Curso de Ciência da Computação
Disciplina Tópicos Especiais em Computação - Sistemas Distribuídos
Prof. Ademir Goulart

Sistemas Distribuídos e Comunicação em Grupo

UNIVALI – Universidade do Vale do Itajai

SUMÁRIO

Lista de Figuras

Lista de Quadros

1 INTRODUÇÃO

Redes de computadores estão por toda parte. A internet é uma, como são as muitas redes que a compõem. Redes de telefonia celular, redes corporativas, redes fabris, redes em Campus, redes domésticas, redes embarcadas, todas estas, tanto separadamente como em conjunto, compartilham as características essenciais que fazem delas um modelo relevante para o estudo sob o título Sistemas Distribuídos. Uma definição de sistemas distribuídos é aquela no qual os componentes de hardware e software localizados em computadores interligados por rede, comunicam e coordenam suas ações somente através da troca de mensagens (COULOURIS, 2001).

A comunicação entre os computadores interligados, pode ocorrer de diferentes formas, usando diversos protocolos tanto em ambiente de rede local como ambiente de redes de longa distância. Um caso particular de comunicação ocorre quando uma mesma mensagem tem que ser enviada para diversos computadores na rede. Alguns ambientes físic os de rede permitem o *broadcast*, onde um único comando de envio manda a mensagem para todos os endereços da rede porém a confiabilidade deste método não é garantida, podendo ocorrer alguma perda sem que o emissor da mensagem seja notificado desta perda.

Visando atender esta necessidade de comunicação de um para muitos, com confiabilidade, escalabilidade, e de forma transparente para quem desenvolve a aplicação, foi criado um novo paradigma chamado Comunicação em Grupo. Um protocolo de Comunicação em Grupo separa a complexidade de controle e gerenciamento das mensagens, da complexidade da aplicação, tratando os diversos computadores como sendo pertencentes a um grupo. Assim o projetista se preocupa apenas com a aplicação, suas funcionalidades, seus procedimentos, seus algoritmos particulares, sem se envolver com os problemas da comunicação. Passa a ser responsabilidade do Protocolo de Comunicação em Grupo a controle de fluxo, a confiabilidade, o sequenciamento e a garantia de entrega das mensagens ao aplicativo final, bem como o controle de quais computadores estão fazendo parte deste grupo, gerenciando a entrada de novos membros no grupo, saída voluntária ou saídas involuntárias devido a quebras no computador ou particionamento da rede.

Este trabalho visa apresentar uma conceituação básica sobre sistemas distribuídos e na seqüência um capítulo apresentando de forma genérica o conceito de comunicação em Grupo.

2 CONCEITUAÇÃO BÁSICA de SISTEMAS DISTRIBUÍDOS

Neste capitulo apresentamos uma série de conceitos que são importantes para o perfeito entendimento e enquadramento de nosso assunto objeto, Comunicação em Grupo. Iniciamos com uma caracterização dos sistemas distribuídos e paralelos, bem como apresentando motivações para o uso de sistemas distribuídos. Na seqüência é abordado o aspecto de hardware, apresentando arquiteturas atualmente em uso para sistemas distribuídos. Os aspectos de software como sistemas operacionais, linguagens de programação e ambientes distribuídos também são explorados neste capitulo. Para encerrar a conceituação básica apresentamos as arquiteturas adotadas na comunicação de sistemas distribuídos.

