            OutOfServiceNodeDomain);

    // Inicie serviço - nó = especificado + eleição
    void start_service_on_node_number (in ServiceNumber svcnum,
                                       in NodeNumber node)
        raises(UnknownService,

```

```

        OutOfServiceNodeDomain);

// Rode serviço p/ outro nó
// nó = política de takeover + eleição

void rotate_service_name (in ServiceName sname)
    raises (UnknownService,
            ServiceNotStarted);
void rotate_service_num (in ServiceNumber snumb)
    raises (UnknownService,
            ServiceNotStarted);
void rotate_service_id (in ServiceId service)
    raises (UnknownService,

            ServiceNotStarted);

// Migre serviço p/ outro nó
// nó = nó especificado + eleição

void migrate_service_name ( in ServiceName service,
                           in NodeName node)
    raises (UnknownService,
            ServiceNotStarted,
            OutOfServiceNodeDomain);

void migrate_service_num ( in ServiceNumber service,
                           in NodeNumber node)
    raises (UnknownService,
            ServiceNotStarted,
            OutOfServiceNodeDomain);

void migrate_service_id_to_node_name (in ServiceId svc,
                                       in NodeName node)
    raises (UnknownService,
            ServiceNotStarted,
            OutOfServiceNodeDomain);

void migrate_service_id_to_node_number(in ServiceId svc,
                                       in NodeNumber node)
    raises (UnknownService,
            ServiceNotStarted,
            OutOfServiceNodeDomain);

void migrate_service_from_to_name ( in ServiceName svc,
                                    in NodeName from,
                                    in NodeName to)
    raises (UnknownService,
            ServiceNotStarted,
            NotRunningOnThatNode,
            OutOfServiceNodeDomain);

```

Figura 4-33 Gerenciador de Serviços de Cluster de DSMCA

4.2.11 Serviço de Grupos

Este serviço proporciona ao gerenciador de serviços de *cluster* e a qualquer outro cliente interessado em serviços de comunicação para grupos de processos ou objetos. Ele oferece comunicação em modalidade *broadcast* atômico com ordenação global, votação de propostas em 1, 2 ou n fases e solicitações de alteração de estado de grupo em 1 ou 2 fases. É essencial para a coordenação de serviços distribuídos entre os nós. Abaixo, podemos observar uma parte do código do módulo IDL *GroupService*, que define as interfaces e tipos de dados do serviço de grupos de DSMCA. A interface *Member* é o *callback* do serviço com um membro de algum grupo. Nela, o serviço notifica o membro sobre solicitações de alteração de estado, votação e mensagens genéricas, além de comunicar o resultado de votações, solicitações de alteração de estado e mudanças na composição dos membros do grupo.

```
/* $Revision: 1.1.0.0 $ */

#ifndef _GROUP_SERVICE
#define _GROUP_SERVICE

#include "Types.idl"
#include "Group.idl"

module GroupService {

    // Interfaces definidas neste módulo
    // -----

    interface Connected;
    interface Member;
    interface Proxy;
    interface GroupAdmin;
    interface Admin;

    // Exceções
    // -----

    exception AlreadyConnected    {};
    exception NotConnected        {};
    exception InvalidClient       {};

    exception DuplicateMember     {};
```

```

exception NotAGroupMember    {};
exception NotGroupLeader     {};

// Base para interfaces callback e proxy
// -----

interface Connected {

    boolean    is_connected();

    void    disconnect();
    boolean    confirm_disconnect();

};

// A interface de um membro de grupo, ou callback de membro,
// a interface que um membro de um grupo deve implementar a
// fim de ficar conectado ao serviço e receber notificações

interface Member : Connected {

    // Para que o serviço saiba que estamos vivos, vai nos
    // chamar aqui periodicamente

    HBSequence heartbeat(in GroupId group_id, in MemberId leader);

    // Solicitações de junção, partida e falha de membros.
    // O membro deve responder através de send_my_opinion()
    // na interface Proxy

    void    member_join_requested ( in ProposalId id );
    void    member_leave_requested ( in ProposalId id );
    void    member_failure_detected ( in ProposalId id );

    // Notificação de alteração no membership do grupo
    // devido a junções, partidas e falhas de membros

    void    membership_change(in MemberIdList membership);

    // Mudança do valor de estado do grupo foi proposta.
    // Algum membro chamou propose_status_change() ou
    // vote_status_change() em seu proxy.
    // Deve-se responder através de send_my_opinion() na
    // interface Proxy

    void    status_change_proposed ( in ProposalId    propo,
                                     in ProposalPhase phase_number,
                                     in GroupStatusData new_status );

    // Notificação de mudança do valor de estado do grupo
    // Algum membro chamou request_status_change() em seu proxy
    // ou uma proposta de mudança de estado foi aprovada.

```



```

void status_changed ( in ProposalId propo,
                      in GroupStatusData new_status );

// Notificação que solicita ao membro que envie seu voto
// para a proposta proposal_id, item proposal_item. O membro
// fica conhecendo também o número de itens da proposta,
// e o número específico do item em votação (item_number),
// além do número de fases que a votação do item terá e qual
// a fase corrente.

// Notificação para votação de um item de proposta
void proposal_from_member ( in ProposalId      id,
                           in GroupId         group_id,
                           in ProposalItem     item,
                           in ProposalItemNumber item_number,
                           in ProposalItemNumber total_items,
                           in ProposalPhase    phase_number,
                           in ProposalPhase    propo_phases
                           );

// Notificação do resultado de uma fase de votação de um item

// Resultado geral de votação de um item
void proposal_item_result ( in ProposalId      id,
                           in GroupId         group_id,
                           in ProposalItemNumber item_number,
                           in ProposalItemApproval item_appr,
                           in ProposalPhaseResultList phase_results
                           );

// Resultado específico - implícita de aprovação do
// item, grupo e membro
void proposal_item_approved (in ProposalId      id,
                            in ProposalItemResult item_result );

// Idem, mas com semântica implícita de rejeição do item
void proposal_item_rejected (in ProposalId      id,
                             in ProposalItemResult item_result );

// Notificação do resultado final da votação de
// toda uma proposta (todos os seus itens)

void proposal_result ( in ProposalId      id,
                      in GroupId         group_id,
                      in Proposal        propo_detail,
                      in ProposalApproval approval,
                      in ProposalItemResultList votation
                      );

// Notificação do resultado final da votação de
// toda uma proposta:

// com semântica implícita de aprovação da mesma
void proposal_approved ( in ProposalId id,
                        in ProposalResult result );

```

```

// com semântica implícita de rejeição da mesma
void proposal_rejected ( in ProposalId id,
                        in ProposalResult result );

// Mensagem de algum membro do grupo
void message_from_member ( in MessageId id,
                          in GroupMessage msg );