2.1 Características dos Sistemas Distribuídos e Paralelos

Os primeiros computadores produzidos para uso comercial, na década de 70 tinham uma estrutura padrão baseada em processador, memória e armazenamento de dados. Devido ao alto custo dos componentes, na época, as capacidades de memória e de discos destes precursores dos equipamentos hoje conhecidos como mainframe, eram mínimas se comparado com equipamentos atuais. Para exemplificar um modelo de equipamento IBM 360/30, tinha memória de 32Kb, 3 discos de 10 MB, leitora perfuradora de cartões, leitora de fita de papel, e 6 unidades de fitas magnéticas. Na evolução tivemos os minicomputadores, os microcomputadores o computador pessoal ou PC e a interligação dos equipamentos em rede. A interligação das máquinas em redes propiciou o surgimento do conceito de sistemas distribuídos, aproveitando o potencial das máquinas interligadas. De outro lados temos os sistemas compostos de um único processador, memória e periféricos os quais são chamados de sistemas centralizados. Segundo Tanenbaum (1992) no passado era válido a Lei de Grosch, um especialista em computação, preconizando que “o poder computacional de um processador é proporcional ao quadrado de seu preço, ou seja, pagando duas vezes mais, pode-se obter o quádruplo da performance”. Hoje em dia esta Lei de Grosch não é mais válida, pois com as novas tecnologias, em especial usando arquitetura INTEL a cada ano, pelo

mesmo preço, obtemos o dobro ou o quádruplo de performance em relação ao equipamento anterior.

Quando avaliamos o custo de micros pessoais PC encontramos ótima relação preço/performance. Se colocarmos uma maquina independente para cada usuário teremos a dificuldade de compartilhar os dados. Alem da comunicação necessária entre os usuários é desejável o compartilhamento de recursos tais como impressoras, gravador de cd, etc. Este compartilhamento é possível se as máquinas estiverem interligadas em rede. Assim um sistema distribuído pode ser implementado com computadores pessoais ligados em rede a servidores os quais executariam os processos mais pesados de forma distribuída.

Apesar de todas as vantagens dos sistemas distribuídos, encontramos algumas dificuldades neste ambiente. O software para sistemas distribuídos é mais complexo. Assim sistemas operacionais, linguagens e mesmo aplicativos para este ambiente são relativamente novos, em fase de pesquisa, e não estão plenamente consolidados. Também a rede para interligar, seja rede local (LAN- Local Area Network) ou rede de longa distância (WAN- Wide Area Network) não tem a confiabilidade total para a interligação, podendo ocorrer perda por interrupção do link ou saturação devido à sobrecarga na banda disponível. Cabe então as soluções de software, tratar estas interrupções com protocolos e rotinas adequadas a este contexto. Um resumo das características de sistemas distribuídos, segundo Tanenbaum (1992) é apresentado na Figura 1.

Item	Descrição
Relação Custo Beneficio	Os microprocessadores oferecem uma melhor relação preço/performance do que a oferecida pelos mainframes
Desempenho	Um sistema distribuído pode ter um poder de processamento maior que o de qualquer mainframe
Confiabilidade	Se uma máquina sair do ar, o sistema como um todo pode sobreviver.
Crescimento Incremental	O poder computacional pode crescer em doses homeopáticas
Compartilhamento	Permite que mais de um usuário acesse uma base de dados comum, ou periféricos muito caros.

Comunicação	Torna muito mais simples a comunicação pessoa a pessoa, por exemplo, empregando o correio eletrônico
Flexibilidade	Espalha a carga de trabalho por todas as máquinas disponíveis ao longo da rede
Restrições	Até o presente momento não há muita disponibilidade de software para os sistemas distribuídos. Podemos ter problemas de muito trafego na rede ou problemas com segurança dos dados.

Figura 1 Características de Sistemas Distribuídos

2.2 Estratégias para Sistemas Distribuídos

As estratégias adotadas para a implementação de sistemas distribuídos dependem de diversos fatores. Temos que levar em conta o objetivo da aplicação, as características e restrições do ambiente, o custo do projeto, etc. As estratégias mais importantes, segundo Veríssimo (2001) são: distribuição para informação e compartilhamento de recursos; distribuição para maior disponibilidade e performance; distribuição para modularidade; distribuição para a descentralização; distribuição para segurança.

2.2.1 Considerações Básicas

Alguns compromissos precisam ser atendidos, dependendo das particularidades das aplicações:

- **Controle centralizado versus descentralizado** – Controle centralizado é mais fácil de implementar e gerenciar. Contudo alguns problemas de descentralização e natureza distribuída são melhores resolvidos com aplicações descentralizadas. Também podem ser consideradas arquiteturas intermediarias com transparência distribuída mas de controle centralizado.
- **Distribuição seqüencial versus concorrente** - O uso do processamento concorrente, onde temos diversos procedimentos executando ao mesmo tempo devido a requisições simultâneas, comparativamente ao

processamento seqüencial no caso de chamadas que podem ser atendidas uma a uma.

- **Distribuição visível versus invisível** – Podemos dizer que a distribuição é invisível ou transparente quando usamos modelos que escondem a distribuição. Assim em modelos usando RPC (Remote Procedure Call), CORBA (Common Object Request Broker Architecture) ou DCOM (Distributed Component Object Model) a distribuição se torna transparente. Distribuição visível é quando a aplicação se vale de mecanismos de passagem de mensagens, tais como orientados a grupos e modelos usando barramento de mensagens.
- **Envio de dados versus código** – O que distribuímos, dados ou código? Normalmente distribuímos dados para serem processados e equipamentos diferentes porém é possível também enviar código para processar os dados localmente, como no caso de *applets*.
- **Servidor versus serviço** – Devemos considerar que um serviço pode estar em mais do que um servidor e que em um servidor podemos estar executando diversos serviços.
- **Escala versus performance** – A escalabilidade normalmente é inversamente proporcional à performance. Uma boa arquitetura que permita escalabilidade, ou seja possa crescer em numero de componentes, não deve ter a sua performance linearmente diminuída, devido a este crescimento.
- **Sincronismo versus assincronismo** – Assíncrono são sistemas mais simples, mas independentes de tempo enquanto síncronos são sistemas mais complexos e tem a habilidade de assegurar especificações de controle de tempo.

Muitos destes compromissos podem ser avaliadas em conjunto e assim podemos classificar a distribuição como segue:

- **Distribuição em baixa escala** – Em ambientes mais homogêneos usando LANs ou redes de alta velocidade, onde um comportamento controlado pode ser obtido para aplicações que requerem segurança e sincronismo

- **Distribuição em larga escala** – Em ambientes de sistema distribuídos, em ambientes abertos, tipicamente de natureza heterogênea, sobre WANs interconectando LANs onde o comportamento é incerto e aplicações podem existir de forma não síncrona.

2.2.2 Distribuição para Informação e Compartilhamento de Recursos.

Esta é a forma como nasceram sistemas distribuídos e ainda é o objetivo estratégico da maior parte dos sistemas distribuídos. A tecnologia que diz respeito a esta estratégia de implementação, está relacionada com servidores centrais, protocolos de sessão remota, computação cliente-servidor, http, cliente leve e computação em rede. Também está relacionada com as clássicas formas de disseminação da informação como news, bbs e e-mail. Os esforços estão concentrados no lado do usuário, para garantir o acesso as informações e recursos que residem em servidores centrais. São importantes neste contexto o gerenciamento de usuários, segurança no estabelecimento da sessão, largura de banda e confiabilidade e software no lado cliente.

2.2.3 Distribuição para maior disponibilidade e performance

Na estratégia de distribuir para maior disponibilidade e performance, o que temos é a vantagem de diversos servidores contra um único servidor central ou mesmo mainframe. Se temos um único ponto de falha com servidor central, com múltiplos servidores teremos uma disponibilidade muito maior pois em caso de queda de um dos servidores ainda teremos a disponibilidade de atendimento por outros servidores. Quanto à performance, um único equipamento tem um limite na sua performance enquanto que com diversos servidores teremos a possibilidade de balancear o acesso aos servidores o que nos dará uma performance muito superior a de um único servidor central.

2.2.4 Distribuição para modularidade

Mesmo para uma empresa que tenha todas as facilidades de computação centralizadas, onde todos os usuários estão no mesmo ambiente ainda assim a

modularidade é relevante como uma das estratégias. Mesmo com um custo superior se comparado a um único sistema integrado, na modularidade podemos ter benefícios no gerenciamento das incertezas quanto ao crescimento da organização e sua atividade (geografia, escala, reengenharia, reorientação, mercados, etc). Em atendendo a modularidade, adicionalmente teremos as vantagens discutidas anteriormente de disponibilidade e performance.