};

```

4.2.12 Monitor de Adaptadores de Comunicação

Este serviço realiza a monitoração de adaptadores (interfaces) de rede do cluster, coletando os endereços e seus identificadores de todos os adaptadores dos nós diretamente acessíveis por difusão no nível de enlace. Ele realiza uma verificação periódica do estado dos adaptadores dos nós vizinhos utilizando mensagens de *heartbeat*. Comunica a seus clientes qualquer alteração no conjunto de adaptadores ativos da rede. Trata-se de um serviço distribuído e coordenado sendo, que uma de suas instâncias, ou seja, um dos nós, é eleito o líder do grupo de adaptadores. O líder tem várias incumbências no grupo, tais como manter o *membership* consistente, agregar novos membros ao grupo, desligar membros do grupo, receber notificações de morte de membros, além de receber e propagar listas de *membership* recebidas de outros líderes de grupos de adaptadores vindas de outras redes do *cluster*, de modo a haver uma visão mais ampla sobre os adaptadores ativos em todas as redes. Quando uma rede se particiona, formam-se dois ou mais grupos de adaptadores na mesma, com *memberships* distintos e coordenados cada um por seu líder. Ao trocarem informações de seus grupos, os líderes conseguem informações essenciais para seus clientes pois deste modo eles são informados sobre as condições reais das redes e podem compor caminhos de comunicação lógicos pela elaboração de grafos de conectividade topológica. O principal cliente dos serviços de monitora de adaptadores é a monitoria de nós do *cluster*, que será visto a seguir. Mais informações

sobre o serviço de monitoria de adaptadores podem ser obtidas Figura 4-34.

4.2.13 Monitor de Nós

A partir das informações recebidas dos monitores de adaptador, o serviço de monitoria de nós cria listas de *membership* de nós ativos no *cluster*, isto é, nós para os quais existe conectividade lógica através de algum caminho de comunicação e que também estão apresentando resultados positivos em testes de *liveness* e responsividade (obtidos através de mensagens tais como *heartbeat* nos serviços de adaptador). O serviço de monitoria de nós agrega informação suficiente para se montar um mapa topológico da rede e dos nós do *cluster* e deste modo pode elaborar mapas com os possíveis caminhos de comunicação entre os nós, além de agregar informações tais como as distancias de comunicação envolvidas, latências e atrasos típicos. O serviço de mensagens do *cluster* é o principal cliente da monitoria de nós e tira vantagem das informações que dele obtém para realizar o envio de suas mensagens ao ter à mão listas de caminhos possíveis até um certo destino. Ele pode tentar um ou mais desses caminhos até conseguir ter confirmação do envio com sucesso de uma mensagem.

Figura 4-34 Monitoria de Adaptadores, Nós e Grupos

4.3 Análise de Implementações de CORBA

Como parte dos requisitos iniciais, a utilização da arquitetura de objetos distribuídos do OMG foi utilizada na modelagem orientada a objetos dos serviços de DSMCA. Assim, um conjunto completo de interfaces para os objetos de serviço de DSMCA foi elaborado e descrito utilizando-se a linguagem de definição de interfaces IDL, a qual é um padrão para modelar aplicações CORBA. Essas interfaces são acompanhadas da definição de diversos tipos de dados adequados aos seus parâmetros, a seus valores de retorno, bem como para as aplicações que acessarão ou implementarão os objetos de serviço. A viabilidade da utilização de CORBA constituiu-se parte deste trabalho, tendo havido um profundo estudo do modelo de objetos e da arquitetura de objetos distribuídos proposta pelo OMG, onde se conclui que a mesma apresenta características que não só a viabilizam como também permitem que a implementação dos objetos de serviço do sistema seja facilitada, bem como auxilia e facilita a construção de aplicações distribuídas que dos serviços necessitem. Algumas implementações de CORBA foram analisadas, incluindo aí suas ferramentas, compiladores, código e aplicações geradas, as quais foram testadas a fim de se ter boa noção do desempenho e consumo de recursos das mesmas. Foram feitos testes com as seguintes implementações:

- MICO
- OmniORB
- IONA Orbacus

MICO [MICO] é uma implementação de um ORB CORBA e de um ambiente de desenvolvimento de aplicações CORBA em C++, desenvolvido como um projeto de código aberto. Uma análise de MICO

mostrou que se trata de uma implementação robusta e confiável, sendo conformante com CORBA 2.2 e apresentando características tais como geração de código via compilador IDL de tamanho razoável e de bom desempenho. Infelizmente, tal implementação ainda é objeto de pesquisa e está em desenvolvimento, sendo que alguns aspectos essenciais para a arquitetura DSMCA ainda estão pouco maduros. Em particular, os aspectos relativos aos protocolos IOR e ao estabelecimento de políticas de comunicação ainda apresentam pouca capacidade de controle e configuração por parte da aplicação, o que torna a implementação dos serviços externos de DSMCA, tais como o serviço de mensagens e do monitor de adaptadores, problemática. Um fator determinante para a implementação de tais serviços, o protocolo IOR UDP/IOP não se encontra ainda completamente implementado. Por essa razão, a utilização de MICO foi inicialmente descartada. Porém, acreditamos que em futuro breve MICO deverá contemplar os requisitos necessários para sua utilização na implementação de DSMCA e de aplicações para DSMCA.

Em seguida, uma análise de OmniORB foi realizada. Constatou-se que OmniORB [OMNIORB] é uma implementação robusta e madura, conformante com CORBA 2.3 e disponibilizando alguns serviços CORBA padronizados. OmniORB é um projeto de um ORB CORBA e de um correspondente ambiente de criação de aplicações em C++ e desenvolvido nos laboratórios AT&T de Cambridge, UK em modalidade de código aberto e que também encontra-se em desenvolvimento. Apresenta capacidade de controlar as políticas de comunicação de seu protocolo de IOR, o IIOP, tais como *timeouts* de comunicação e conexão. Infelizmente, até a versão existente, OmniORB disponibilizava apenas o protocolo IIOP, o que inviabiliza a implementação dos serviços externos de DSMCA que necessitam do protocolo *connectionless* baseado em UDP. Entretanto, OmniORB posiciona-se como um dos mais eficientes e rápidos ORB CORBA existentes, possuindo compilador IDL com capacidade de gerar

códigos extremamente compactos e cujo código executável é bastante pequeno. Isto se deve provavelmente às premissas iniciais de seu projeto, nas quais aplicações para equipamentos móveis e celulares baseadas em um ORB CORBA eram previstas.

Por fim, o IONA Orbacus [ORBacus] foi analisado. Orbacus é um produto originalmente desenvolvido pela OOC – *Object Oriented Concepts*, posteriormente incorporada pela IONA Inc, quando então passou a integrar sua linha de produtos. ORBacus é um ORB CORBA totalmente conformante com as especificações OMG e é distribuído com o código fonte. Não é o único produto ORB CORBA desse fabricante, e não possui muitos dos serviços disponíveis no *Orbix EA*, um outro ORB CORBA da própria IONA. ORBacus não é em geral utilizado para atender as exigências extremas de escalabilidade de aplicações empresariais. Entretanto, seu menor *footprint* faz com que se adapte mais facilmente a produtos desenvolvidos por ISVs. ORBacus é um ORB orientado para os desenvolvedores e pesquisadores CORBA que desejam ter acesso ao código fonte. ORBacus é disponível em duas versões de linguagem: C++ e Java. A versão atual de ORBacus é inteiramente conformante com CORBA 2.4. Orbacus apresenta uma interface avançada e flexível para o controle e o estabelecimento de políticas de comunicação de seu ORB, permitindo ajustes independentes por protocolo IOP, por servidor e por objeto. Orbacus disponibiliza o protocolo UDP como base para o IOP, o que o torna um candidato para a implementação dos serviços de DSMCA. Além disso, permite que políticas relativas a *timeouts*, *retries*, *idletime* e reuso de soquetes de comunicação para requisições e conexões sejam estabelecidas, possibilitando controle completo sobre os aspectos de comunicação em ambientes onde esses problemas devem ser previstos, tratados e contornados. Além disso, disponibiliza também interfaces relativas ao controle das políticas de concorrência de execução de chamadas aos objetos, possibilitando que diversos arranjos de linhas de

execução sejam estabelecidos, tais como *threads* dedicadas a objetos, a métodos de objeto, ou que um *pool* de *threads* esteja disponível a todas as chamadas. Essas políticas podem ser definidas por protocolo. De modo que Orbacus apresenta, dentre as implementações de código aberto de um ORB CORBA, aquela que tem maior flexibilidade e recursos disponíveis ao desenvolvedor, possibilitando provavelmente a melhor combinação de facilidade e eficiência de implementação e desempenho dos objetos de serviço de DSMCA.

Capítulo V

TRABALHOS RELACIONADOS

5 Trabalhos Relacionados

Neste capítulo faremos um breve comparativo entre algumas soluções para construção de *clusters* da classe *Beowulf* com características de alta disponibilidade de serviços e o sistema DSMCA proposto. Esses sistemas dividem-se entre aqueles utilizados em *clusters* de pequeno porte, tais como o *LVS*, *Heartbeat*, *Fake*, *MON* e o *cluster HAL* e também sistemas comerciais tais como o *SGI FailSafe* e o *IBM HACMP PSSP* e *RSCT*. Estes sistemas serão descritos nas próximas seções.

5.1 LVS

O LVS – sigla para *Linux Virtual Server*, é um sistema de redireção de conexões TCP e UDP com a finalidade de realizar balanceamento de carga de conexões entre um grupo de servidores de um serviço qualquer. O sistema LVS trabalha com granularidade de distribuição no

balanceamento ao nível de conexão e, portanto, direciona cada nova conexão para um endereço de servidor diferente. LVS permite aos clientes desse grupo de servidores acesso transparente aos serviços, isto é, os clientes necessitam ter apenas conhecimento de um único endereço denominado *endereço de serviço* composto de um endereço IP e do número de uma porta TCP ou UDP. Na Figura 5-1 podemos ver um exemplo de um pequeno *cluster* composto de três nós servidores e um nó de balanceamento de carga. As conexões dos clientes aos servidores do *cluster* é feita através do nó de balanceamento de carga. Na referida figura vemos a representação da ocorrência de três conexões TCP – *TCP C1*, *TCP C2* e *TCP C3*, onde cada uma delas é redirecionada para um dos nós servidores. As características de LVS, tais como o balanceamento de carga de conexões e a de transparência de acesso, caracterizam LVS como um sistema de suporte para a construção de sistemas em *cluster* eficientes no ponto de vista do desempenho, porém de arquitetura simples. Dada a sua simplicidade, podemos relacionar aqui alguns de seus pontos fortes e fracos. Dentre seus pontos fortes, destacamos:

- Elevado desempenho. LVS tem seu balanceador de carga implementado em nível de *kernel* na forma de um módulo carregável em *run-time*. Apresenta baixo *overhead* e pouca latência nas tarefas de redireção de conexões TCP e UDP. Isto caracteriza LVS como um sistema orientado ao desempenho elevado de atendimento de serviços de *cluster*.

Figura 5-1 O sistema LVS compoendo um pequeno *cluster* de quatro nós, sendo três deles nós servidores e um deles realizando balanceamento da carga de trabalho.

- Boa escalabilidade. LVS suporta o redirecionamento para um número qualquer de nós servidores no *cluster*, possibilitando que as capacidades de trabalho, processamento e de transações por segundo oferecidas pelo *cluster* sejam aumentadas pela simples adição de novos nós servidores ao mesmo.
- Boa flexibilidade. LVS é simples e fácil de ser instalado, configurado e gerenciado. Isto advém do fato de que LVS é um sistema com objetivo e foco bem definidos, que é o de prover balanceamento de carga de conexões. As ferramentas de controle e de manutenção de LVS são simples e flexíveis, possibilitando grandes variações e combinações de configurações de seus *clusters*.

Dentre seus pontos fracos, destacamos:

- Disponibilidade de Serviços. LVS não possui características de alta disponibilidade de serviços. LVS não tem nenhum mecanismo para determinação do estado operacional dos nós do *cluster*. Problemas tais como a falha completa de um nó do *cluster* não são detectados por LVS, de modo que ele mantém o seu balanceamento de carga mesmo sob esse tipo de circunstância. Desse modo, durante todo o tempo da falha do nó LVS redireciona parte das conexões ao *cluster* para um endereço IP não-responsivo, o que se traduz numa diminuição imediata da disponibilidade de serviços do *cluster*.
- Baixa confiabilidade. LVS não possui também nenhum mecanismo para detectar o estado operacional e de responsividade das aplicações e serviços para os quais redireciona as conexões (LVS redireciona endereços IP e portas TCP/UDP). Desse modo, se uma aplicação servidora falhar em um nó, LVS manterá mesmo assim a redireção de conexões para ela e isto causa situações ambíguas onde alguns clientes podem obter resposta positiva do serviço enquanto que outros podem ao mesmo tempo receber respostas negativas com alegações de inexistência daquele serviço no *cluster*.

Apesar desses problemas, LVS é uma solução muito interessante e pode ajudar a compor *clusters* de alto desempenho e capacidades de serviços. Quando conjugada a outros sistemas, pode-se conseguir arranjos de *cluster* bastante interessantes, tais como quando conjugada a soluções que visam melhorar a disponibilidade dos serviços do *cluster* e aumentar a confiabilidade que se tem nos mesmos. A conjugação de LVS com outros sistemas tais como *Heartbeat*, *Fake* e *MON* visam atingir esses objetivos, e será o assunto tratado nas próximas sessões.

5.2 O projeto Linux Virtual Server

A partir do sucesso do LVS ocorreu uma iniciativa na comunidade *Linux* dedicada a prover o sistema de características adicionais para atender às necessidades de alto disponibilidade e confiabilidade, as quais não se apresentavam na implementação pura de LVS, mas que se mostraram altamente desejáveis e necessárias para se poder construir *clusters* da classe *Beowulf* com a plataforma *Linux* aptos a enfrentar um ambiente e missão mais crítica. Esse projeto passou a chamar-se *Linux Virtual Server Project* [LVS] e no seu *web site* podem ser encontradas as últimas notícias, referências, manuais e códigos das últimas implementações. A rigor, o projeto adotou algumas soluções que já haviam sido implementadas por outros grupos para prover LVS das características que se procurava. Foi assim então que surgiu uma solução híbrida do sistema LVS, composta do núcleo original de balanceamento de carga agregado de alguns programas adicionais para prover alta disponibilidade e confiabilidade de serviços. O resultado não é a rigor um sistema com forte integração, tendo tornado-se um agregado de programas que conseguem, em certo grau, atingir os objetivos propostos, mas que não foram projetados e construídos para trabalhar com esse nível de integração e interação entre si.

5.2.1 Heartbeat

Heartbeat [HAL] é um sistema de monitoração de nós para pequenos *clusters*. Ele é capaz de realizar o monitoramento operacional dos nós parceiros de um *cluster* através do envio mútuo de mensagens denominadas *heartbeats*, cuja semântica é essencialmente a de uma mensagem tipo “*I am alive!*”. Quando um nó recebe a mensagem *heartbeat* de um nó parceiro, ele considera o parceiro em estado operacional ativo (“vivo”) e, portanto, cumprindo o seu papel e funções no *cluster*. Quando as mensagens de *heartbeat* deixam de ser recebidas,

considera-se o parceiro em estado operacional inativo (“morto”), possivelmente devido a uma falha do mesmo. Em um *cluster* onde *heartbeat* deve operar, cada nó deve ativar o programa *daemon* denominado “heartbeat” em alguma fase durante sua inicialização (através de *scripts* de *boot* ou outro mecanismo equivalente). Sempre que o *daemon heartbeat* de um nó detecta a falha operacional do nó parceiro, ele inicia a execução de um *user exit*, que pode ser tanto um programa externo quanto um *Shell script*. A rigor, mudanças no estado operacional de um nó (tais como *NodeUp* e *NodeDown*) causam a execução de *user exits* específicos nos nós parceiros. Um *user exit* é um código que normalmente realiza ações do tipo *start/stop* em recursos ou serviços, ativando-os e desativando-os no nó em questão de acordo com as informações recebidas em seus parâmetros tais como a identificação do nó que mudou de estado operacional. A finalidade principal de *heartbeat* é portanto prover o *cluster* com uma melhor disponibilidade de seus serviços, reiniciando-os em outro nó do *cluster* sempre que o *nó-default* de um serviço falhar.

Na Figura 5-2 podemos observar um *cluster* operando com monitoração de estado operacional de seus dois nós através do programa *heartbeat*.

Figura 5-2 Um *cluster* de dois nós utilizando *heartbeat* como monitor de estado operacional de seus nós.

Neste *cluster*, quando ambos os nós estão em estado operacional ativo, o nó identificado como *Servidor 1* executa os serviços HTTP e FTP, enquanto que o nó identificado como *Servidor 2* executa os serviços DNS e SMTP. Este é considerado o *Estado Operacional Normal*. Quando ocorre falha de um dos servidores, o *daemon heartbeat* no nó que detecta a falha executa seu *script* de condição *NodeDown*, o qual “toma” os serviços que estavam em execução no nó parceiro em falha em uma ação de *takeover*. Na Figura 5-3 podemos observar como ficaria o estado dos nós e serviços desse *cluster* após a falha do nó *Servidor 1*. Nessa figura, o estado representado é o *Estado Operacional Contingencial* do *cluster*.

Figura 5-3 O mesmo *cluster* após a falha do nó Servidor 1

A seguir relacionamos alguns pontos fortes e fracos de um sistema em *cluster* baseado em *heartbeat* como provedor do recurso de alta disponibilidade. Como pontos positivos podemos destacar:

- Simplicidade. Tanto de instalação quanto de configuração. É um sistema pequeno com configurações baseadas em poucos arquivos de texto, de sintaxe bastante simples.
- Flexibilidade. Pode operar tanto utilizando interfaces de comunicação de LANs tais como *Ethernet* e *FDDI* para troca de mensagens de *heartbeat* entre os nós quanto através de interconexões diretas através de portas seriais ou paralelas, *out-of-band*, aproveitando esse recurso para livrar as interfaces de LAN para o atendimento dos serviços aos clientes.

Como pontos negativos, destacamos os seguintes:

- Pouca escalabilidade. A arquitetura e as semânticas de protocolo adotadas por *heartbeat* é pouco escalável. O sistema *heartbeat* utiliza a prática de difusão por *unicast* de suas mensagens de *heartbeat*, o que equivale dizer que uma mensagem *heartbeat* gerada por um nó é de destino único e que cada nó gera um número de mensagens igual ao número de nós parceiros que houver no *cluster*. Para um *cluster* com poucos nós isto não chega a ser um problema. Porém, para *clusters* com muitos nós o volume de mensagens de *heartbeat* pode ser significativo. Para um *cluster* com oito nós, onde *heartbeat* está programado para gerar mensagens a cada segundo, teremos um volume de $8 \times 7 = 56$ mensagens por segundo. Se a troca dessas mensagens ocorre através de uma LAN, isto não chega a ser um problema, ainda que este tráfego venha a competir com o tráfego de serviço do *cluster*. Esta taxa corresponde, provavelmente, a um volume de dados inviável para conexões via portas seriais e suas velocidades típicas.
- Possibilidade de conflitos. O sistema *heartbeat* realiza ações através de *user exits* – geralmente *scripts* do usuário. Esses *scripts* operam por uma lógica extremamente simples, levando em consideração na tomada de uma ação de ativação ou desativação de serviço, apenas a identificação do nó em falha. Quando ocorre falha de um nó, todos os outros nós detectam quase que simultaneamente sua falha e todos disparam seus *scripts* com a condição de *NodeDown*. E os serviços que desapareceram com a morte do dito nó deverão ser tomados, assumidos por um ou mais dos nós sobreviventes. Nesse momento, a possibilidade de que dois ou mais dos nós decidam assumir um mesmo serviço e seus recursos de um modo conflitante é real, uma vez que *heartbeat* não provê ou utiliza nenhuma espécie de sistema de coordenação em grupo para auxiliar na tomada dessas decisões, que devem ser em