2.2.5 Distribuição para a descentralização

Diversas atividades conduzidas por pessoas nas organizações, são descentralizadas pela natureza da própria atividade. Antes dos sistemas distribuídos, ou tínhamos estas facilidades de computação em um computador central, ligando terminais remotos aos locais do usuário ou tínhamos ilhas de processamento com computadores isolados atendendo estas aplicações descentralizadas. Com a estratégia de distribuir para a descentralização, passamos a ter integração e ao mesmo tempo a disponibilidade em áreas descentralizadas das empresas. Esta estratégia permite descentralizar o controle, colocando ele onde for necessário, enquanto mantém o grau de integração e coordenação entre os diferentes pontos distribuídos.

2.2.6 Distribuição para segurança.

A segurança pode ser aumentada se temos distribuição. Em áreas de proteção e criptografia baseadas em servidores de votação temos mais robustez se comparado a sistemas de um único servidor. Sistemas de arquivamento baseados em fragmentação e espalhamento em diversos servidores de arquivos, aumentam a segurança se comparado a servidor único que contem o arquivo inteiro.

2.3 Classificação de Sistemas Distribuídos e Paralelos

Segundo Buyya (1999), “a principal razão para a criação e o uso de computadores paralelos é que o paralelismo é uma das melhores formas de resolver o problema de gargalo de processamento em sistemas de um único processador”. A relação preço performance de um pequeno sistema de processamento paralelo em cluster ´muito

melhor se comparado com um minicomputador. O desenvolvimento e a produção de sistemas de moderada velocidade usando arquitetura paralela é muito mais barato do que o seu equivalente em performance para sistemas de um único processador. A taxonomia das arquiteturas de computadores paralelos, de acordo com seus processadores, memória e forma de interconexão podem ser (BUYYA, 1999):

- MPP (Massively Parallel Processors) Processadores Paralelos
- SMP (Symmetric Multiprocessors) Multiprocessadores simétricos
- CC-NUMA (Cache-Coherente Nonuniform Memory Access) Processador com memória cache e que também acessa as memórias dos outros processadores
- Sistemas Distribuídos
- Cluster

O Quadro 1 mostra uma comparação das características funcionais e arquitetura destas máquinas.

Quadro 1 Características principais dos computadores paralelos

Características	MPP	SMP CC-NUMA	Cluster	Distribuição
Número de Nós	0(100)- (1000)	0(10)-0(100)	0(100)ou menos	0(10)-0(1000)
Complexidade do Nó	Granularidade fina ou média	Granularidade Média ou grossa	Granularidade média	Alcançe largo
Comunicação inter Nos	Troca mensagens / variáveis compartilhadas	Centralizada e Memória compartilhada distribuída	Troca de mensagens	Arquivos compartilhados, RPC , troca mensagens e IPC
Controle de jobs	Fila única no host	Fila única geralmente	Múltiplas filas mas coordenadas	Filas independentes
Suporte a SSI Imagem de Sistema Único	Parcialmente	Sempre no SMP e alguns NUMA	Desejável	Não

Copias de SO e tipo	N micro-kernel monolítico ou SO em camadas	Um monolítico SMP e muitos para NUMA	N SO plataforma homogênea ou micro-kernel	N SO plataforma homogêneas
Espaço endereçamento	Múltiplo – Simples para DSM	Simples	Múltiplo ou Simples	Múltiplo
Segurança entre os nós	Não necessário	Não Necessário	Requer se exposto	Requer
Onde se aplica	Uma Empresa	Uma Empresa	Uma ou mais empresas	Muitas empresas

A classificação de Buyya (1999) segue a classificação de Tanenbaum (1995) que é mais geral e que refina a classificação apresentada por Flynn para arquitetura de computadores. A taxonomia de Flynn leva em consideração fluxo de instrução e fluxo de dados, como segue:

- SISD – Single Instruction Stream, Single Data Stream – Um único fluxo de instruções e um único fluxo de dados.
- SIMD – Single Instruction Stream, Multiple Data Stream – Um único fluxo de instruções e múltiplo fluxo de dados.
- MIMD – Multiple Instruction Stream, Multiple Data Stream – Múltiplo fluxo de instruções e múltiplo fluxo de dados.