grupo de modo a haver consenso na decisão sobre quem e quais nós deverão assumir o quê. O sistema *heartbeat* opera com o conceito de *nó-default* para um determinado serviço, um parâmetro que identifica o nó que executa o serviço em condições normais. Porém, em caso de falha do *nó-default* não existe uma regra específica e consensual sobre qual o nó que assumirá o serviço. Para certos serviços, caso dois ou mais dos nós decidam toma-lo simultaneamente, o uso compartilhado, simultâneo e não arbitrado de alguns de seus recursos poderá levá-los a condições de inconsistência e até mesmo de completa destruição e de irrecuperabilidade. A única garantia de não-conflito existente é a de haver cópias idênticas dos arquivos de configuração de recurso replicados por todos os nós. Em um *cluster* com muitos nós, a manutenção destes pode ser complicada e sujeita a erros. É por essa razão que *heartbeat* é utilizado apenas em *clusters* com apenas dois nós de modo a não haver a possibilidade desses conflitos.

- Ausência de monitoração de serviços. As mensagens de *heartbeat* carregam apenas informação sobre o estado operacional do *nó*, ou seja, a respeito da saúde do adaptador de comunicação utilizado, dos protocolos de comunicação, sistema operacional e do próprio sistema *heartbeat*. As mensagens não carregam nenhuma informação a respeito do estado dos *serviços* em execução naquele nó. Desse modo, um sistema baseado apenas em *heartbeat* não pode considerar-se de alta disponibilidade de serviços porque não é capaz de detectar que um serviço específico falhou e não mais responde.

Assim, concluímos que um sistema em *cluster* baseado apenas em *heartbeat* tem sérias limitações. Um outro problema não mencionado,

porém importante, é relacionado ao que acontece quando ocorrem falhas nos canais de comunicação do *cluster*. Como exemplo, suponhamos que as mensagens de *heartbeat* sejam transmitidas através de uma rede local *Ethernet* que opera através de um *hub*. Caso o *hub* apresente alguma falha, o programa *heartbeat* em cada um dos nós do *cluster* vai considerar que todos os demais nós falharam e cada um irá tentar por sua vez assumir todos os serviços supostamente em falha, causando assim problemas maiores do que a falha e a não disponibilidade dos serviços.

5.2.2 Fake

Fake [FAKE] é um sistema de *takeover* de recursos de endereço. Trata-se de um programa capaz de realizar uma tarefa importante relativa a ambientes de *cluster* de alta disponibilidade, que é a de tomar e operacionalizar o funcionamento de interfaces e endereços de rede. Quando ocorre a necessidade de um certo serviço migrar ou ser tomado por um outro nó quando problemas com falhas estão acontecendo, é muito comum que um dos recursos com os serviços em questão seja um endereço de rede – tipicamente um endereço IP. Um problema aqui existe, porque não se trata apenas de configurar o adaptador de rede do novo nó com um novo endereço IP adicional ou *alias*. Trata-se de conseguir com que os clientes efetivamente consigam acessar ou continuar a acessar o serviço no novo adaptador (novo nó), o qual possui um endereço MAC (nível 2) diferente do antigo adaptador (antigo nó). Fake apresenta uma solução simples para esse problema: ao configurar o novo adaptador, *Fake* transmite uma mensagem do protocolo ARP (*Address Resolution Protocol*) tipo *ARP_REPLY* em modo *broadcast*. Essa mensagem imita uma resposta a um suposto *ARP_REQUEST*, que na verdade não existiu (falso, daí a razão do nome *Fake*). A grande maioria das implementações dos protocolos IP e ARP aceita e atualiza seu *ARP Cache* com a informação contida na mensagem, a qual associa o endereço IP do serviço

ao novo endereço MAC, do novo adaptador de acesso ao serviço. Porém, alguns equipamentos e sistemas operacionais possuem implementações de IP e ARP que não aceitam o *ARP_REPLY* gratuito de *Fake*. Uma vez que nem todos os tipos de adaptadores ou *drivers* de adaptadores de comunicação para LANs permitem a troca ou modificação do endereço MAC associado ao mesmo, a forma encontrada por *Fake* para tratar esse problema apresenta-se como talvez a única solução possível. Quando essa funcionalidade existe, migra-se para o novo nó tanto o endereço IP quanto o endereço MAC. Porém, nesses casos sempre será necessário haver um adaptador em espera exclusiva para o serviço, uma vez que endereços MAC distinto não podem ser compartilhados por um mesmo adaptador.

Um aspecto interessante de *Fake* é que ele pode ser conjugado com *Heartbeat* (já apresentado em 5.2.1) de modo que os *scripts* que realizam operações de tomada de serviços (*takeover*) possam executá-lo e assim promover também a tomada dos endereços de rede associados ao serviço e assim rapidamente disponibilizá-los aos clientes. Nas últimas versões de *Heartbeat*, o código de *Fake* foi incorporado ao mesmo e assim suas funcionalidades são automaticamente executadas, não havendo a necessidade dos *scripts* as invocarem explicitamente.

Na Figura 5-4 podemos observar um pequeno *cluster* de dois nós em seu estado normal, utilizando *heartbeat* para monitoria do estado operacional dos nós e *Fake* para a tomada de recursos de endereços IP no caso de falha de um dos nós. O nó *Servidor 1* opera com o endereço IP 200.190.10.1 de modo que os serviços HTTP e FTP são acessíveis através desse endereço, enquanto que o nó *Servidor 2* opera com o endereço IP 200.190.10.2 de modo que os serviços DNS e SMTP são acessíveis através deste outro endereço. Havendo uma hipotética falha do nó *Servidor 1*, o programa *heartbeat* no nó *Servidor 2* irá disparar a execução de seus *scripts* para dar início aos serviços que se perderam no nó *Servidor 1*. Um desses *scripts*

invoca *Fake* a fim de tomar para si o recurso de endereço IP 200.190.10.1, adicionando-o ao adaptador de comunicação e promovendo o *ARP Reply spoofing* característico de modo a alterar o conteúdo da entrada com esse endereço IP no *ARP Cache* do equipamento *Roteador* para o endereço MAC do adaptador do nó *Servidor 2*.

Figura 5-4 Um pequeno *cluster* de dois nós utilizando *Fake* para o *takeover* de endereços IP, em seu estado normal.

Na Figura 5-5 podemos observar o estado dos nós deste mesmo cluster, de seus serviços e recursos após a falha do nó Servidor 1 e a conseqüente tomada dos seus serviços e recursos pelo nó Servidor 2, inclusive do seu recurso de endereço IP.

Figura 5-5 O estado dos serviços e recursos do cluster após falha do nó *Servidor 1* e o takeover de recursos, inclusive do endereço IP

5.2.3 MON

MON [MON] – um mnemônico de *Monitor* - é um sistema de monitoria de serviços de *cluster*. Ele pode ser utilizado para monitorar o estado operacional de um serviço através de chamadas explícitas às portas de serviço do servidor sob monitoração e a elas requisitar uma operação-padrão de reconhecimento de serviço. MON é utilizado para monitorar o estado operacional de servidores *web*, servidores de arquivo, *e-mail*, banco de dados e muitos outros. Para cada tipo de serviço, MON possui um programa externo ou *script* especializado no serviço, que conhece suas características, portas de acesso e protocolo de aplicação, além de ter conhecimento sobre requisições padronizadas que podem ser solicitadas e cujos resultados podem ser avaliados ou comparados para se determinar o estado do serviço. MON utiliza um arquivo de configuração simples onde

os endereços dos nós do *cluster* são especificados, bem como os serviços que deverão ser monitorados e os intervalos de tempo adequados entre as execuções de verificação. Tipicamente, uma execução de verificação é realizada a cada 20 segundos, mas dependendo do serviço esse valor pode ser configurado para intervalos menores ou maiores. Quando um serviço deixa de responder no intervalo de verificação, considera-se que o serviço falhou e um *user exit* é executado, tipicamente um *Shell script*. Podem ser especificados diversos *scripts* tanto para a condição de falha de serviço, a condição denominada *alert*, quanto para a condição de retorno ao estado normal do serviço, condição denominada *upalert*. Assim, MON oferece ao gerenciamento do *cluster* chamadas a programas ou *scripts* com semânticas de *ServiceUp* e de *ServiceDown*. Dentre as características positivas de MON podemos destacar:

- Monitoramento ao nível de serviços. MON Introduz a capacidade de monitoramento de serviços para pequenos *clusters* baseados em *software* aberto, permitindo que ações sejam tomadas tanto durante a detecção de falhas quanto de retorno à normalidade dos serviços.
- Flexibilidade. É simples e fácil de ser instalado e configurado, havendo apenas a necessidade de se copiar e editar alguns pequenos arquivos de texto com os parâmetros necessários. A maioria dos serviços *web*, de Internet, de arquivos e outros bastante comuns já vem com *scripts* de monitoria pré-escritos e com configurações-padrão suficientemente adequadas.

Porém, devido à sua simplicidade, MON apresenta também algumas fraquezas, tais como:

- Ausência de Coordenação. Este problema diz respeito ao fato de que MON dispara seus *scripts* de alerta sempre que detecta falha ou retorno à normalidade de um certo serviço. Esses *scripts* são

saídas para código do usuário, *scripts* escritos tipicamente por um gerente de sistemas e que possuem capacidades limitadas na tomada de suas decisões. A rigor, tais *scripts* deveriam trabalhar de modo coordenado com todas as demais instâncias do mesmo que fossem ativadas em quaisquer outros nós. Essa capacidade em geral só se resolve se existe um serviço de suporte assíncrono para comunicação de grupos de processos, o que não é o caso de MON.

- Pouca Visibilidade. Basicamente o problema dos *scripts* disparados por MON é que eles possuem pequena visibilidade sobre o que de fato está realmente acontecendo no *cluster* como um todo. Este é um problema que também aflige o sistema *heartbeat* (5.2.1), o qual já foi comentado anteriormente. As ações que esses *scripts* devem tomar precisam estar pautadas por conhecimento global do estado dos demais nós do *cluster* e mais: de suas intenções. Um nó que percebe uma falha em um serviço pode não ser o único a fazê-lo, e assim todos os nós que a detectaram são potencialmente candidatos a entrar em uma região de disputa pela posse do serviço, uma situação similar àquela que ocorre em ambientes de compartilhamento de recursos em modo concorrente. O grande problema aqui é que certos recursos e serviços são exclusivos, ou seja, apenas um dos nós do *cluster* pode executá-lo, sob pena de que ou o serviço simplesmente não irá funcionar ou, pior ainda, ocorre a real possibilidade de perda de pedidos, de dados e talvez perda irreversível de informações e arquivos. Sendo assim, deveria haver um mecanismo global de coordenação entre esses *scripts* de modo a que eles não entrassem em conflito e realizassem a tomada dos serviços que falharam de modo consensual.

Na Figura 5-6 podemos observar o estado de um pequeno *cluster* de dois servidores onde seus serviços estão sendo monitorados por MON. Neste *cluster*, quatro serviços estão em execução, sendo que cada um deles utiliza um endereço IP distinto para acesso ao mesmo pelos clientes. A configuração é tal que o programa MON de um nó monitora os próprios serviços e os serviços do outro nó e vice-versa. Além de MON, o *cluster* conta também com o auxílio de *heartbeat* para cuidar da monitoria de nós e realizar *takeover* de todos os serviços de um nó em falha e também de *Fake* para realizar *takeover* de endereços IP. Na Figura 5-7 vemos a situação no momento em que o serviço HTTP em execução no nó *Servidor 1* deixa de responder e tal fato é detectado por MON em ambos os nós servidores. No nó *Servidor 1*, MON executa seu *script* de alerta de falha, o qual deve realizar liberação de outros recursos que estavam de posse do serviço naquele nó, tal como o endereço IP 200.190.10.3. No nó *Servidor 2*, MON executa seu *script* e este inicia o processo de tomar o referido serviço para si. Além da tomada do serviço HTTP, o programa MON no nó *Servidor 2* deve acionar *Fake* para que haja também a tomada do endereço IP 200.190.10.3. Na Figura 5-8 podemos observar o estado final do *cluster* após todas as operações envolvendo a tomada e ativação do serviço HTTP e recursos relacionados. Em relação a todas estas operações, vale aqui lembrar os pontos fracos apontados de MON, onde citamos o fato de que é necessário haver coordenação e visibilidade durante as retomadas de serviço caso contrário conflitos podem ocorrer. De um modo simples, podemos exemplificar uma situação em que vão ocorrer problemas nos processos liberação e tomada de recurso. Uma vez que os *scripts* de alerta de falha executam assincronamente nos dois nós, nada garante que no momento em que o recurso IP 200.190.10.3 estiver sendo tomado por *Fake* no nó *Servidor 2* ele já esteja completamente liberado do nó *Servidor 1*.

Figura 5-6 Cluster executando quatro serviços monitorados por MON

Figura 5-7 Falha do serviço HTTP é detectada por MON no nó Servidor 2

Se essa condição ocorrer, existe a chance de que algum cliente solicite um *ARP_Request* exatamente no momento em que *Fake* está enviando seu *ARP* falso. Se o nó *Servidor 1* responder ao *ARP*, esse cliente e outros tem a chance de preencher seu *ARP cache* com a resposta do nó *Servidor 1* e não com a resposta de *Fake*. Uma vez que o nó *Servidor 1* está liberando o endereço IP, mais tarde o problema desaparecerá porém atrasos ocorrerão.

Figura 5-8 O serviço HTTP e o recurso IP 200.190.10.3 são finalmente tomados pelo nó Servidor 2

5.2.4 O *cluster* HAL: LVS, Heartbeat, Fake e MON

Utilizando todos os sistemas vistos até o momento, podemos compor um pequeno *cluster* através de uma combinação das funcionalidades de cada um deles. Na Figura 5-9 podemos observar uma típica configuração de *cluster* de serviços de alta disponibilidade proposta pelo projeto *High-*

Availability Linux [HAL]. Nessa configuração de *cluster* temos um nó balanceador de carga LVS e dois nós de execução de serviços. O gerenciamento de disponibilidade de serviços é realizado em conjunto entre MON e *Heartbeat*. Este tipo de *cluster* utiliza ao menos duas redes, a saber: uma rede de acesso externa (rede IP 200.190.10.0) e uma rede interna para tráfego entre o balanceador de carga e os servidores e também entre os servidores (rede IP 10.0.0.0).

Figura 5-9 Um *cluster* de alta disponibilidade com arquitetura HAL

Neste *cluster*, o serviço HTTP é acessível externamente através do endereço IP 200.190.10.1 enquanto que o serviço FTP é acessível através do endereço IP 200.19.10.2. Estes endereços estão configurados no adaptador de rede externo do balanceador de carga. O LVS instalado no balanceador de carga, ao receber conexões e mensagens TCP e UDP nesses endereços redireciona-as para a rede interna e para tanto traduz o endereço de serviço externo para um dos endereços de serviço interno.

Essa funcionalidade de LVS é denominada NAT – *Network Address Translation*. De acordo com a tabela de roteamento de LVS que está mostrada na Figura 5-9 conexões e mensagens destinadas ao endereço IP 200.190.10.1 e porta TCP 80 (HTTP) são traduzidas e redirecionadas para os endereços IP 10.0.0.1 ou 10.0.0.2, enquanto que conexões e mensagens destinadas ao endereço IP 200.190.10.2 e porta TCP 21 (FTP) são traduzidas e redirecionadas para os endereços IP 10.0.1.1 ou 10.0.1.2. A razão do “ou” anterior é porque ocorre um processo de balanceamento de carga, ou seja, para cada conexão o balanceador do LVS escolhe um dos endereços IP internos e assim redireciona o pedido de serviço para um dos servidores internos. Ao enviar a resposta do pedido, o servidor retorna mensagens ao balanceador de carga, o qual traduz novamente os endereços do *cluster* dos valores internos para o valor externo correspondente ao serviço. Temos, portanto, que tanto o serviço HTTP quanto o serviço FTP executam concorrentemente em ambos os nós de serviço do *cluster*. Na figura não há uma indicação clara, mas o fato é que os servidores HTTP e também FTP existem em duplicata em ambos os servidores e desse modo o estado interno de cada um deles deve ser idêntico. Para esse tipo de serviço, entende-se então que as pastas, arquivos, programas e dados que compõem os *web sites* hospedados no *cluster* deve ser comum a ambos os servidores HTTP e também os arquivos armazenados no servidor FTP devem ser iguais em ambos, caso contrário teremos uma situação de inconsistência dos serviços.

Nessa configuração de *cluster* HAL, o serviço de monitoria de serviços pode ser ativo no próprio nó de balanceamento. Uma vez que os serviços existem ativos em todos os nós, não há necessidade de migração ou *takeover* dos mesmos. Porém, no caso de uma falha de serviço é essencial que ocorra uma mudança na tabela de roteamento do LVS a fim de que não mais sejam direcionadas conexões para um nó onde o serviço não mais responde e daí a razão de MON estar em operação no nó de

balanceamento. Esse procedimento evita que conexões a serviços sejam enviadas para um nó não preparado para responder e garante portanto a continuidade do serviço, ainda que provavelmente com alguma penalidade em termos de desempenho.

Um outro problema com essa configuração de *cluster* é o fato de que o balanceador de carga, por ser único no *cluster*, representa um *SPOF* – *Single Point of Failure* (ponto de falha único), o que significa que uma falha no nó de balanceamento pode comprometer toda a disponibilidade dos serviços desse *cluster*.

Figura 5-10 Um *cluster* HAL dotado de redundância no balanceador de carga

Por conta destes problemas, pode-se pensar em uma melhoria na configuração desse *cluster* de modo a torna-lo mais resistente à falhas do balanceador. Uma solução possível é replicar o nó de balanceamento de carga, tal como mostrado na Figura 5-10.

Nesta nova configuração, um segundo *LVS Load Balancer* fica em estado de espera por falhas no nó de balanceamento primário. Para tanto, utiliza-se *heartbeat* como monitor entre os nós de balanceamento e configura-se LVS como o principal serviço desse par de nós do *cluster*. Caso ocorra uma falha no nó de balanceamento primário, o *heartbeat* no nó de balanceamento secundário (*stand-by*) realiza uma tomada tanto do serviço de balanceamento de carga (LVS) quanto da monitoria de serviços (MON) e também dos endereços IP de serviço e também do endereço interno do balanceador (no caso, 10.0.9.1) utilizando *Fake*.

Façamos agora uma análise dos pontos fortes e fracos dessa arquitetura. Dentre os pontos fortes, podemos destacar:

- Bom desempenho
- Boa disponibilidade
- Boa Escalabilidade
- Boa Flexibilidade

Entretanto, temos que ao mesmo tempo existem alguns pontos fracos na arquitetura. Em parte, esses pontos fracos existem porque soluções de *cluster* tais como HAL [HAL] são construídos a partir de diversos subsistemas distintos e elaborados para finalidades diversas. Não são necessariamente projetados para operar em conjunto, de modo integrado. Desse modo, a integração entre eles é muito fraca, alguns serviços essenciais não existem e surgem assim vários pontos de fraqueza na solução apresentada. Dentre os pontos fracos, destacamos:

- Ausência de coordenação global. Durante certos momentos, é essencial a existência de algum mecanismo ou serviço que coordene as tomadas de ações em conjunto. Como já foi

exemplificado anteriormente em (5.2.1) e (5.2.3), podem ocorrer situações que levam ao desastre caso essa coordenação não existir. Para que coordenação global possa acontecer, via de regra é preciso que um serviço adicional de coordenação, independente dos serviços de administração do *cluster*, exista e esteja disponível para as varias áreas gerenciais do *cluster*. A coordenação de ações gerenciais pode ser obtida ao utilizar-se algum serviço de comunicação em grupo que ofereça uma semântica transacional para os seus clientes, garantindo o recebimento de propostas de mudança ou ação, garantindo que houve aceitação e consenso sobre ações e mudanças, efetivando mudanças com atomicidade, garantindo o retorno ao estado original em caso de discordância ou impossibilidade e garantindo também a serialização (ordenação) de suas diversas etapas com sincronia virtual ou estrita.

- Ausência de Gerenciamento de Serviços. Para que um sistema em *cluster* possa ter seus serviços bem administrados, é preciso que exista alguma espécie de gerência global de serviços do mesmo. Essa gerência deve ter uma capacidade de visão global de todos os serviços que o *cluster* deve executar e também deve saber o estado e local onde os mesmos estão executando em um dado momento. Além disso, deve ter conhecimento para coordenar possíveis migrações dos serviços para outros locais e deve também conhecer as políticas envolvendo os serviços do *cluster*.
- Ausência de Gerenciamento de Recursos. Para que sistemas em *cluster* possam administrar eficientemente e corretamente seus serviços, é essencial que o conceito de recurso seja bem definido como uma entidade no sistema e que este tenha um tratamento dedicado e uma gerência específica. Recursos necessitam ter suas características completamente definidas e conhecidas pelo sistema

que os administra, de modo que o seu controle e configuração possam estar sob responsabilidade do sistema de administração do *cluster*. Para um eficiente controle, configuração e monitoração de seus recursos, o gerenciamento dos mesmos deve ser parte integrada ao sistema de administração do *cluster*. Em sistemas simples de *cluster* tais como HAL, a competência para lidar com recursos fica concentrada e restrita à lógica embutida em *scripts* de comandos e não em uma base de conhecimento global.

- Ausência de um Serviço de Eventos. Eventos são mensagens assíncronas oriundas dos recursos e das diversas partes da administração de um sistema em *cluster*. Eventos devem ser coletados, encaminhados e interpretados por serviços específicos relacionados ao gerenciamento de recursos. Os eventos que os recursos podem emitir devem ser previamente conhecidos pelos serviços de administração do *cluster* de modo que ações adequadas a cada tipo de evento possam ser determinadas e encaminhadas para os serviços competentes. A reação a eventos diversos é parte essencial de um sistema eficiente de administração de *cluster* pois permite que as diversas partes, serviços e recursos existentes no *cluster* possam emitir mensagens assíncronas que tem um significado específico no seu contexto e com isso é possível elaborar e iniciar uma reação para responder rapidamente ao evento. Em sistemas tais como HAL, o conceito de evento fica restrito aos monitores de recurso e de nós. Nesse modelo, os eventos não são oriundos dos recursos em si, mas de programas que os observam. Também, não é possível a programas interessados nos eventos detectados por esses monitores receber alguma espécie de notificação. Essas características mostram que o modelo de eventos utilizado é extremamente pobre.

- Ausência de um Serviço de Ações. Quando um evento ocorre, é necessário que alguma área administrativa do *cluster* os receba, os processe e os analise, procurando correlacionar o evento a uma ação cabível, possível e/ou necessária. Um sistema de administração de recursos deve prever e manter estabelecidas certas ações gerenciais padronizadas que devem ser levadas a cabo sempre que certas situações ocorrerem. Os eventos são os informantes que podem tornar ciente o gerenciamento de que uma dessas situações ocorreu e assim uma ação padronizada deve ser executada. Serviços de ações normalmente são divididos nas funções de processamento e análise de eventos, correlação evento-ação e execução de ações. Em sistemas tais como HAL, ações são executadas quando certos eventos são detectados, tais como a falha de um nó ou de um serviço. Porém, as ações estão na verdade contidas na lógica de um *script* externo do usuário. Esta abordagem, ainda que permita a reação adequada, apresenta o problema de ter poucos subsídios e informações na tomada de uma ação. Executores de ação integrados à gerência do *cluster* tem acesso à uma gama muito maior de detalhes dos recursos e de suas características, possibilitando assim que estes possam realizar ações do forma mais eficiente, focadas e com informações mais precisas e facilmente acessíveis.

5.3 SGI Linux FailSafe™

Linux FailSafe [FAILSAFE] é um produto desenvolvido como um projeto de código aberto pela Silicon Graphics, Inc (SGI). Trata-se de uma solução de Alta Disponibilidade (HA) de nova geração para suportar aplicações críticas de negócios. Fornece uma maneira simples de fazer as aplicações de *cluster* terem características de HA sem ter que se recorrer a reescrita ou recompilação de código ou a necessidade de investimentos em

soluções de *hardware* de custo elevado. Linux FailSafe fornece um ambiente de *clustering* robusto ao apresentar a capacidade de suportar falhas em componentes e nós do *cluster* e eliminar os problemas de pontos de falha único (*SPOF's*). Linux FailSafe é arquitetado para escalar a *clusters* de até 16 nós com compartilhamento recursos de armazenamento entre seus membros, porém é possível trabalhar em modo totalmente não compartilhado. O armazenamento compartilhado de Linux FailSafe permite que múltiplos servidores possam assumir o controle dos dados no caso de uma falha. No momento da uma falha, as aplicações são recomeçadas nos sistemas (nós) sobreviventes do *cluster* e os sistemas de arquivo são tornados automaticamente disponíveis para as aplicações. Linux FailSafe trabalha com o conceito de *plug-in* como forma de interface entre ele e as aplicações e/ou recursos, o que o torna um sistema de alta disponibilidade. Para a comunicação entre os nós do *cluster*, Linux FailSafe permite a utilização de diversos tipos de redes locais tais como *Ethernet* (2.4.3.1) e *FDDI* (2.4.3.2), além de permitir a utilização e aproveitamento das interfaces seriais dos equipamentos para a composição de uma rede de controle e de monitoração via um protocolo de *heartbeat*.

Na Figura 5-11 podemos observar uma configuração típica de um *cluster* SGI Linux FailSafe composto de dois nós servidores e um gabinete de discos compartilhados através de um barramento SCSI (2.5.1)(2.6.1).

Figura 5-11 Um cluster SGI Linux FailSafe [FAILSAFE] típico.

5.3.1 Visão Geral da Arquitetura

Figura 5-12 Arquitetura do sistema Linux FailSafe

As camadas da arquitetura de SGI *Linux FailSafe* podem ser observadas na Figura 5-12. A Silicon Graphics define uma camada de serviços orientados à construção de aplicações de alta disponibilidade denominada

CHAOS - *Clustered High Availability Operating Services*, na qual apóiam-se alguns de seus produtos, tais como Linux FailSafe e outros relacionados. Trata-se de uma camada comum de infraestrutura de serviços para *clusters* de alta disponibilidade. CHAOS fornece, portanto, o suporte necessário para a construção de soluções de alta disponibilidade. Todas as camadas de CHAOS residem no espaço do usuário. As camadas de infraestrutura de CHAOS compreendem: (1) Serviços de *Membership* do *Cluster* (CMS), (2) Serviços de Comunicação em Grupo (GCS), o (3) Gerenciador de Recurso do Sistema (SRM), os Serviços de Restauração de *Cluster* (CRS).

Linux FailSafe é construído a partir de uma camada de administração de sistemas em *cluster*, apoiada na camada CHAOS. É uma camada de serviços e é denominada CAS – *Cluster Administration Services* (Serviços de Administração de *Cluster*) composta do *daemon* de administração de *clusters* (CAD), do *daemon* servidor da base de dados de configuração de *clusters* (CDBD) e da base de dados da configuração propriamente (CDB).

A seguir faremos uma breve descrição dos serviços e funcionalidades de cada camada de CHAOS e CAS.

5.3.2 CMS – *Cluster Membership Service*

Os serviços de *membership* de *cluster* (CMS) compõem o nível mais baixo da infraestrutura de CHAOS, fornecendo o *membership* de nós de *cluster* às camadas superiores, oferecendo a seus clientes o *status up/down* dos nós dentro do *cluster*. O CMS compõe-se de um *daemon* servidor e de uma biblioteca que define a interface com seus clientes. Há sempre um *daemon* em execução por nó no *cluster*. Estes servidores monitoram-se trocando mensagens de *heartbeat* e são responsáveis por fornecer aos seus clientes informação sobre o *status* dos nós. São também responsáveis para a estabilidade das visões de *membership* e da concordância sobre o

histórico dos mesmos. O serviço de *membership* tem também a função de prover ao *cluster* a propriedade *I/O CLEAR* – a qual define que os nós que não são membros não são autorizados a alcançar os recursos compartilhados do *cluster*.

5.3.3 GCS – *Group Communication Services*

O serviço de comunicação em grupo de CHAOS (GCS) é um serviço distribuído apoiado sobre os serviços de *membership* de CMS. O GCS define uma abstração para grupos de processos – uma coleção de processos distribuídos através de um *cluster* cooperando para fornecer um serviço de aplicação distribuída. O GCS fornece aos seus clientes: (a) uma visão consistente do *membership* do grupo mesmo durante mudanças no *membership* ou durante a ocorrência de falhas de processos-membro ou de nós do *cluster*, e (b) um serviço de mensagens confiável, ordenado e atômico. Aplicações distribuídas utilizam o GCS para a notificação de término normal e de falha das instâncias da aplicação que rodam em um nó do *cluster*. Ainda que as aplicações distribuídas devam empreender a tarefa de recuperar e reintegrar suas instâncias, GCS alivia a aplicação da tarefa de monitorar instâncias e suas condições de *liveliness* em um ambiente distribuído.

5.3.4 SRM – *System Resource Manager*

O SRM - gerente de recurso do sistema - controla os recursos ao longo do *cluster*. O SRM é uma aplicação distribuída e utiliza os serviços de comunicação em grupo para comunicação entre suas instâncias, de modo a fornecer acesso exclusivo aos recursos que gerencia. O SRM contém informação detalhada pertinente ao gerenciamento dos recursos do sistema, possibilitando o controle de recursos tais como discos, volumes, sistemas de arquivo, adaptadores de rede, endereços IP e outros recursos, inclusive aqueles definidos pelo usuário. O SRM é responsável por iniciar,

parar e monitorar recursos de alta disponibilidade no *cluster*. Ele também tem a incumbência de reiniciar localmente recursos em um nó antes de tentar mover o grupo de recursos a que ele pertence para um outro nó do *cluster*. SRM fornece também aos clientes uma interface que permite adicionar mais tipos de recursos ao *cluster* através do uso de *plug-ins* – módulos externos para controle e monitoração de recursos.

5.3.5 O Módulo *LinuxFailSafe*

O LinuxFailSafe é um módulo do sistema Linux FailSafe cuja finalidade é fazer com que as políticas de recursos do *cluster* sejam cumpridas. Ele apóia-se nos serviços do SRM para manter os recursos e o funcionamento do *cluster* dentro das políticas estabelecidas. Para isso, ele toma decisões de alocação dos grupos de recursos ao longo dos nós, em particular durante a ocorrência de falhas e de mudanças na configuração do *cluster*. Esse módulo permite que os clientes do *cluster* e usuários definam novas políticas para recursos e grupos de recursos.

5.3.6 Ferramentas Administrativas *Linux FailSafe*

A administração de um *cluster* da arquitetura *Linux FailSafe* é feita sobre os serviços do CAS através de aplicações dotadas de interfaces gráficas com o usuário (GUI) ou de utilitários de linha do comando (CLI), os quais permitem que os usuários administrem, configurem e monitorem a camada CHAOS e as outras aplicações controladas por FailSafe. Toda a informação de configuração é armazenada em uma base de dados da configuração (CDB).

5.3.7 Base de Dados de Configuração

A base de dados da configuração do *cluster* (CDB) é composta de duas porções: (1) Um *front-end* que fornece a visão de cliente do CDB, compreendido por uma biblioteca de API e por outros serviços

implementados pelo CAS. (2) Um ou mais *back-ends*, bibliotecas "de encaixe" (*plug-in*) que guardam o armazenamento real de algum subconjunto de dados de configuração, ou em alguns casos, a síntese dinâmica daqueles dados. A base de dados de configuração é replicada em um conjunto de nós e fornece uma semântica de transação para operações de apenas leitura ou de atualização. Utiliza os serviços do GCS para tal semântica. Se um nó falhar ou for desligado, ou, se a base de dados de configuração em um nó se tornar obsoleta, inicia-se uma operação de cópia da nova versão da base de dados para aquele nó no momento em que este solicita junção ao *membership* do grupo GCS da base de dados.

5.3.8 Uma análise do SGI *Linux FailSafe*

Feitas as descrições sobre o sistema SGI *Linux FailSafe* nos tópicos acima, teceremos agora alguns comentários sobre tal sistema. Em primeiro lugar, trata-se de um produto que, apesar de estar em desenvolvimento em regime de código aberto, é comercial e propriedade da Silicon Graphics Inc. e portanto tem seu desenvolvimento e direcionamento dirigidos por essa empresa. Uma análise das descrições do produto revela que ele destina-se a prover os *clusters* de estações de trabalho e de servidores SGI de um ambiente de alta disponibilidade para aplicações tradicionais e largamente utilizadas, tais como compartilhamento de arquivos em rede via NFS, alta disponibilidade de serviços *web* e Internet tais como o Apache, servidores FTP, servidores de e-mail SMTP (Postfix e Sendmail) e servidores POP3, serviços de integração com ambiente Microsoft tais como o Samba e também prover alta disponibilidade de servidores de banco de dados tais como Oracle e DB2. Desse modo, fica claro qual é o foco do sistema e produto. A análise do código e pacotes necessários para sua instalação revela que Linux FailSafe foi desenvolvido utilizando o suporte de outros produtos SGI, tais como o SGI Rhino. Rhino é o nome de uma infraestrutura para construção de aplicações que configuram,

controlam e monitoram *hardware* e *software*. Rhino fornece interface de usuário gráfica comum (GUI), consistente, baseada em tarefas, internacionalizada, além de uma interface de linha de comando (CLIs) que os administradores do sistema podem usar para escrever seus *scripts*.

Uma análise mais criteriosa do projeto de Linux FailSafe revela que, ainda que seja possível conseguir-se as especificações e as descrições das interfaces (APIs) internas através do código fonte e de arquivos contendo documentação das APIs de Rhino disponibilizados na distribuição, os serviços internos de Linux FailSafe, tais como os serviços da camada CHAOS e CAS, não são explicitamente documentados e não parece ter sido esse exatamente o foco do projeto. Existe documentação para o desenvolvimento de *plug-ins* para que novas aplicações possam ser agregadas ao sistema, possibilitando que este possa gerencia-las. Desse modo, o projeto e implementação de aplicações distribuídas reais para esse *cluster*, utilizando sua própria infra-estrutura de serviços, fica deveras complicado. É largamente aceito que o projeto de aplicações distribuídas envolve o suporte de uma série de serviços específicos, dentre os quais muitos já se encontram na arquitetura de SGI Linux FailSafe. Assim, é natural que se cogite em utiliza-los para aplicações destinadas a rodar em um *cluster* Linux FailSafe, uma vez que tais serviços, sendo da própria arquitetura, são extremamente indicados pois têm uma visão ampla e concreta do *cluster*, seus recursos e estados. Mas dado o fato de que a documentação para as suas APIs é um tanto obscura, deduz-se que, ao menos até o momento, a utilização de tal produto deve ser restrita para *clusters* rodando aplicações tradicionais, não distribuídas e mono-instanciadas, comumente denominadas *non-cluster-aware applications*.

Por outro lado, a arquitetura e a descrição funcional de seus módulos e camadas revela um projeto que contempla as áreas necessárias para prover um ambiente completo para um moderno *cluster* de serviços. As áreas e

serviços disponíveis são suficientes para o desenvolvimento de aplicações para *cluster* em modalidade multi-instancia e com elevado grau de confiabilidade. As gerências e serviços existentes são aquelas que realmente se fazem necessárias e os detalhes e o tratamento conceitual revela um projeto bem estruturado. A implementação do sistema e de seus módulos sugere que foi utilizada uma abordagem bastante tradicional, em particular nos aspectos de comunicação inter-processo (IPC) utilizando onde soquetes do domínio *unix* (*Unix domain sockets*), *pipes* e, em alguns casos, soquetes do domínio de protocolos Internet para comunicação via protocolos TCP/IP entre os módulos. A utilização de algum grau de mecanismos de nível mais elevado para comunicação entre as partes do sistema, tais como RPC – *Remote Procedure Call* – teria sido muito bem vinda, pois, traria grandes benefícios para os programadores e desenvolvedores de aplicações para esse ambiente. Infelizmente utiliza-se comunicação via soquetes e protocolos proprietários entre os módulos, o que implica em uma maior dificuldade em se desenvolver aplicações. Felizmente existem disponíveis algumas bibliotecas que implementam interfaces de aplicação (API) para a camada de serviços básicos (CHAOS), porém a documentação das mesmas não é disponibilizada pela SGI, pelo menos até a presente data.

5.4 IBM HACMP™ e PSSP™

A IBM oferece produtos voltados para prover seus clusters de um ambiente de alta disponibilidade. Dentre eles temos o HACMP [CHIAKPO1996][CHIAKPO1997] e o PSSP [BEYER1995]. Na Figura 5-13 podemos ver um diagrama estrutural da arquitetura de serviços de suporte a alta disponibilidade oferecida. A seguir faremos uma breve descrição desses produtos e de suas arquiteturas.

Figura 5-13 A infra-estrutura de suporte para HA da IBM em seus clusters (reproduzido de [CHIAKPO1997])

5.4.1 IBM PSSP

O IBM PSSP – *Parallel System Support Programs* – ou Programas de Suporte para Sistemas Paralelos, consiste de um conjunto de programas, *daemons* e bibliotecas cuja finalidade é prover suporte para o desenvolvimento de aplicações paralelas distribuídas. Consiste dos seguintes módulos: *Topology Services*, *Reliable Messages Library*, *Group Services*, *Event Management* e *Problem Management*. Atualmente, a IBM também fornece esses módulos para seus *clusters* baseados em Linux sob o nome de RSCT – *Reliable and Scalable Cluster Technology* [FERREIRA2002].

5.4.2 IBM HACMP

O IBM HACMP – *High Availability Cluster Multiprocessing* – consiste de uma aplicação de gerenciamento de *cluster* e de recursos que, apoiada

nos serviços de PSSP, é capaz de oferecer um ambiente de *cluster* de serviços de alta disponibilidade, permitindo a definição de recursos, grupos de recursos, domínios de nós de serviços e políticas de migração e de *takeover* de serviços no *cluster*. De um modo simplificado e guardadas as devidas diferenças, pode-se dizer que HACMP e PSSP juntos compõem um sistema equivalente ao sistema SGI Linux FailSafe. Para maiores detalhes, vide a seção 5.3 . Nas seções seguintes faremos uma breve descrição dos principais módulos de PSSP / RSCT.

5.4.3 TS - Topology Services

O *Topology Services* (TS) - ou Serviços de Topologia, é um subsistema distribuído de PSSP/RSCT e compõe a camada a mais baixa da arquitetura de *clusters* IBM. A sua finalidade é manter e oferecer informação de disponibilidade a respeito dos nós e dos adaptadores da rede (adaptadores operando com endereços IP). O TS considera um nó operacional se este puder ser alcançado através de qualquer um de seus adaptadores seguindo qualquer um dos trajetos disponíveis. O serviço de topologia é executado como um *daemon* (denominado hatsd). Em todos os nós de um domínio de *cluster* (também chamado de *partição do sistema*) existe este *daemon* em execução (incluindo aí a estação de trabalho de controle (CWS). Estes *daemons* compartilham de um conjunto comum de definições e parâmetros. A CWS pode ter múltiplos *daemons* se houver mais de uma partição. Cada *daemon* na CWS comunicar-se-á e sincronizar-se-á com os outros *daemons* que funcionam na mesma partição do sistema. O TS deve fornecer informação de disponibilidade e de conectividade às camadas superiores do sistema (discutidas mais tarde). Entretanto, o TS é um subsistema distribuído, e assim a informação pode ser gerada em qualquer nó da partição, porém ela tem que ser a mesma não importa de qual nó os clientes do TS a obtenham (TS oferece consistência de informação). A

comunicação entre o TS e seus clientes se dá através de um soquete do domínio *Unix* (*Unix Domain Socket – UDS*).

5.4.4 RML - *Reliable Messages Library*

A, *Reliable Messages Library* - ou Biblioteca de Mensagens Confiáveis, é uma biblioteca que implementa um serviço de envio de mensagens no ambiente de *cluster*. Esta biblioteca é usada pelo Serviço de Grupo e pela Gerência do Eventos, enviando e recebendo mensagens de seus pares. Para utilizá-la, os *daemons* desses serviços ligam-se à biblioteca em tempo de execução e a utiliza para emitir e receber mensagens entre seus parceiros ao longo dos nós do *cluster*.

Figura 5-14 Os serviços de topologia (TS), de Grupos (GS) e a biblioteca de mensagens confiáveis em um *cluster* RSCT

Essa biblioteca subscreve-se como cliente dos serviços de topologia (TS) para receber informação de conectividade e assim manter uma tabela de conectividade da rede. Desse modo, ela é capaz de determinar caminhos de comunicação alternativos ou não-usuais entre dois nós quaisquer. Para

rotear mensagens por caminhos não-usuais na rede, a estratégia da biblioteca é inserir no campo de opções de seus datagramas IP informações de *source routing* quando necessário. Na Figura 5-14 podemos observar o relacionamento entre a biblioteca de mensagens e outros subsistemas de RSCT, tais como o Serviço de Grupos, o qual será brevemente descrito a seguir.

5.4.5 GS - *Group Services*