Para Tanebaum (1995) as maquinas com múltiplo fluxo de instruções e múltiplo fluxo de dados ainda se dividem nos chamados **multiprocessadores,** para os que têm memória compartilhada e **multicomputadores** para os que não tem memória compartilhada. Tanto os multiprocessadores como os multicomputadores podem ser divididos em duas categorias tomando por base a arquitetura de rede de interconexão sendo uma do tipo baseada em barramento e outra comutada respectivamente. Os multiprocessadores cuja conexão é feita com barramento, ou sejam um backplane de alta velocidade são também chamados de **fortemente acoplados** e os multicomputadores interligados via rede ou outros dispositivos comutados em baixa velocidade são também chamados de **fracamente acoplados**. A Figura 2 mostra esta classificação para os sistemas distribuídos e paralelos.

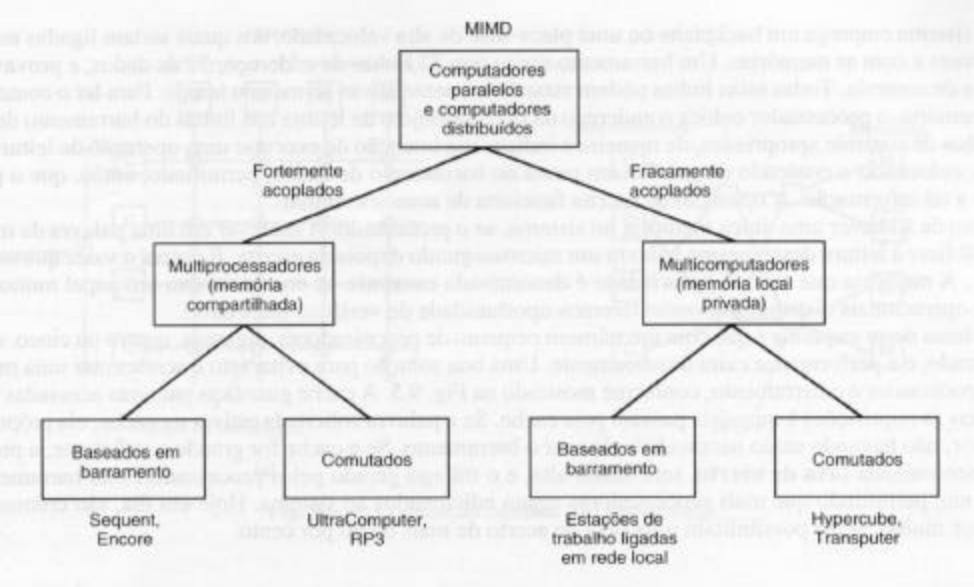

Figura 2 Uma taxonomia para sistemas distribuídos e sistemas paralelos (TANENBAUM, 1992)

2.3.1 Multiprocessadores

A característica básica dos multiprocessadores é que a comunicação entre os processadores se dá através de memória compartilhada. Uma área única de memória sendo acessada simultaneamente por diversos processadores pode causar um gargalo. Assim o conceito de memória de alta velocidade de acesso também conhecido como memória *cache*, passa a ser adotado, tendo cada processador a sua memória *cache* própria. Para manter todos os *cache* atualizados uma técnica chamada write-trhough onde toda a gravação é feita na memória principal e a leitura na memória *cache*. Também todas as *cache* ficam monitorando o barramento e se uma alteração ocorre na memória principal cujo endereço também está na *cache* de um processador este atualiza o conteúdo da *cache*. A Figura 3 ilustra um sistema distribuído baseado em barramento.