O serviço *Group Services (GS)* - ou Serviços de Grupo, são aqueles na camada seguinte da arquitetura de PSSP/RSCT. É um subsistema distribuído que fornece serviços para a sincronização e a coordenação de seus clientes. Os clientes conectam-se aos serviços através de uma API denominada *Group Service API (GSAPI)* ou API do Serviço de Grupo. O GS é um subsistema crítico que deve estar sempre ativo e rodando no *cluster* a fim de que outros subsistemas de RSCT/PSSP possam trabalhar corretamente. Os subsistemas que são clientes do GS incluem: A Gerência de Eventos, o subsistema de disco compartilhado virtual recuperável (RVSD), os sistemas de arquivos paralelos distribuídos (tais como GPFS), e indiretamente, a gerência de problemas, entre outros.

Os Serviços de Grupo de PSSP/RSCT compõem um *middleware* na arquitetura e funciona apoiado nos serviços de topologia (é um cliente dos serviços da topologia - TS), fornecendo serviços às camadas superiores tais como a gerência de eventos, entre outros. É composto de um *daemon* (denominado *hagsd*) e das bibliotecas de API denominadas GSAPI (cujos nomes são *libha_gs.a* e *libha_gs_r.a*). A interação entre uma aplicação e os serviços de grupo é feita unicamente através da GSAPI. Não é oferecida nenhuma interface de linha de comando ou interface gráfica de usuário para se lidar ou gerenciar os serviços de grupo. O GSAPI permite conexões locais somente através de soquetes do domínio *Unix*, conforme pode ser visto na Figura 5-14. Na Figura 5-15 temos uma visão da

interação entre o serviço de grupos, de topologia, da biblioteca de mensagens e podemos observar alguns desses serviços distribuídos como membros de grupos do GS.

Figura 5-15 O serviço de grupos (GS) de PSSP/RSCT (reproduzido de [CHIAKPO1996])

5.4.6 EM – Event Management

O *Event Management* (EM) - ou Gerência de Eventos, compõe a camada mais alta e mais externa desta arquitetura e tecnologia de *clusters*. A gerência de eventos é um dos clientes dos serviços de grupo. Ela define um grupo no GS denominado “ha_em_peers” a fim de manter a informações comuns de estado e de *membership* através dos nós. Ela utiliza também a biblioteca de mensagens confiáveis para uma comunicação *inter-daemon*. A gerência de eventos fornece um serviço de monitoração através de uma interface de programa de aplicação (API) denominada EMAPI. Através desta API, as aplicações (chamadas clientes do EM) podem pedir ao subsistema de eventos para monitorar circunstâncias específicas. A gerência de eventos monitora tais

circunstâncias e notifica seus clientes quando tais circunstâncias são encontradas ou atingidas.

A função do subsistema de eventos é comparar a informação sobre o estado de um recurso do sistema com a informação sobre as condições do recurso que são do interesse dos clientes do EM. Os clientes do EM podem incluir aplicações, subsistemas e outros componentes de *hardware* e de *software*. O estado de um recurso é representado por *variáveis* de recurso. As condições do recurso são representadas por *expressões*. A gerencia de eventos define o conceito de Monitores de Recurso. Monitores de recurso são entidades (geralmente programas) que observam o estado de recursos específicos do sistema e transformam este estado em diversas variáveis do recurso. Os monitores de recurso passam periodicamente estas variáveis ao *daemon* da gerencia de eventos. O *daemon* então aplica as expressões, as quais foram especificadas pelos clientes da EM, para cada variável de recurso sendo monitorada. Se a expressão for verdadeira, um evento é gerado e emitido ao cliente apropriado do EM. Os clientes do EM podem também solicitar ao *daemon* da gerencia de eventos sobre os valores atuais das variáveis de recurso. As variáveis de recurso, os monitores de recurso e outras informações relacionadas são contidos em diversas classes de objetos do chamado Repositório de Dados do Sistema (SDR). Essa informação, armazenada em classes do SDR, são traduzidas para um formato que pode ser facilmente usado pela gerência de eventos.

5.4.7 Uma análise do IBM HACMP e PSSP/RSCT

Feitas as breves descrições dos sistemas IBM HACMP e PSSP/RSCT nos tópicos acima, teceremos agora algumas considerações sobre os mesmos. Em primeiro lugar, tratam-se de sistemas maduros e já se encontram a um bom tempo no mercado de *clusters* de alta disponibilidade. Trata-se de produtos comerciais, de código proprietário, desenvolvido pela IBM e portanto, assim como SGI FailSafe, tem seu desenvolvimento e

direcionamento dirigidos por essa empresa. Uma análise das descrições do produto revela que ele destina-se a prover os *clusters* de estações de trabalho, de servidores RS6000 e de sistemas SP2, de um ambiente de alta disponibilidade para aplicações tradicionais e também para aplicações distribuídas, sejam elas aplicações de processamento de alto desempenho, sejam elas aplicações de alta disponibilidade ou mesmo aplicações transacionais de elevado *throughput*. Os sistemas PSSP e RSCT são na verdade ambientes de suporte para o desenvolvimento dessas aplicações, podendo oferecer a elas serviços que as tornam altamente resistentes a falhas de *software* e *hardware*, além de as prover com os serviços necessários para uma operação distribuída confiável. O HACMP é um produto para o gerenciamento de *cluster* IBM baseado na infra-estrutura de suporte de PSSP ou RSCT. Fica claro, portanto, qual é o foco desses sistemas e produtos. Analisando-se suas arquiteturas e seus aspectos e características externas, percebe-se que eles foram desenvolvidos como subsistemas de programas independentes. Porém, ainda que independentes, formam um conjunto fortemente integrado e coeso.

Uma análise mais cuidadosa do projeto e da apresentação desses produtos revela que a documentação sobre as APIs é extensa e rica em detalhes. Os serviços disponíveis nas APIs, os comportamentos e os estados possíveis dos módulos e *daemons* de serviço é fartamente documentado. As referências citadas neste trabalho, tais como [CHIAKPO1996], [CHIAKPO1997], [BEYER1995] e [FERREIRA2002], entre muitas outras, compõem uma grande fonte de informações sobre esses produtos e sobre suas APIs, permitindo ao desenvolvedor de aplicações distribuídas para os ambientes de *cluster* IBM ter todos os subsídios necessários para tal.¹⁷ Sendo assim, o projeto e implementação de aplicações distribuídas reais para esse *cluster*, utilizando sua própria infra-estrutura de serviços, é

¹⁷ Para maiores informações, consulte <http://www.redbooks.ibm.com/redbooks>

bastante facilitada e produtiva. Uma vez que os serviços oferecidos são suficientes, além de integrados à própria arquitetura do *cluster*, eles são extremamente indicados, pois tem uma visão ampla e concreta do *cluster*, seus recursos e estados. A arquitetura e a descrição funcional dos módulos de PSSP e RSCT e de suas camadas revela um projeto que contempla as áreas necessárias para prover um ambiente completo para um moderno *cluster* de serviços. As áreas e serviços disponíveis são suficientes para o desenvolvimento de aplicações de *cluster* em modalidade multi-instância com elevado grau de confiabilidade. As gerências e serviços existentes são aquelas que realmente se fazem necessárias e os detalhes e o tratamento conceitual revela um projeto bem estruturado. A implementação do sistema e de seus módulos sugere que foi utilizada uma abordagem, assim como para o SGI FailSafe, bastante tradicional, em particular nos aspectos de comunicação inter-processo (IPC) utilizados, onde soquetes do domínio *unix* (*Unix domain sockets*), *pipes* e, em alguns casos, soquetes do domínio de protocolos Internet para comunicação via protocolos TCP/IP entre os módulos. A utilização de algum grau de mecanismos de nível mais elevado para comunicação entre as partes do sistema, tais como RPC – Remote Procedure Call – novamente teria sido muito bem vinda, pois traria grandes benefícios para os programadores e desenvolvedores de aplicações para esse ambiente. Infelizmente utiliza-se comunicação via soquetes e protocolos proprietários entre os módulos, o que implica em uma maior dificuldade em se desenvolver certas aplicações. Felizmente existem disponíveis as bibliotecas que implementam interfaces de aplicação (API) para a camada de serviços básicos (PSSP/RSCT), cuja documentação é fartamente disponibilizada e com riqueza de detalhes.

5.5 Comparações com o DSMCA

Para encerrar este capítulo, faremos agora uma comparação das características do sistema DSMCA com os demais sistemas apresentados e

teceremos algumas considerações fundamentais. Destacaremos os pontos importantes e fundamentais da arquitetura de DSMCA que o diferenciam dos demais. Vamos procurar discutir e comparar aqui alguns dos pontos fracos e fortes das arquiteturas apresentadas ao longo deste capítulo e, ao mesmo tempo, faremos uma comparação e discussão destas com os pontos fracos e fortes, respectivamente, de DSMCA.

5.5.1 Comparação Arquitetural

Feitas as devidas descrições, ao longo dos tópicos anteriores, da arquitetura de outros trabalhos e sistemas relacionados existentes e feitas as descrições arquiteturais de DSMCA no capítulo 4, iniciemos então um comparativo entre a arquitetura do sistema proposto e a arquitetura dos trabalhos mencionados.

5.5.1.1 A Plataforma de Desenvolvimento

Este trabalho é uma proposta de uma arquitetura de gerenciamento e de suporte para aplicações distribuídas e convencionais para *clusters*, com um enfoque em *clusters* da classe *Beowulf*. A arquitetura elaborada recebeu a denominação DSMCA e na elaboração de sua arquitetura elegeu-se como premissa básica a utilização de uma plataforma baseada em objetos distribuídos. Desse modo, a OMG CORBA foi escolhida e adotada como solução, por ser talvez a mais largamente aceita, implementada e empregada. A escolha da OMG CORBA também se deu pelo fato de que as plataformas de desenvolvimento para essa arquitetura são diversas, contemplando muitas plataformas de *hardware* e de sistemas operacionais, o que contribui para que a arquitetura DSMCA possa no futuro ter sucesso como solução para *clusters* que empregam *hardware* e *software* diversos e largamente utilizados, tais como os ambientes *Linux* e *Microsoft* e computadores com processadores de arquitetura Intel x86. Assim, destaca-se aqui a preocupação inicial com a arquitetura do sistema DSMCA no

sentido de facilitar sua implementação em ambientes diversos e heterogêneos. Na verdade, essa é uma das premissas da arquitetura OMG CORBA, que é a de prover interoperabilidade de aplicações em ambientes heterogêneos.

Ao optar-se pela arquitetura CORBA como plataforma arquitetural de objetos distribuídos para o sistema proposto, naturalmente tem-se que a arquitetura de DSMCA é orientada a objetos e esse fato por si só estabelece desde o início que os serviços e funcionalidades do sistema serão todos eles acessíveis através de chamadas a métodos de objetos de serviço disponíveis e acessíveis através de um ORB CORBA. Uma consequência direta dessa escolha arquitetural é a de que o desenvolvimento de aplicações para rodar em *clusters* DSMCA passa necessariamente por dotar a aplicação de capacidades de se comunicar com objetos em um ORB CORBA. Na verdade, essa não é uma exigência estrita, uma vez que muitas aplicações de *cluster*, em particular as aplicações tradicionais mais simples, não necessitam na maioria dos casos serem aplicações *cluster-aware*, ou seja, elas não precisam necessariamente ter ciência de ou estar conectadas ou em comunicação com o sistema de gerenciamento do *cluster*. Isso pode decorrer meramente do fato de que a aplicação e seus serviços possuem características que dispensam essa necessidade, ou, porque a própria aplicação implementa internamente mecanismos de comunicação inter-instâncias que dispensam o suporte dos serviços do *cluster*. Entretanto, uma aplicação do *cluster* que mantém contato com a gerência do mesmo, estando capaz tanto de solicitar serviços da mesma quanto de receber notificações e informes sobre eventos relevantes sobre o ambiente onde opera, certamente será uma aplicação capaz de adequar-se mais rapidamente a mudanças no ambiente e desse modo pode ter uma sobrevivência mais longa. Isso vai diretamente de encontro a uma segunda premissa deste projeto: a alta disponibilidade de serviços. DSMCA visa prover e promover, também

como premissa básica, um ambiente de alta disponibilidade de serviços através de alta integração e conectividade entre ele e as aplicações que dele se utilizam. Internamente, DSMCA é estruturado de modo a ser ele próprio uma plataforma de suporte extremamente robusta, confiável e estável, atingindo graus de disponibilidade elevados. Essa necessidade é básica se ele se propõe a ser o suporte para a alta disponibilidade e robustez de outras aplicações.

Neste aspecto, fazemos a comparação possível com os outros sistemas apresentados. O *cluster* HAL, proposto pelo projeto *High-Availabiltiy Linux* [HAL] é uma proposta simples e de baixo custo, onde componentes diferentes e de origens diversas são agregados como os componentes de gerenciamento do *cluster*. Compões-se de um conjunto modular composto de uma série de programas, *daemons* e módulos de *kernel* Linux que, arranjados e configurados de certa maneira, cumprem algumas das funções básica de um *cluster* com certas características de alta disponibilidade e desempenho. Estes módulos tem pouca ou nenhuma ciência dos demais módulos com os quais interoperam ou colaboram, e desse modo a arquitetura é extremamente falha nesse aspecto. Já os *clusters* com as arquiteturas SGI *FailSafe* e IBM HACMP e PSSP/RSCT são compostos de subsistemas e módulos fortemente integrados e cientes da integração e inter-operação entre eles. Desse modo, tais sistemas cumprem o requisito básico para terem robustez e confiabilidade no ambiente de *cluster*.

5.5.1.2 Interfaces Para Desenvolvimento de Aplicações

Um aspecto importante da adoção da arquitetura OMG CORBA, como o *middleware* de integração dos módulos e de acessibilidade aos serviços de DSMCA, diz respeito às especificações das interfaces voltadas para o desenvolvimento de aplicações, ou APIs. OMG CORBA é fortemente apoiada na utilização da linguagem IDL, cuja principal finalidade é a

definição de interfaces de classes de objeto. A IDL – *Interface Definition Language* permite que, de uma forma simples e direta, os métodos, propriedades, parâmetros e tipos de dados concernentes à interface de um objeto sejam declarados. A partir das definições IDL de uma interface, compiladores IDL são capazes de gerar código para que programas possam facilmente acessar um objeto possuidor de tal interface. Via de regra, esses compiladores são capazes de gerar também código para que objetos detentores de uma interface igual à da definição IDL possam ser implementados. O primeiro tipo de código gerado, aquele que pode acessar uma interface de objeto, é chamado de código *cliente* do objeto, e assim o código gerado pelo compilador é freqüentemente denominado *client stub*. O segundo tipo de código gerado, aquele que permite a implementação de um objeto e de sua interface, é freqüentemente denominado *server skeleton*. Quando um *server skeleton* é completado com código que implementa a lógica, estado e comportamento do objeto, temos um *object server*, um programa executável servidor de objetos. Os compiladores IDL geram também código para que tanto *client stubs* quanto *server skeletons* sejam capazes de se comunicar através do ORB CORBA. Temos, portanto, que dada a definição de uma interface IDL, as dificuldades de se criar código para acessar um objeto CORBA possuidor de tal interface são poucas e o processo é relativamente simples, além do que os compiladores IDL são capazes de gerar código em diversas linguagens de programação diferentes, tais como C, C++, Java e Pascal. Portanto, ter acesso a serviços implementados como objetos CORBA é simplesmente uma questão de se ter à mão a definição IDL das interfaces desses objetos, além de se ter o suporte de um compilador IDL de preferência, o qual a maioria dos ambientes de desenvolvimento CORBA fornecem.

Neste aspecto, fazemos as comparações cabíveis aos outros sistemas apresentados. No caso do *cluster HAL*, toda e qualquer interação das

aplicações que rodam no *cluster* no sentido de buscar serviços dos seus módulos de gerenciamento, basicamente se resumem a comandos e linhas de *Shell script* que, de algum modo, conseguem obter ou causar alguma mudança em alguns de seus aspectos de comportamento ou no seu estado. No caso do *cluster* SGI FailSafe, a interação das aplicações com os módulos de gerenciamento e de serviços básicos de *cluster* tais como as camadas CHAOS e CAS, certamente pode se dar de maneira mais consistente e robusta, permitindo a construção de aplicações confiáveis. Existe boa documentação para a construção de módulos *plug-in* baseados em *Shell script* a fim de dar suporte às aplicações que se deseja executar no *cluster*. Esses *scripts* visam criar a interface de interação entre o gerenciamento do *cluster* e a aplicação em questão. Porém, tal estratégia não é das mais robustas. Como já mencionamos, uma análise do código mostrou que existem bibliotecas que implementam API's para acesso aos serviços de CHAOS e CAS, porém notou-se a ausência de documentação das chamadas dessas API's. Acreditamos que a SGI deve liberar tal documentação em breve. Em comparação com os produtos de *cluster* da IBM, as interfaces para o desenvolvimento de aplicações são disponíveis, porém nem todas elas são documentadas. As API's de todos os módulos subsistemas de PSSP/RSCT são implementadas na forma de bibliotecas de rotinas, as quais devem ser ligadas em tempo de compilação ao programa interessado nos serviços da biblioteca. Essas bibliotecas provêm à aplicação uma comunicação direta com o programa *daemon* servidor que realiza e fornece o serviço. Via de regra, essa comunicação ocorre através de soquetes de comunicação do domínio *Unix*. Em alguns casos, quando os clientes de um *daemon* podem ser remotos, são disponibilizados também soquetes de comunicação do domínio de protocolos Internet. A comunicação entre uma aplicação e o *daemon* diretamente através de um soquete é possível, porém não há documentação aberta sobre os protocolos utilizados. Analisando-se a documentação disponível sobre as API's, nota-se que as mesmas são bem arquitetadas e flexíveis, permitindo a

construção de aplicações de modo relativamente simples. A documentação IBM foca em particular as API's dos serviços de grupo (*Group Services*), dos serviço de eventos (*Event Services*) e das API's necessárias para programas gerenciadores de recursos, os quais podem ser construídos para se agregar ao sistema novos tipos de recurso. Essa é uma estratégia similar à utilizada por SGI FailSafe e seus *plug-ins* para permitir novos recursos e aplicações. Essas API's são em geral suficientes para o desenvolvimento da maioria das aplicações distribuídas de alta disponibilidade para *cluster*.

5.5.1.3 Interoperabilidade Entre Sistema e Aplicações

Uma dos impactos decorrentes da adoção de CORBA como plataforma para o desenvolvimento dos módulos de DSMCA é o fato de que isto implica que muitas das aplicações desenvolvidas para esse ambiente deverão não só implementar código de cliente para os objetos de serviço como também código de servidor. Isto equivale dizer que certas aplicações deverão, no seu desenvolvimento, utilizar-se de código *server skeleton* gerado por um compilador IDL a fim de se integrar completamente ao ambiente. Esse código *server skeleton* deverá ser referente aos chamados objetos de *callback*, ou objetos de cliente, do serviço. Nas definições dos módulos de interface de DSMCA, as interfaces desses objetos de cliente são comumente denominadas de *Client*. Uma aplicação implementa um objeto *Client* se esta necessitar estar completamente integrada ao ambiente de DSMCA. A interface *Client* em geral permite que o DSMCA notifique a aplicação sobre a ocorrência de eventos relevantes ou da chegada ou mudança de estados e informações pertinentes. Um exemplo típico de aplicação que deverá implementar uma interface *Client* é aquela que deseja tornar-se cliente do serviço de monitoria de nós (*Cluster Node Monitor*). Esse serviço tem por finalidade notificar seus clientes sobre quaisquer mudanças no *membership* de nós do *cluster*. Ainda que seja possível ao cliente de tal serviço obter a informação sobre o *membership* corrente de forma explícita e síncrona, a interface *Client* permite que uma

mudança no *membership* de nós do *cluster* seja notificada tão logo ela se concretize, de modo assíncrono. Um outro caso semelhante e auto-explicativo é quando uma aplicação deseja tornar-se membro de algum grupo do *Group Services*, pois nesse caso é imprescindível que a aplicação implemente um objeto *Client* do grupo a fim de participar das atividades do mesmo. Um outro impacto dessa decisão arquitetural é a de que os objetos de cliente e de serviço devem estar conectados entre si. Na terminologia CORBA, objetos encontram-se conectados quando ambos possuem referências um do outro. Assim, os módulos de definição de serviços de DSMCA sempre definem uma interface-base denominada *Connected* onde alguns métodos simples e básicos são definidos para permitir que objetos *Client* e *proxies* do serviço estabeleçam conexão mútua, bem como métodos para permitir uma desconexão ordenada entre eles quando um ou outro não mais desejar estar conectado.

Quando as API's para clientes de um serviço são implementadas na forma de bibliotecas, como é o caso de SGI FailSafe e IBM PSSP/RSCT, a ligação entre provedor e cliente de serviço estabelece-se quando alguma rotina da biblioteca é chamada, usualmente alguma rotina com uma semântica e nome similar a *init()*. Em seguida, se a interação deve ser bidirecional, rotinas para o estabelecimento de funções de *callback* devem ser chamadas. Como parâmetros, essas rotinas esperam referências locais aos endereços das rotinas do cliente. Desse modo, permite-se que rotinas da biblioteca possam chamar funções no aplicativo cliente. O estabelecimento de um canal de interação de mão dupla como este ou como o mecanismo de objetos e interfaces de *callback Client* de DSMCA garantem um mecanismo limpo e eficiente para tal. Entretanto, no caso das API's baseadas em bibliotecas, fica a cargo do código da aplicação permitir que a biblioteca estabeleça chamadas às suas rotinas de *callback*. Como exemplo, o serviço de grupos de PSSP exige que o aplicativo cliente chame a rotina *gs_dispatch()* quando este deseja verificar se há

alguma mensagem do grupo pendente. Essa rotina pode ser chamada em modo bloqueante ou não-bloqueante, dependendo de um de seus parâmetros. Na modalidade não-bloqueante, a rotina retorna imediatamente, porém a aplicação deve realizar periodicamente sua chamada. Em contraste, a construção de uma aplicação onde a interface de *callback* é implementada através de um objeto CORBA possibilita de um modo relativamente simples que chamadas assíncronas ocorram nesse objeto sem que a aplicação tenha que realizar chamadas em alguma rotina de verificação. O ORB CORBA e o objeto de *callback* do cliente podem ser colocados para rodar em linhas de execução (*threads*) independentes do código principal da aplicação. Para uma aplicação baseada em uma interface gráfica de usuário (GUI) este é um aspecto bastante interessante uma vez que neste tipo de programa é importante que não existam pontos de bloqueio do código, sob risco de haver congelamento da interface gráfica. Ainda que isto seja possível para uma aplicação cliente de uma biblioteca de API que realiza *callbacks*, o fato de que a criação de linhas de execução independentes para os objetos em um servidor CORBA ser um aspecto praticamente “*default*” mostra que a decisão arquitetural por CORBA é uma boa escolha ao facilitar o desenvolvimento de aplicações com as características aqui apontadas.

5.5.1.4 Modelo de Comunicação Inter-Processos

Em todos os sistemas de *cluster* analisados até aqui, verificamos que diversas formas de comunicação inter-processos foram utilizadas a fim de prover comunicação entre os módulos do sistema e de prover comunicação entre os serviços de *cluster* e os clientes. Cada uma delas atendendo os requisitos das partes comunicantes e do tipo de informação sendo transmitida. Faremos agora uma análise dos aspectos de comunicação utilizados nos mesmos. Primeiramente, dos mecanismos de comunicação inter-nós (externos).

Comunicação Inter-Nós (Externa)

No *cluster* HAL, o módulo de *heartbeat* realiza tanto o envio de mensagens do seu protocolo diretamente através das portas seriais quanto por empacotamento UDP para envio através de adaptadores de rede local. Nos *clusters* SGI *FailSafe* e nos *clusters* IBM PSSP / RSCT, os serviços de *membership* (CMS) e de topologia (TS) utilizam-se também de mensagens UDP para realizar suas tarefas. A razão da escolha desse protocolo está no fato de que se trata de um protocolo não-orientado à conexão (*connectionless*) e sem reconhecimento (*acknowledge*) e de baixo *overhead*. Essas características do UDP o tornam ideal para a tarefa de manter o monitoramento de adaptadores e de nós do *cluster*, haja visto que esta é uma área crítica para a alta disponibilidade de serviços. Uma vez que as mensagens de monitoramento são freqüentes, um protocolo leve, sem conexão e reconhecimento alivia o tráfego de mensagens na rede e possibilita também que canais de baixa velocidade possam ser utilizados, tais como as interfaces seriais. Uma das razões para o uso de um protocolo sem reconhecimento é a de que as mensagens de *heartbeat* não necessitam de tal confirmação, pois elas apenas informam ao monitor (destinatário) o estado operacional ativo de quem as envia. Caso uma mensagem de *heartbeat* se perca não é necessário reenviá-la. O nó monitor aguarda que, no próximo ciclo, uma nova mensagem de *heartbeat* seja recebida do seu nó monitorado. O nó monitor considera que o nó monitorado está com problemas se durante um certo número de ciclos não surgirem mais mensagens de *heartbeat*. O monitor pode realizar alguma espécie de teste explícito com o monitorado antes de considerá-lo morto. O protocolo dos serviços de monitoria de adaptadores e de nós das arquiteturas mencionadas são ricos em tipos de mensagens, não havendo apenas mensagens de *heartbeat*. A mensagem de *heartbeat* é somente aquela que é utilizada com maior freqüência. Outras mensagens existem e servem para que todos os monitores de um grupo de nós tenham uma visão

consistente do grupo em qualquer circunstancia. Portanto, mensagens protocolares visando estabelecer semânticas transacionais existem. A estratégia mais utilizada é a de realizar votações e estabelecimento de consenso sobre o *membership* de nós e adaptadores através da técnica de *two-phase-commit*, de modo similar às técnicas já extensamente abordadas em (3.3.6). Outras mensagens existem, tais como as utilizadas por um novo nó para solicitar junção ao grupo, para um nó solicitar saída do grupo, para um nó proclamar liderança do grupo, para um monitor reportar a morte de um membro, entre outras. Todas essas mensagens podem ser enviadas na forma de *datagramas* sem confirmação, uma vez que a maioria delas espera apenas que os outros nós respondam com mensagens adequadas e, caso não haja resposta por um certo tempo, simplesmente reenvia-se a mensagem. A comunicação segue portanto um padrão simples de pedido-resposta sem confirmação explícita de entrega nem do pedido nem da resposta pelo serviço de comunicação.

O outro serviço que realiza comunicação externa nessas arquiteturas é o serviço de mensagens. Ainda que ele não seja um serviço visível no *cluster* SGI FailSafe, ele o é na arquitetura IBM PSSP/RSCT na forma da biblioteca de mensagens confiáveis (RML). Novamente, temos um serviço que utiliza-se do protocolo UDP para realizar suas tarefas. A partir dos mapas topológicos dos serviços de monitoria de nós (CMS) e de topologia (TS), esses serviços enviam mensagens de nó para nó permitindo o endereçamento de clientes distintos. Eles são chamados de serviços de mensagens confiáveis porque implementam um protocolo de confirmação de entrega de mensagens com características apropriadas ao serviço em questão. Eles realizam tais tarefas apoiados apenas nos serviços do protocolo UDP. Esse protocolo consiste de mensagens simples para envio, confirmação de entrega e reporte de problemas. Esse último tipo de mensagem consiste de notificações de problemas caso as mensagens não puderem ser entregues. Dentre as características desejadas, esse protocolo

deve (a) ser robusto no sentido de conseguir entregar mensagens em um nó não diretamente acessível, utilizando caminhos alternativos e não-usuais, e (b) prover opcionalmente confirmação de entrega ou não-entrega.

Todos os demais serviços que necessitam comunicação externa no *cluster* via de regra apóiam-se no serviço de mensagens de sua arquitetura particular. Uma questão desafiadora para o projeto de DSMCA surgiu ao se constatar que, como padrão para a comunicação entre objetos de um ORB CORBA quando esses ORBs residem em computadores distintos, o protocolo utilizado é o IIOP – *Internet Inter-Orb Protocolo*, o qual apóia-se no protocolo TCP para realizar seus serviços. Como se sabe, TCP é um protocolo de transporte orientado à conexão e dotado de mecanismo de reconhecimento de entrega (confirmação), e desse modo é um protocolo que exige maior carga de trabalho tanto dos equipamentos computadores quanto dos equipamentos e canais da rede, pois tanto o volume de informação que gera é maior quanto suas máquinas de protocolo são mais complexas. Ao se imaginar a implementação de um serviço de monitoria de adaptadores, nós e de topologia para o *cluster* DSMCA utilizando-se IIOP como protocolo de comunicação entre os ORBs nos computadores do *cluster*, constata-se que o resultado pode ser um serviço ineficiente do ponto de vista computacional e de comunicação. Felizmente, a partir da versão 2.3 de CORBA o OMG publicou um padrão para utilização do protocolo UDP como IOP (*Inter-Orb Protocol*). Desse modo, surgiram recentemente implementações de CORBA dotadas desse protocolo em seus módulos de comunicação, em particular daquelas que já adotam o modelo aberto de comunicação OCI – *Open Communication Interface* para seus ORBs. Uma das restrições da utilização de tal protocolo é a de que os métodos das interfaces de objetos CORBA devem ser declaradas com a diretiva *oneway* e ter *void* como seu tipo de valor de retorno, pois faz-se necessário haver uma conexão protocolar estabelecida entre o cliente e o servidor de objeto para que invocações a métodos possam

retornar valores. Uma vez que UDP não é um protocolo orientado à conexão, ficam proibidos os métodos de interface que exigem valores de retorno. A tentativa de se executar chamadas a métodos que retornam valores utilizando-se UDP/IOP ergue uma exceção de sistema para a aplicação. Felizmente, sabe-se que a realização dos serviços externos pode ser feita perfeitamente a partir de um protocolo simples de *datagrama*, operando com um paradigma de pedido-resposta. Desse modo, consegue-se especificar através de IDL um conjunto de interfaces de objetos que realizam os serviços de monitoria de adaptadores, nós e podem realizar o serviço de mensagens confiáveis, bastando que seus métodos tenham as diretiva *oneway* e sejam do tipo *void*. Os módulos *Adapter*, *AdapterServices*, *Message* e *MessageService* de DSMCA possuem interfaces denominadas *Service* que tem apenas métodos com essas características, e podem ser usadas para a implementação CORBA dos serviços externos.

Comunicação Intra-Nós (Interna)

Nos sistemas de *cluster* vistos até aqui, a comunicação interna entre os diversos módulos do sistema se dá via de regra através de mecanismos de IPC locais, tais como *pipes* (UDS) mapeados no sistema de arquivos (nomeados) ou não, arquivos regulares, memória compartilhada (SM), sinais e semáforos *mutex*. Dependendo dos módulos comunicantes e das necessidades de comunicação, um destes mecanismos é escolhido. Por exemplo, o serviço de monitoria de adaptadores e topologia de PSSP/RSCT utiliza dois mecanismos de comunicação com seus clientes: UDS e SM. Para os clientes que desejam uma interação síncrona com o serviço ou que necessitam apenas de notificações sobre alterações no seu estado, um soquete UDS é suficiente. Para os clientes que necessitam estar continuamente acessando as informações mais recentes sobre conectividade, o uso de memória compartilhada é mais indicado por ser

mais eficiente e de muito menor *overhead*. Assim, os serviços de grupo utilizam um UDS para receber notificações do TS. Já o serviço de mensagens confiável utiliza SM para ter acesso contínuo à informação de conectividade. Nota-se, portanto, que a escolha dos mecanismos de IPC são pautadas por uma adequação à situação específica.

Na elaboração da arquitetura de DSMCA existiu uma preocupação com os aspectos de comunicação interna entre os objetos do ORB CORBA. Algumas implementações de CORBA são capazes de prover automaticamente comunicação eficiente entre os seus objetos ao explorar a localidade dos mesmos. Quando objetos encontram-se em um mesmo espaço de memória, alguns ORBs redirecionam as chamadas entre esses objetos de modo que a comunicação não atinja a camada de IOP e se faça de modo direto. Esse procedimento de redirecionamento faz com que a eficiência das chamadas seja de um nível comparável à chamadas de métodos locais em objetos não-CORBA. Quando objetos encontram-se em processos distintos em um mesmo sistema computador, alguns ORBs são capazes de criar canais de comunicação entre os processos utilizando mecanismos de IPC locais tais como *pipes*. Isto faz com que a eficiência da comunicação seja elevada entre os objetos. Entretanto, algumas implementações de ORB CORBA tem apenas a capacidade de realizar essa comunicação através de mecanismos de IPC de maior *overhead*, tais como soquetes TCP/IP, mesmo havendo espaço para se explorar a localidade. Nesses casos, a melhor opção é direcionar a comunicação através de uma interface de *loopback*. O aspecto interessante de CORBA para a comunicação entre seus objetos é que, independentemente do mecanismo de IPC (ou, em última análise, do protocolo IOP) a implementação do objeto não se altera em nenhum aspecto. Também, é possível a um objeto CORBA estar ativamente recebendo chamadas através de diversos tipos distintos de mecanismos de IPC ou protocolos de IOP. A implementação desse objeto é indiferente em relação ao IOP que

proverá acesso ao mesmo. Desse modo, objetos de serviço podem ser implementados tanto para receber pedidos de clientes locais quanto remotos. Surge aqui a questão da confiabilidade: se um objeto pode ser acessado remotamente de qualquer outro nó do *cluster*, porque a razão da existência do serviço de mensagens confiáveis? A razão é simples: os protocolos IOP para CORBA, tais como o IIOP, apóiam-se nos serviços de TCP ou de UDP, que por sua vez apóiam-se nos serviços de IP. O protocolo IP trabalha com regras de roteamento simples, as quais não são capazes de lidar com particionamentos anormais das redes no nível de enlace, os quais podem ocorrer por acidente ou durante situações contingenciais. O serviço de mensagens confiável, por sua vez, utiliza-se de mapas topológicos que refletem os particionamentos das redes no nível de enlace. Assim, consegue manter comunicação entre parceiros mesmo em situações muito adversas, uma vez que esse serviço utiliza-se de nós intermediários entre as partições das redes a fim de obter conectividade. Para os serviços que são críticos para o *cluster*, tais como as gerências de serviços distribuídos e serviços de grupos, manter comunicação entre suas partes, mesmo que com baixa eficiência, é um aspecto extremamente importante e relevante na questão de alta disponibilidade e confiabilidade. Entretanto, um cliente de um serviço remoto pode, a seu critério, dispensar os serviços de mensagens confiáveis existente e estabelecer diretamente, através do ORB CORBA, uma conexão com o objeto provedor do serviço em outro nó, apoiando-se apenas nos protocolos de IOP. Nestes casos, o serviço do objeto remoto não deve ser crítico, ou seja, a perda de comunicação com o mesmo pode ser tolerada ou contornada.

5.5.2 Comparação Funcional

Feitas as devidas comparações arquiteturais, passemos agora a comparações das características funcionais.

5.5.2.1 *Acessibilidade aos Serviços*

Uma das características mais importantes a respeito de um sistema de gerenciamento de *cluster* diz respeito às capacidades de acesso aos diversos serviços existentes no mesmo por aplicações externas ou de novos serviços que se deseja agregar ao sistema. Os sistemas já apresentados neste capítulo, tais como o SGI FailSafe e o IBM HACMP e PSSP/RSCT mencionados nos tópicos (5.3) e (5.4), respectivamente, permitem que aplicações externas tenham acesso a apenas alguns de seus serviços. Certos serviços são considerados como internos e, ainda que possuam APIs que em princípio poderiam ser utilizadas para acesso de outras aplicações, não são documentadas e possivelmente isto se dá como modo de desestimular essa tentativa. Uma possível razão para adoção de tal política pode estar no fato que as interfaces e serviços internos dos mesmos não são projetadas para tal finalidade. Podem, em alguns casos, haver restrições quanto ao número de clientes utilizando-as simultaneamente, chegando a ser possível que algumas interfaces entre módulos sejam de uso exclusivo de apenas um único cliente. Um caso que exemplifica tal situação é o dos serviços de topologia de PSSP/RSCT, onde aparentemente o único cliente possível através do soquete UDS parece ser o serviço de grupos. Uma outra área onde aparentemente os serviços são pouco visíveis a aplicações externas reside nas gerências de recursos, serviços e de *cluster*, as quais são implementadas por HACMP nos *clusters* da IBM. A interface entre HACMP e aplicações externas se dá, assim como faz SGI FailSafe, através de *user exit scripts*.

A razão da discussão sobre as possibilidades de acesso aos diversos serviços da infra-estrutura de serviços de gerenciamento do *cluster* apóia-se na crença de que uma arquitetura que se proponha a ser também suporte para aplicações deve prover acesso a qualquer um de seus serviços. Logicamente, uma aplicação que se proponha a ter acesso a certos serviços deve ser escrita por um programador que conheça bem suas interfaces e

seus aspectos comportamentais. Também, aspectos de segurança do *cluster* e de seus sistemas devem ser levados em conta uma vez que nem todos os serviços podem estar abertos a qualquer usuário indiscriminadamente. Entretanto, havendo a possibilidade de acesso aos serviços e havendo também mecanismos previamente planejados para esse fim, o sistema torna-se extremamente flexível e extensível. Se for desejável que um novo tipo de serviço seja agregado ao sistema de gerenciamento e suporte do *cluster*, havendo pontos para permitir essa conexão com forte integração, esta tarefa torna-se facilitada. É por essa razão que todos os módulos de DSMCA são módulos de serviço com interfaces definidas por IDL e que têm a características multi-cliente. O modelo adotado para tal funcionalidade foi o mesmo modelo adotado pelo serviço de eventos de CORBA, publicado e padronizado pelo OMG como o *Corba Event Service* [ORBEvent]. Nesse modelo, um cliente do serviço solicita à interface administrativa do serviço um objeto representante do serviço (*proxy*) para sua comunicação exclusiva com ele. Em geral, o serviço cria um objeto *proxy* exclusivo e único para cada novo cliente. Nas seqüências da Figura 5-16, Figura 5-17, Figura 5-18 e Figura 5-19 podemos acompanhar o processo em que dois clientes de um serviço conectam-se à ele através da solicitação de *proxies* do serviço. Essa estratégia ajuda na implementação do serviço, pois mantém automaticamente uma relação de objetos distintos identificados com cada um de seus usuários. Um outro aspecto interessante da estratégia de uso de *proxies* é o fato de que, para muitas implementações de serviço CORBA, cada objeto de serviço executa em uma linha de execução distinta (*thread*) e assim a solicitação do serviço através de *proxies* facilita a busca por melhoria de desempenho ao fazer com que o serviço possa disponibilizar automaticamente uma *thread* para cada cliente.

Figura 5-16 Objeto-cliente solicita *proxy* para conexão ao serviço DSMCA

Figura 5-17 Objeto-cliente conecta-se ao *proxy* de serviço e passa a fazer solicitações.

Figura 5-18 Um segundo objeto-cliente faz também solicitação por um *proxy*.

Figura 5-19 O segundo objeto-cliente conecta-se ao *proxy* do serviço.

5.5.2.2 Serviços de Evento