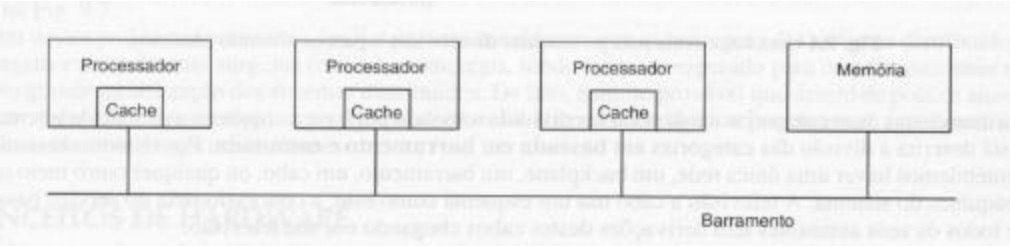

Figura 3 Sistema distribuído baseado em barramento (TANENBAUM, 1992)

Os sistemas multiprocessadores podem também ser ligados por comutação. Assim dividimos a memória em blocos e o acesso dos processadores aos blocos de memória pode ser feito através de comutação *crossbar* como mostra a Figura 4. Em cada intersecção existe uma chave eletrônica que pode ser aberta ou fechada por hardware.

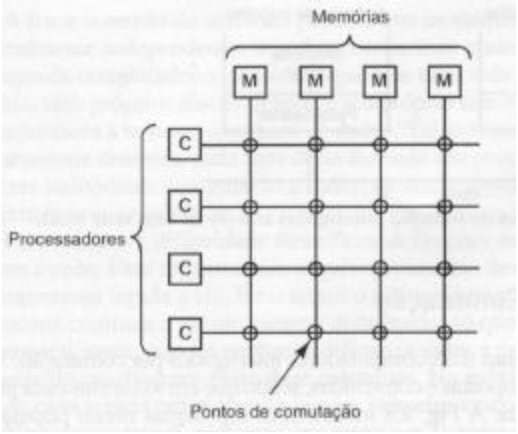

Figura 4 Comutação crossbar (TANENBAUM, 1992)

Outra alternativa é usar uma solução denominada rede ômega onde cada chave tem duas entrada e duas saídas reduzindo o número de comutações de n^2 para nlog2n. A Figura 5 mostra esta alternativa para n igual a 4.

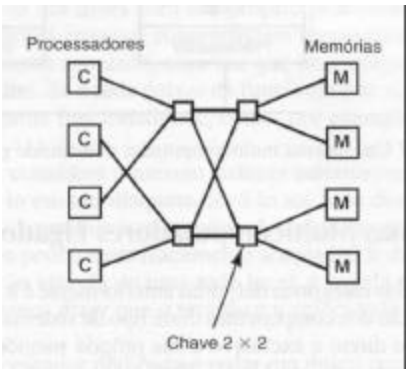

Figura 5 Rede omega (TANENBAUM, 1992)

Esta alternativa exige chaves de comutação muito rápidas e se tivermos um número de processadores muito grande as chaves de comutação se tornam um gargalo.

2.3.2 Multicomputador

Os sistemas multicomputadores, onde não temos memória compartilhada entre os processadores, não apresentam as dificuldades citadas anteriormente. Para os sistemas multicomputadores ligados em barramento a interconexão pode ser feita via rede local onde a velocidade de placas de 100 a 1000 Mbps é suficiente para o tráfego de comunicação processador-a-processador. Esta conexão é suportada inclusive entre redes de maior distância com largura de banda adequada, usando tecnologia como ATM ou Gigabit Ethernet. Na Figura 6 podemos ver um conjunto de estações de trabalho ligada em rede que ilustram este sistema multicomputador baseado em barramento.

Figura 6 Sistema constituído por computadores interligados em rede (TANENBAUM, 1992)

Outra forma de ligação de multicomputadores é a interligação por comutação. Duas topologias são apresentadas por Tanenbaum (1992) sendo a primeira chamada de topologia em grade, facilmente implementada em placas de circuito impresso, e mostrada na Figura 7.