Uma das características que diferenciam DSMCA dos demais sistemas é a extrema estruturação da área de eventos do sistema. Em sistemas como o PSSP/RSCT da IBM, a área de eventos é implementada através do serviço *Event Management* (EM), a qual apóia-se no serviço de grupos, o *Group Services* (GS) e em programas monitores de recursos, os *Resource Monitors* (RM). Nesse sistema, um monitor de recurso é um programa capaz de monitorar uma ou mais *variáveis de recurso* (*Resource Variable*). Uma variável de recurso representa um atributo, propriedade ou valor de um recurso qualquer do sistema que pode ser lido ou acessado pelo programa monitor. Como exemplo, uma variável de recurso pode representar o tamanho de um sistema de arquivos, o espaço livre em blocos do sistema de arquivos, o número de processos ativos em um nó do *cluster*, o tamanho do espaço de *swap* disponível em um certo nó, e assim por diante. Uma variável de recurso possui uma *classe* e está associada a um único monitor de recurso. Se uma variável de recurso possui múltiplas instâncias, ela tem também um atributo de instância. Um monitor de recurso pode implementar várias classes de variáveis e várias variáveis por classe. Com base nessas definições, o serviço *Event Management* de PSSP/RSCT pode ativar monitores de recurso a fim de que o valor de suas variáveis de recurso estejam a ele disponíveis através da API padronizada do monitor. O serviço permite que seus clientes registrem predicados associados às variáveis de recurso. Sempre que o predicado de uma variável de recurso se torna verdadeiro, o serviço emite um evento ao interessado. O *Event Management* é um serviço distribuído de PSSP/RSCT e apóia-se no GS para operar consistentemente.

Ao se comparar com DSMCA, logicamente notamos que a funcionalidade provida pelo *Event Management* do PSSP/RSCT é existente também em DSMCA através do serviço *Remote Monitoring*. De um certo modo, esse serviço DSMCA cumpre o mesmo papel que o *Event Management* da

IBM, porém, a estrutura elaborada em DSMCA apresenta diferenças. Em primeiro lugar, o conceito de variável de recurso dá lugar ao gerenciador de recurso (*Resource Manager*) e às suas diversas interfaces de gerenciamento de recursos e de propriedades de recurso. Os recursos de DSMCA são fortemente tipados e hierarquizados, tendo associados a si os atributos de *categoria*, *tipo* e o seu *gerenciador* propriamente. Além disso, recursos possuem também outros atributos que os classificam, tais como: a área gerencial onde atuam, a *classe* de gerenciamento que realizam, o *cluster* onde existem, entre outras. Essas diferenciações são denominadas *níveis* de recurso. Um gerenciador de recurso DSMCA pode implementar um número qualquer de recursos, de diversos tipos e categorias. Em DSMCA, os níveis de diferenciação e hierarquização de recursos também são considerados recursos. Assim, um *cluster* é um recurso, uma *categoria* é um recurso, assim como um gerenciador de recurso também é visto como um recurso. O gerenciador de recurso tem uma atuação gerencial completa sobre seus recursos, podendo acessá-los, monitorá-los, configurá-los e controlá-los. Em contraste, os monitores de recurso de PSSP/RSCT permitem apenas ações de monitoração de variáveis de recurso, ficando o controle e a configuração de recursos a cargo dos *scripts* disparados por HACMP e por outros programas do sistema, tais como o *SRC – System Resource Controller*.

Feitas essas considerações, prosseguiremos agora abordando o tratamento de eventos nos sistemas citados. No sistema DSMCA, todo recurso é gerenciável através de uma ou mais interfaces de gerenciamento implementadas por objetos dentro de seu gerenciador de recurso. Essas interfaces apresentam métodos para que se possa solicitar objetos *geradores de eventos* do recurso. Tais objetos, quando solicitados, são automaticamente conectados a objetos *encaminhadores de eventos* do gerenciador de recurso, de modo que os eventos gerados sigam para as áreas do sistema responsáveis por processá-los e tratá-los. Os objetos

geradores de evento podem ser instruídos sobre como e/ou quando eventos do recurso devem ser gerados. Uma das possibilidades é a especificação de *predicados* para um atributo ou propriedade do recurso. Quando o predicado torna-se verdade, um evento é gerado. Os predicados podem referenciar na sua expressão valores históricos do recurso em si e também a valores obtidos através de definições de *metadados* do sistema. Assim, o recurso pode ser, por exemplo, comparado com o valor de outro recurso ou de outro dado disponível no sistema. No sistema IBM PSSP/RSCT, a definição de predicados é similar, porém notemos que em tal sistema, a avaliação de predicados é função do *Event Manager*. Já no sistema DSMCA, é o próprio gerenciador do recurso que deve implementar sob demanda os objetos de geração de eventos do recurso. Parte-se do pressuposto que esse elemento conhece melhor e está mais próximo do recurso, muitas vezes sendo ele próprio a implementação do recurso, e assim pode de um modo mais eficiente observar e monitorar o recurso, gerando eventos quando necessário. Os eventos gerados por um gerenciador de recursos DSMCA formam um fluxo que sobe pela árvore hierárquica de recursos interna ao gerenciador através de encaminhadores em direção à sua raiz. De lá, o encaminhador de eventos repassa o evento para o *hub* de recursos de DSMCA, o qual concentra e encaminha os eventos de todos os recursos. O caminho dos eventos é, naturalmente, para um serviço especializado no processamento e tratamento de eventos, o *Resource Event Selector* do DSMCA. Esse serviço pode prover seus clientes, em número qualquer, de eventos provenientes do *hub*, desde que estes manifestem qual o tipo de evento e a origem (recurso, nível, área, etc.) do mesmo. O *Resource Event Selector* do DSMCA permite que seus clientes solicitem *seletores* de evento e estes, por sua vez, permitem que se possa especificar as características (tipo) dos eventos de interesse. Temos então que, neste nível, eventos são recebidos e selecionados, de acordo com as preferências dos clientes do serviço. Notar que, até este ponto, estamos lidando com serviços locais, ou seja, são serviços exclusivos de

um nó do *cluster*. De modo que é possível para um cliente apenas de eventos de recursos locais de um nó solicitar que estes lhe sejam enviados, seletivamente, utilizando-se do serviço *Resource Event Selector*. Em contraste, no sistema IBM PSSP/RSCT a recepção de eventos associada a predicados só é possível através do *Event Manager*, seu serviço global e distribuído de eventos de recursos. No ambiente PSSP/RSCT, uma aplicação interessada em eventos de um recurso local pode, a fim de minimizar o caminho de acesso a valores do recurso e a seu critério, acessar ela própria o monitor de recurso e monitorar suas variáveis e aplicando seus próprios predicados, realizando isso através de amostragem temporal explícita. No sistema DSMCA isso é também possível, pois uma aplicação pode acessar diretamente as interfaces dos objetos de gerenciamento de um recurso no gerenciador do mesmo. Ela pode localizar o gerenciador utilizando o serviço de localização de recursos ou acessando as interfaces dos objetos *proxy* no mesmo. De qualquer maneira, o grande diferencial reside no fato de que, para a maioria dos recursos, a aplicação pode receber eventos diretamente do recurso, pois ela pode conectar o encaminhador de eventos local do recurso a si, ou pode recebê-los indiretamente através do *Resource Event Selector*. Ainda, uma aplicação pode solicitar diversos objetos de geração de evento ao recurso e ajustá-los de modos diversos, fazendo com que o gerenciador envie eventos gerados por sintaxes diversas.

5.5.2.3 Serviços de Ação

Em geral, sistemas de gerenciamento de *cluster* utilizam a abordagem dos *user exits* para implementar suas ações gerenciais e desse modo controlar e configurar recursos quando necessário. Nos sistemas SGI FailSafe e IBM HACMP, essa é a estratégia comum. No sistema SGI FailSafe, a gerência de recursos (SRM) cria processos externos para ativar *plug-ins* que têm a finalidade de controlar, configurar e monitorar os recursos. O

SRM utiliza seus *plug-ins* de monitoração para obter eventos dos recursos. No sistema da IBM, o HACMP cria processos externos a fim de executar diversas categorias de *scripts* que visam controlar seus recursos. HACMP utiliza-se dos serviços de evento (EM) para que, de um modo indireto, tenha acesso aos monitores de recurso (RM). De um modo geral, esses *plug-ins* ou *scripts* são classificados como agentes de *ações* nos recursos. Observa-se, portanto, uma separação entre as estratégias de evento e de ação, havendo elementos distintos ligados a cada um deles. O sistema DSMCA faz essa separação, porém eventos e ações são realizados em um único elemento: o gerenciador de recurso, o qual tem as capacidades de tanto monitorar quanto de controlar e configurar recursos. Por outro lado, eventos e ações são tratados por áreas distintas no sistema, havendo áreas específicas para ações, eventos e controle de recursos. No SGI FailSafe, essas três áreas são partes internas do SRM, e desse modo não há interfaces separadas visíveis para elas. No ambiente de *cluster* IBM, pode-se observar a área de eventos como um elemento distinto (o *Event Manager*), o HACMP como elemento gerenciador de recursos e serviços e, como elementos da área de ações, seus diversos *scripts* externos.

O sistema DSMCA estrutura essas três áreas - eventos, ações e gerenciamento de recursos - de maneira separada, porém integrada. Em relação a ações, DSMCA possui áreas distintas tanto para a seleção de ações baseada em eventos (ou correlação evento-ação) quanto para a execução de ações pré-estabelecidas (ou execução de ações), para coleta e distribuição seletiva de eventos, local (*Event Selector*) ou remota (*Remote Monitoring*), e uma gerência de recursos que atua como configuradora, controladora, intermediária e encaminhadora de ações aos recursos. As fronteiras e interfaces de todas essas áreas são visíveis e documentadas, havendo portanto a possibilidade de acesso e também de extensão das funcionalidades do sistema por novos módulos (vide Figura 4-13). Ações são seqüências de chamadas a métodos ou propriedades específicas do

recurso visando alterar seu estado a fim de configura-lo, recupera-lo, corrigi-lo ou outras finalidades. A utilização de *scripts* externos com tal finalidade com certeza é uma estratégia que permite a fácil alteração das ações por parte da gerência do *cluster* e a adição ou modificação da lógica e comportamento das ações. Por outro lado, uma estratégia de ações tal como a implementada por DSCMA torna rígida as regras e as possibilidades de uma ação, uma vez que esta é baseada apenas nas características, interfaces e propriedades documentadas no sistema sobre o recurso. Em compensação, pode minimizar erros e ações equivocadas, aquilo que os *scripts* externos permitem com facilidade. Porém, não podemos ignorar a flexibilidade introduzida pelo conceito de *user exit* dos *scripts*, e assim não se descarta a possibilidade de que ações sejam implementadas como uma intercalação de seqüências rígidas e de chamadas de *scripts* externos a fim de dar maior flexibilidade ao processo de elaboração da ação. Para tanto, o gerenciador de recurso pode ativar um *script* externo sempre que determinada seqüência da ação está em andamento. Uma das razões que justificam tal estratégia é o fato de que o executor de ações, pelo menos até o presente momento, foi projetado para executar incondicionalmente uma seqüência. Os *user exit scripts* trariam, portanto, a possibilidade de haver execução condicional de cada seqüência da ação. Ou também, num aprimoramento maior, dotar as definições de ações e seqüências e o executor de ações de DSMCA de semânticas condicionais a fim de minimizar a necessidade de agentes externos na execução de ações.

5.5.2.4 Gerenciamento de Recursos e Serviços

Nos sistemas comerciais apresentados, o gerenciamento de recursos e serviços compreendem serviços locais e globais ao *cluster*. No sistema SGI FailSafe, a gerência de recursos (SRM) em conjunto com o módulo de políticas FailSafe controlam os serviços do *cluster*. Nesse sistema,

existem os conceitos de *recurso* e de *grupo de recursos*, elementos controlados pelo SRM de acordo com as políticas de recursos definidas e reforçadas pelo módulo FailSafe. O SRM cuida para que recursos de um grupo sejam ativados em uma ordem correta e pré-estabelecida, configura-os, controla-os e monitora-os através de seus *plug-ins*. A ativação, migração, *takeover* e *failover* de grupos de recursos é controlada pelo módulo FailSafe, que como o SRM é um serviço distribuído. No IBM HACMP operando com o suporte de PSSP, o HACMP cumpre os papéis políticos em relação a grupos de recursos, tais como migração, *takeover* e *failover* e também é ele que cuida das ações de ativação em ordem correta e pré-estabelecida de grupos de recursos. Vemos portanto que nesses dois sistemas essas áreas e funções gerenciais ficam agregadas com modularidade distinta.

No sistema DSMCA, a distinção entre as áreas e serviços gerenciais que cuidam dos aspectos de recursos e grupos de recursos é bem definida e distinta. DSMCA divide o problema de gerenciamento de recursos em três áreas distintas: gerenciamento local de recursos, gerenciamento de serviços e gerenciamento de clusters. A primeira área de gerenciamento tem suas funções realizadas pela gerência de recursos (*Resource Management*), uma área gerencial local que cuida de recursos apenas no âmbito de um nó, somente quando o recurso está ativo naquele nó. A segunda área de gerenciamento tem suas funções realizadas pela gerenciador de serviços (*Service Manager*). DSMCA define o conceito de *serviço* a fim de desvincular a interface dessa área do conceito de recurso, ou seja, essa área é direcionada e tem suas primitivas baseadas em serviços e não em recursos. Os clientes da gerência de serviços solicitam ações sobre serviços, os quais possuem identificadores. Além disso, serviços podem ter políticas locais associadas aos mesmos, que servem para que a gerência de serviços saiba como lidar com problemas no serviço localmente. Logicamente, o gerenciador de serviços busca no

serviço de dados global as informações pertinentes aos serviços tais como o grupo de recursos que utiliza e as suas relações de ordem e seqüência a fim de poder solicitar corretamente ações sobre os recursos à gerência de recursos. Finalmente, a terceira área gerencial, a gerência de cluster (*Cluster Manager*) gerencia *clusters* e serviços de *cluster* em nível global. É o módulo de serviço que reforça as políticas de *cluster* e dos serviços de um *cluster*, determinando o nó ou nós mais apropriados ou indicados para a execução de um determinado serviço, determinando qual a ação de migração ou rotação de serviço mais indicada e tratando exceções de falha de serviço ou de nós de cluster, promovendo o *failover* de um serviço para um outro nó ou outro conjunto de nós. Novamente temos um serviço que visa desvincular serviços da localidade do serviço, ou seja, do local onde o mesmo está executando ou pode executar. Muitas das chamadas ao serviço de *Cluster Manager* são desvinculadas de informações de localidade, podendo-se solicitar a ativação de um certo serviço sem haver necessidade de se especificar o nó ou nós. Nestes casos, o serviço utiliza-se de suas políticas para iniciar votação e consenso sobre o melhor local para a execução do serviço e após essa etapa o serviço é iniciado no local escolhido pelos nós componentes do domínio do serviço. Outras chamadas do serviço, tais como *rotate()* (para que um serviço rode para outro nó ou nós do seu domínio) têm semânticas similares. Nesse sentido, o *Cluster Manager* aproxima-se em suas funções do módulo FailSafe e das funcionalidades do IBM HACMP. Porém, um diferencial interessante de DSMCA nesse aspecto é o fato de que é possível que aplicações externas ao sistema possam participar das políticas diversas envolvidas, sejam as políticas de serviços do *cluster* até as políticas locais de serviços e recursos. No caso dos serviços globais de *cluster*, é possível que uma aplicação junte-se ao grupo de decisões do *Cluster Manager* (que utiliza o serviço de grupos para tal) e, como membro do grupo, participe de decisões conjuntas sobre serviços. Neste caso, tem-se um elemento ativo no processo decisório e político, implementado segundo as necessidades e

requerimentos específicos daquela instalação do *cluster*, reforçando políticas específicas para os serviços. No caso do gerenciamento local de recursos, uma aplicação pode solicitar ser encaminhadora de eventos e ações do recurso e assim participar ativamente de decisões sobre ações nos recursos. Essa última possibilidade provavelmente é mais rara em termos de utilização, porém possível.

Capítulo VI

CONCLUSÕES E TRABALHOS FUTUROS

6 Conclusões e Trabalhos Futuros

De acordo com a proposta inicial, o sistema DSMCA atende a um conjunto de requisitos básicos referentes ao gerenciamento de *clusters* e das aplicações convencionais e distribuídas que executam nos mesmos. Dentre esses requisitos, temos aqueles relativos à capacidade do sistema de lidar com condições de falha de elementos internos ao *cluster* tais como nós processadores, programas, aplicações e elementos de rede. A arquitetura de DSMCA provê, como visto nos capítulos 4 e 5, um conjunto de programas, gerências e serviços capazes de lidar com essas possíveis falhas. Os gerenciadores de recurso de DSMCA são elementos altamente especializados em cada recurso que gerenciam, podendo emitir mensagens de evento relevantes sobre os recursos, tais como mudanças de estado e ocorrência de falhas, além de poder receber chamadas para realizar ações gerenciais corretivas e de ajuste nos mesmos a fim de recuperar possíveis problemas ou evitar que eles ocorram. A arquitetura

provê também serviços capazes de informar eventos a grupos de interessados, executar ações gerenciais, oferecer mecanismos de coordenação, suporte a grupos de processos, sincronização, consenso e votação, além de prover mecanismos de comunicação confiável. Como pode ser constatado nos capítulos anteriores, DSMCA provê uma série de serviços visando atender esses requisitos, tais como os serviços de eventos e de ações, os serviços de comunicação em grupo e o serviço de mensagens. Esses serviços básicos funcionam como suporte para que as gerências de recursos, de serviços e de *cluster* possam atender os requisitos de confiabilidade e disponibilidade das aplicações do *cluster*, dotando o *cluster* da capacidade de detectar falhas em diversos níveis e de diversos tipos, nos recursos e serviços, apresentando a capacidade de migrar serviços para outros nós do *cluster* em diversas situações distintas, não apenas em condições de falha, mas também por razões de distribuição de carga de trabalho ou por solicitação da aplicação de gerenciamento de *clusters*. A divisão modular da arquitetura proposta é resultado de análises realizadas nos requisitos necessários e também nas funcionalidades e módulos existentes em outras arquiteturas, tais como aquelas apresentadas no capítulo 5. Procurou-se expor as áreas de serviço e gerências em um nível de granularidade fina o suficiente para que os serviços interessantes tivessem suas interfaces expostas, pois que se considerou isso um aspecto importante. Também, manteve-se a granularidade a um nível tal que certas tarefas ficassem integradas em um único módulo a fim de facilitar a compreensão de sua lógica e a sua implementação.

Um fato a respeito da OMG CORBA é que ela não é uma arquitetura de interconexão de objetos voltada a desempenho elevado de comunicação. O seu principal objetivo é prover fácil interoperabilidade entre objetos em ambientes heterogêneos. Os protocolos IOP, tais como o GIOP e os seus pares IIOP e UDP/IOP não são protocolos simples, tampouco são simples as implementações de CORBA. Entretanto, é notável o desempenho e o

overhead relativamente baixo de algumas implementações desse ORB. Assim, não se espera que DSMCA seja utilizado como plataforma para o desenvolvimento de aplicações HPC. Para esses casos, aconselha-se o desenvolvimento com a utilização de bibliotecas de passagem de mensagem tais como MPI e PVM. O DSMCA deve ser visto como uma plataforma para o desenvolvimento de aplicações distribuídas, paralelas, mas que não tem necessidades de comunicação intensa e de baixa latência entre suas instâncias. Vale racapitular aqui o suporte à comunicação oferecido por DSMCA pelos seus serviços de mensagem e de grupos. São formas de comunicação robustas e dotadas de semânticas especiais, porém tem grande *overhead* e podem apresentar, sob certas circunstâncias, latências altas. Porém o objetivo desses serviços é oferecer comunicação baseada em semânticas de transação e confiáveis sob circunstâncias extremas e contingenciais, nas quais a comunicação deve ser conseguida a qualquer custo. Uma das razões disto é que os serviços devem manter, na medida do possível, consistência e coordenação. Sendo assim, a adoção de CORBA não invalida a arquitetura, pois atende seus requisitos.

Um outro aspecto interessante da adoção de CORBA tem a ver com a interoperabilidade. O desenvolvimento de aplicações que se beneficiam dos serviços DSMCA, que nele se integram, ou que o gerenciam, pode ser facilitado pelo fato de que as interfaces para qualquer serviço são baseadas em IDL. Havendo um ambiente de desenvolvimento CORBA para uma dada plataforma de *hardware* e *software* específico, não é difícil desenvolver uma aplicação para DSMCA através do mesmo. Pode-se, por exemplo, desenvolver uma aplicação de gerenciamento do *cluster* em um ambiente *Microsoft Windows* utilizando-se de um ambiente que disponibilize um ORB CORBA e um compilador IDL para aquela plataforma. O *cluster* em questão pode ser composto, por exemplo, de máquinas rodando *Linux* e DSMCA. A interoperabilidade entre essa

aplicação de gerenciamento e o *cluster* DSMCA será extremamente facilitada, normalmente conseguida de modo totalmente transparente.

Como propostas para a continuidade deste trabalho, temos algumas considerações a fazer. Em primeiro lugar, tem-se a noção clara de que a implementação de certos serviços de DSMCA representa um desafio, particularmente devido à adoção de CORBA como plataforma de desenvolvimento. Alguns dos serviços de DSMCA tem clara e intrinsecamente um comportamento assíncrono, porém são totalmente *statefull*. É o caso do serviço de grupos, no qual é possível constatar que toda e qualquer interação de um membro com o grupo deve ser registrada e identificada a fim de que o processo ocorra como uma transação. O serviço deve registrar essa transação e iniciar o envio de recebimento de mensagens relativas a ela entre os membros, assincronamente, até que esta se complete. Um claro desafio para esse serviço é o fato de que CORBA pressupõe a ativação de *threads* para o atendimento das chamadas a seus objetos e desse modo haverão várias seqüências de chamadas correlatas entre objetos de um servidor ocorrendo em *threads* distintas, a fim de que a transação chegue a um final. Aqui, o desafio é a sincronização dessas diversas linhas de execução, o que provavelmente implicará no uso de semáforos *mutex* e também monitores ou variáveis de condição. Felizmente, esses elementos de controle de execução são facilmente utilizáveis, particularmente em ambientes de programação *Unix* e *Linux*. Um outro serviço cuja implementação CORBA deve lançar mão de artifícios similares é o serviço de monitoria de adaptadores. Este, além de necessitar de comunicação externa através de protocolos *stateless*, deverá ser regido internamente por elementos de manutenção de estado. Já tecemos comentários em (4.1.3) a respeito do protocolo de manutenção do *membership* utilizado, de modo que a implementação deste por objetos CORBA representa um desafio. Um problema real é a constatação de que as implementações de CORBA podem ignorar a diretiva *oneway* dos

métodos de interface IDL. Felizmente, este não parece ser o caso de Orbacus [ORBacus], porém deve-se pensar seriamente nas questões de portabilidade do código. Um outro desafio que se apresenta para esse serviço é a necessidade de utilização do tempo como evento de mudança na máquina de estado do protocolo. A combinação de um ambiente de servidor CORBA, onde múltiplas *threads* estão em execução concorrente devido às chamadas aos objetos, com a presença de múltiplos temporizadores (de *timeout*, etc.) ativos, que podem gerar alarmes assincronamente, representa um desafio considerável. Algumas implementações de CORBA provêm automaticamente para suas aplicações temporizadores associados a ações explícitas, tais como os temporizadores destinados a erguer exceções por *timeout* durante invocações a métodos remotos de objeto e durante a conexão a servidores remotos. Porém, eventos gerados por alarmes assíncronos definidos pelo programador representam uma situação completamente diversa e desafiadora no ambiente *multithreaded* de uma aplicação CORBA. Situação parecida deve envolver a implementação de muitos dos objetos geradores de eventos de recurso de DSMCA, em particular dos objetos presentes nos gerenciadores de recurso que devem realizar amostragem temporal de valores dos recursos gerenciados (os objetos da classe *sampler*) a fim de compará-los com predicados ou condições de geração de eventos. Esses objetos, que tem suas interfaces definidas no módulo *Event::*, são implementações de objetos CORBA executando em *threads* de servidor, as quais podem ser selecionadas para execução assincronamente devido a sinais de alarme dos temporizadores utilizados para determinar os instantes de amostragem. Constituem, portanto, um desafio de implementação, pois em princípio um servidor de objetos CORBA seleciona uma *thread* para execução de um método de objeto somente quando uma solicitação via protocolo IOP é recebida e assim é preciso cuidado para evitar situações de conflito. Tais situações provavelmente envolverão o uso de *threads* separadas para os diversos

tipos de eventos que a aplicação deverá tratar, o que obrigará o uso de semáforos *mutex* para coordenar o acesso a dados compartilhados por regiões críticas de seu código.

Um outro tipo de desafio interessante na implementação de DSMCA é que alguns de seus serviços essenciais na verdade apóiam-se mutuamente. É o caso, por exemplo, do serviço gerenciador de (serviços) de *cluster*, do serviço gerenciador de dados e do serviço de monitoria remota. Como se pode constatar, são estes os casos onde serviços DSMCA constituem aplicações distribuídas, apoiadas em serviços básicos do próprio DSMCA. Esses serviços apóiam-se completamente no serviço de grupos para poder funcionar. Na implementação dos mesmos, será necessária a definição de um conjunto de mensagens protocolares específico para cada um, mensagens essas que serão trocadas entre as instâncias da aplicação através do serviço de grupo, possuindo sintaxe e semântica específica do protocolo interno do serviço e interpretável apenas pelos objetos parceiros do serviço. Como exemplo, o serviço de dados de configuração do *cluster* deve possuir um conjunto de mensagens que será usado para que as instâncias distribuídas do servidor de dados possam manter sincronizadas suas versões do banco de dados através de atualizações atômicas de registros do banco de dados distribuído, entre outras. Na terminologia do serviço de grupos, essas mensagens são chamadas de *propostas* – mensagens que visam estabelecer acordos e consensos por votação (aceite/rejeição) entre as partes da aplicação. Para interações entre instâncias do serviço que não necessitam de acordos ou consensos, o serviço oferece primitivas para o envio de mensagens ordinárias do grupo, com semânticas de atomicidade ou não.

Como proposta de continuidade deste trabalho, o qual constituiu-se na descrição arquitetural do sistema DSMCA, podemos estabelecer uma seqüência de novos trabalhos visando iniciar o seu desenvolvimento e

implementação de todo o sistema. Essa seqüência obedece a uma ordem lógica e não está necessariamente direcionada pelo grau de dificuldade de cada etapa. Segue abaixo a seqüência sugerida:

1. Inicialmente, uma modelagem de dados e de um banco de dados de configuração do sistema DSMCA e dos *clusters* por ele gerenciados.
2. A definição de conjuntos completos de mensagens para os grupos implementando os diversos serviços essenciais, a saber: o serviço de comunicação em grupo, o serviço de monitoria de adaptadores, de monitoria de nós, o serviço de mensagens confiáveis e o serviço de banco dados de configuração.
3. A modelagem dos objetos de serviços básicos, incluindo aqui os modelos de interações, estados e comportamentos, seguida da implementação desses serviços essenciais. Esta etapa, que provavelmente será uma das mais desafiadoras, envolverá elevado grau de *expertise* do programador, pois envolve complexidade e habilidades com ambientes *multithreaded*, sinais, eventos, sincronização, protocolos de comunicação e sistemas gerenciadores de banco de dados.
4. A modelagem e implementação de alguns gerenciadores de recursos básicos e também do serviço de ativação e localização de recursos e gerenciadores de recurso (*hub*). São pré-requisitos para os itens seguintes da seqüência. Compõe também uma etapa envolvendo elevada complexidade e desafio ao programador, conforme discussão já feita.
5. A modelagem e implementação dos objetos de serviços relativos as áreas de eventos de recurso e ações de gerenciamento.

Teríamos aqui: os serviços de seleção de eventos, de seleção de ações e de execução de ações. A conclusão desta etapa permite a construção da área de gerenciamento de recursos, pois esta é integrada aos serviços aqui implementados.

6. A modelagem e implementação dos objetos das áreas de serviços de gerenciamento locais específicos. Teríamos aqui: os serviços de gerenciamento de serviços locais e de recursos locais. A gerência de recursos locais opera em conjunto com os serviços de eventos e ações, como encaminhador de ações.
7. A modelagem e implementação dos objetos das áreas de serviços de gerenciamento globais e específicos. Teríamos aqui: os serviços de gerenciamento de (serviços de) cluster e o serviço de monitoria remota de recursos e eventos. A gerência de serviços de cluster permite a construção de aplicações de gerenciamento do DSMCA e de seus *clusters*.
8. Desenvolvimento de aplicações de gerenciamento de *cluster* DSMCA baseada em interface gráfica do usuário. Desenvolvimento de gerenciadores de recursos novos ou específicos.

Por fim, acreditamos que este trabalho vem contribuir para que a arquitetura e os conceitos teóricos envolvendo sistemas em *cluster* com características de alta disponibilidade, tanto aqueles da classe *Beowulf* quanto sistemas comerciais ou proprietários, possam ser melhor compreendidos, desenvolvidos e utilizados. Esperamos que este texto traga subsídios para desenvolvimentos futuros de outros sistemas e de novas aplicações de *clusters*. Acreditamos que o sistema DSMCA apresentado neste trabalho possa servir como um modelo arquitetural para

estudos futuros e base para desenvolvimento de aplicações de *cluster* e de sistemas de gerenciamento de *cluster*.

REFERÊNCIAS BIBLIOGRÁFICAS

- [ANSACOM95] “A comparison of Object Models”; Watson, Andrew; ANSA Architecture for Open Distributed Systems, Citrix Systems (Research & Development) Ltd; <http://www.ansa.co.uk/ANSATech/95/Bground/15300001.pdf>, Junho/2002
- [ATTANASIO94] “Design and implementation of a recoverable virtual shared disk”; C. Attanasio; M. Butrico; C. Polyzois ; S. Smith ; J. Peterson; IBM T. J. Watson Research Center, Yorktown Heights; 1994; <http://citeseer.nj.nec.com/attanasio94design.html>, Junho/2002
- [BAKER1999] “Cluster Computing: The Commodity Supercomputing”; Baker, Mark; Buyya, Rajkumar; Journal of Software - Practice & Experience; John Wiley & Sons, Inc; vol. 1; 1999; <http://citeseer.nj.nec.com/baker88cluster.html>; Junho/2002
- [BAKER2000] “Cluster Computing White Paper”; Baker, Mark; editor; IEEE Computer Society Task Force on Cluster Computing; <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper>; Junho/2002
- [BEYER1995] “High Availability on the RISC System/6000 Family”; Beyer, Andrew et. al.; IBM ITSO; ISBN 0738407585; IBM Form Number SG244551-00, 1995; <http://www.ibm.com/redbooks> , Junho/2002
- [BUYYA1999] “Cluster Computing at a Glance”; Buyya, Rajkumar; University of Portsmouth; Baker, Mark; Monash University; in “High Performance Cluster Computing”; chap. 1; Prentice-Hall; 1999; <http://www.phptr.com> , Junho/2002

- [CARNS2000] "PVFS: A Parallel File System For Linux Clusters"; Carns, P.H.; Ligon, W. B. III; Ross, R. B., Thakur, R.; Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000, pp. 317-327 <http://parlweb.parl.clemson.edu/pvfs/papers.html>, Junho/2002
- [CHIAKPO1996] "RS/6000 SP High Availability Infrastructure"; Chiakpo, Endy et al.; IBM ITSO; ISBN 0738404098; IBM Form Number SG24-4838-00, 1996; <http://www.ibm.com/redbooks>, Junho/2002
- [CHIAKPO1997] "HACMP Enhanced Scalability"; Chiakpo, Endy et al.; IBM ITSO; ISBN 0738404098; IBM Form Number SG24-2081-00, 1995; <http://www.ibm.com/redbooks>, Junho/2002
- [CISCOETH] "Internetworking Technology Handbook" > "Ethernet"; Cisco Documentation; Cisco Systems, Inc.; © 1992--2002; http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ethernet.htm; Junho/2002
- [CISCOFDDI] "Internetworking Technology Handbook" > "Fiber Distributed Data Interface"; Cisco Documentation; Cisco Systems, Inc.; © 1992--2002; http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/fddi.htm; Junho/2002
- [CISCOLANSW] "Internetworking Technology Handbook" > "Lan Switching"; Cisco Documentation; Cisco Systems, Inc.; © 1992--2002; http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/lanswtch.htm; Junho/2002
- [CORBA] "The Common Object Request Broker Architecture"; Object Management Group, Inc.; Document [formal/02-05-15](http://www.omg.org/technology/documents/formal/corba_iiop.htm); http://www.omg.org/technology/documents/formal/corba_iiop.htm, Junho/2002

- [DIAS96] "A Scalable and Highly Available Web Server"; Dias, Daniel M et al.; IBM Research; COMPCON-96, pages 85-92; "High Performance Cluster Computing", chap. IV; Buya, Rajkumar; editor; 1999; ISBN 013013784; <http://citeseer.nj.nec.com/dias96scalable.html> , Junho/2002
- [DOORS] "DOORS: Towards High-Performance Fault Tolerant CORBA"; Natarajan, Balachandran; Schmidt, Douglas C.; International Symposium on Distributed Objects and Applications; 2000 <http://www.cs.wustl.edu/~schmidt/corba.html>, Junho/2002
- [EBER1996] "Implementing High Availability on RISC/6000 SP"; Eberhardt, Giller et al.; IBM ITSO; IBM Form Number SG24-4742-00, 1996; <http://www.ibm.com/redbooks>, Junho/2002
- [FAILSAFE] "Linux FailsafeTM"; SGI Open Source Project.; Silicon Graphics Corporation; 2000 <http://oss.sgi.com/projects/failsafe/>, Junho/2002
- [FAKE] "Fake"; Horman, Simon; Vergenet Test Network; 2000 <http://www.vergenet.net/linux/fake/>, Junho/2002
- [FERREIRA2002] "Linux Clustering with CSM and GPFS"; Ferreira, Luis et al.; IBM Redbooks; ISBN 0738423963; Form Number SG246601-00, 2002; <http://www.ibm.com/redbooks>, Junho/2002
- [FT-CORBA] "Fault Tolerant Corba Specification"; The Object Management Group; Document -- formal/01-12-63 (CORBA 2.6 - chapter 25 - Fault Tolerant CORBA) ; <http://www.omg.org/cgi-bin/doc?formal/01-12-63>, Junho/2002
- [GPFS] "A GPFS Primer"; IBM Unix Servers White papers; International Business Machines Corp; http://www-1.ibm.com/servers/eserver/pseries/software/whitepapers/gpfs_primer.html, Junho/2002
- [HAL] "The High Availability Linux Project"; <http://www.linux-ha.org>, Junho/2002

- [HAVENST1999] “Object Groups in Transactional CORBA Systems”; Stefan Havenstein; Fachgebiet Computergestützte Informationssysteme; Technische Universität Berlin; 1999
- [HEGER2001] “IBM’s General Parallel File System (GPFS) 1.4 For AIX”; Heger, Dominique; Shah, Gautam; IBM Redpapers; November 2001; <http://www.redbooks.ibm.com/redpapers/pdfs/redp0442.pdf>; Junho/2002
- [IBMCLUSTER] “IBMTMe(logo)server Clusters”; <http://www.ibm.com/servers/eserver/clusters/>, Junho/2002
- [IBMSP2] “RS/6000 SP SystemTM”; Overview of Platform; <http://www.npac.syr.edu/nse/hpccsurvey/orgs/ibm/ibm.html#RS/6000>, Junho/2002
- [IMEXSAN] “SAN/NAS IMEX Industry Report”; IMEX Research; <http://www.imexresearch.com/reports/san.shtml>, Junho/2002
- [KEIDAR2000] “Moshe: A Group Membership Service for WANs”; Idit Keidar et. al; Massachusetts Institute of Technology; Technical Memorandum; 2000
- [LVS] “The Linux Virtual Server Project”; <http://www.linuxvirtualserver.org> , Junho/2002
- [MICO] “MICO: An Open Source CORBA Implementation”; Puder, Arno; Roemer, Kay; Morgan-Kauffman Publishers, Inc.; <http://www.mico.org/>, Junho/2002
- [MIRCHAN93] “FDDI: Technology and Applications”; Mirchandani, Sonu; Khanna, Raman - editors; Wiley Professional Computing; John Wiley e Sons; 1993; ISBN 0471558966;
- [MON] “MON – Service Monitoring Daemon”; Trocki, Jim; Kernel.Org Organization, Inc.,; <http://www.kernel.org/software/mon/> , Junho/2002

- [MPI] “The Message Passing Interface Standard”; Argonne National Laboratory; Mathematics and Computer Science Division; <http://www.mcs.anl.gov/mpi>, Junho/2002
- [MSCS] “The Design and Architecture of the Microsoft Cluster Service”; Vogels, Werner et al.; Dept. of Computer Science, Cornell University; Gamache, Rod et al.; Microsoft Corporation; 1998; www.cs.cornell.edu/rdc/mscs/ftcs28/default.html, Junho/2002
- [MYRINET] “Myrinet Overview”; Myricom, Inc. 325 N. Santa Anita Ave. Arcadia, CA; <http://www.myri.com/myrinet/overview/index.html>, Junho/2002
- [NAM2000] “Group Communication Support for CORBA using OCI”; Nam, Dukyun et. al; School of Engineering, Information and Communications University of Taejon, Korea; Youn, Hee Young; School of Electrical and Computer Engineering, Sungkyunkwan University; Suwon, Korea; 2000
- [OMG] “The Object Management Group”; Object Management Group, Inc.; <http://www.omg.org>, Junho/2002
- [OMNIORB] “OmniORB – CORBA ORB 2.3 Implementation”; Lo, Sai-Lai; Riddoch, David; Grisby, Duncan; AT&T Laboratories Cambridge; <http://www.uk.research.att.com/omniORB>, Junho/2002
- [ORBacus] “ORBacus – For C++ and Java. Version 4.1.0”; IONA Technologias, Inc.; http://www.iona.com/products/orbacus_home.htm, Junho/2002
- [ORBEvent] “Corba Event Service Specification”; The Object Management Group; Document [formal/2001-03-01](http://www.omg.org/technology/documents/formal/event_service.htm) http://www.omg.org/technology/documents/formal/event_service.htm, Junho/2002

- [ORBNaming] “Corba Naming Service Specification”; The Object Management Group; Document formal/2001-02-65 ; http://www.omg.org/technology/documents/formal/naming_service.htm, Junho/2002
- [ORBNotify] “Corba Notification Service Specification”; The Object Management Group; Document formal/2000-06-20 ; http://www.omg.org/technology/documents/formal/notification_service.htm, Junho/2002
- [ORBProperties] “Corba Properties Service Specification”; The Object Management Group; Document formal/2000-06-22; http://www.omg.org/technology/documents/formal/property_service.htm, Junho/2002
- [PFISTER1998] “In Search of Clusters”; Pfister, Gregory; 2nd Edition; Prentice-Hall PTR; ISBN 0138997098, 1998; <http://www.prenhall.com/catalog/academic> ; Junho/2002
- [PVFS] “The Parallel Virtual File System”; The Parallel Architecture Research Lab; Clemson University; <http://parlweb.parl.clemson.edu/>, Junho/2002
- [PVM] “The Parallel Virtual Machine Project”; Oak Ridge National Laboratory; <http://www.epm.ornl.gov/pvm>, Junho/2002
- [RAIDEDU] “The Advanced Computer & Network Corporation” > “Raid.edu”; <http://www.acnc.com/index.html>, Junho/2002
- [REISNER2001] “DRBD”; “Philipp Reisner <philipp.reisner@gmx.at>”; <http://www.complang.tuwien.ac.at/reisner/drbd/>, Junho/2002
- [RODRIGUES1993] “A Low-Level Processor Group Membership Protocol for LANs”; L. Rodrigues et. al; Technical University of Lisboa; IST-UNESC; 1993

- [Satyanarayanan89] “Coda: A Highly Available File System for a Distributed Workstation Environment ”; Satyanarayanan, M. ; Proceedings of the Second IEEE Workshop on Workstation Operating Systems; Sep. 1989, Pacific Grove, CA; <http://www.coda.cs.cmu.edu/index.html>; Junho/2002
- [SCHMIDT1995] “Distributed Object Computing with CORBA”; Douglas C. Schmidt; Computer Sciences and Engineering Dept.; University of Washington; <http://www.cs.wustl.edu/~schmidt/corba.html>; Junho/2002
- [SHERMAN1998] “Choosing the Right Availability Solution”; Sherman, Larry; Stratus Computer Technical White Paper; Stratus Computer Inc; 1998; <http://www.stratus.com/docs/service/whitep/choosing.htm>; Junho/2002
- [SHORT98] “Windows NT Clusters for Availability and Scalability”; Short, Robert et al.; Microsoft Corporation; 1998; http://research.microsoft.com/~gray/mscs_FTCS98.doc; Junho/2002
- [STE1995] “Beowulf: A Parallel Workstation For Scientific Computation”; Thomas Sterling, Donald J. Becker, et al.; Proceedings of the 24th International Conference on Parallel Processing; 1995; <http://citeseer.nj.nec.com/sterling95beowulf.html>; Junho/2002
- [STE2002] “Beowulf Cluster Computing with Linux”; Sterling, Thomas (editor); Massachusetts Institute of Technology Press, Cambridge; 2002; <http://mitpress.mit.edu/catalog/item/default.asp?itype=2&tid=8681>; Junho/2002
- [SUNNFS] “NFS: Network File System Protocol Specification”; RFC1094; Network Working Group; The Internet Engineering Task Force; <http://www.ietf.org/rfc/rfc1094.txt>; Junho/2002
- [TERTIARY] “The Tertiary Disk Project”; Computer Science Division Office”; University of California, Berkeley, CA; <http://now.cs.berkeley.edu/Td/>; Junho/2002

- [VINOSKI1996] “CORBA: Integrating Diverse Applications Within Heterogeneous Environments”; Vinoski, Steve; 1996; <http://www.cs.wustl.edu/~schmidt/PDF/vinoski.pdf> , Junho/ 2002

- [VINOSKI1999] “Advanced CORBA Programming with C++”; Vinoski, Steve; Henning, Michi; Addison-Wesley; ISBN 0201379279; 1999

- [YOUN2000] “The Approaches for High Available and Fault-Tolerant *Cluster* Systems”; Youn, Young; Yu, Chansu; Han, Dong Soo; Lee, Dongman; Information and Communications University of Taejon; Korea; 2000; <http://citeseer.nj.nec.com/512350.html>; Junho/2002