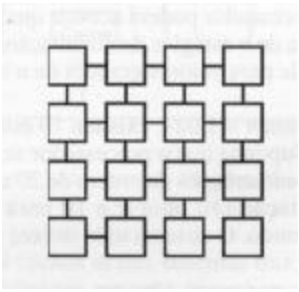

Figura 7 Topologia em grade (TANENBAUM, 1992)

A segunda topologia é chamada de hipercubo onde os vértices representam os processadores e os lados às interligações entre os processadores. Na Figura 8 uma representação para este esquema com dimensão 4 onde cada processador está ligado a outros 4 processadores.

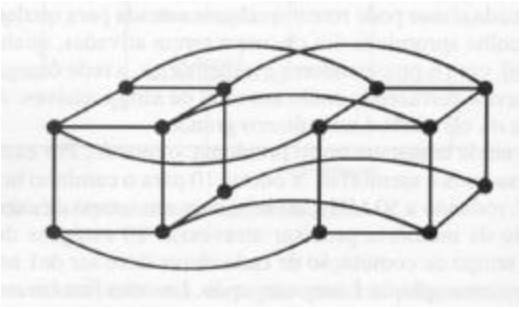

Figura 8 Topologia em hipercubo (TANENBAUM, 1992)

Nesta forma de hipercubo à medida que aumentamos o número de processadores maior fica o caminho para os processadores se comunicarem entre si, pois a interligação não é direta e sim via os processadores que são vizinhos na interligação.

2.3.3 Cluster

Segundo Buyya (1999) “cluster é um tipo de sistema de processamento distribuído ou paralelo, o qual consiste de um conjunto de computadores interligados trabalhando juntos como se fosse um único recurso de computação integrada”. Um nó pode ser um computador multiprocessado ou monoprocessado (PC, estação de trabalho ou SMP) com memória, facilidade de I/O e sistema operacional. Um cluster geralmente se refere a dois ou mais computadores (nós) conectados entre si. Os nós podem estar em

um único gabinete ou interligados via LAN. A Figura 9 mostra uma arquitetura típica de um cluster (BUYYA, 1999).

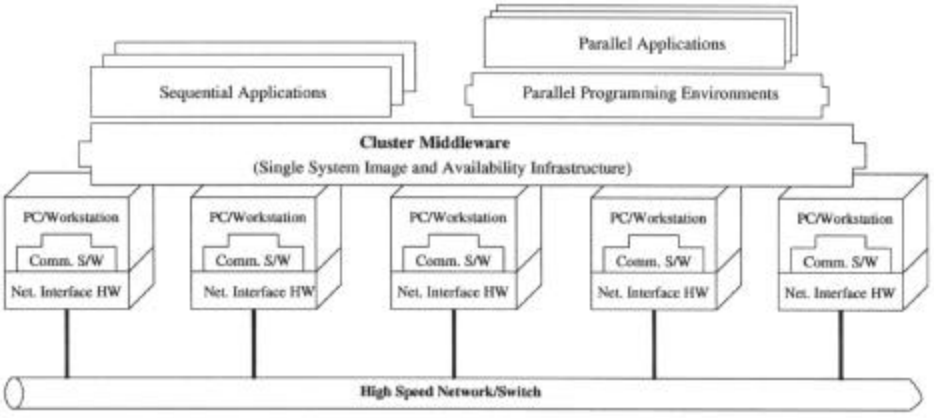

Figura 9 Arquitetura Típica de um Cluster

Cluster oferece as seguintes facilidades a um custo relativamente baixo:

- Alta performance
- Escalabilidade e facilidade de expansão
- Alta performance de processamento
- Alta disponibilidade

A tecnologia de cluster permite aumentar o poder de processamento usando tecnologia comum a preços relativamente baixos. Isto devido ao uso de hardware e software que são padrões de mercado e com preços muito abaixo dos convencionais então existentes. Um benefício muito importante quando usamos cluster é a capacidade de suportar falhas pois se um nó do cluster deixa de funcionar o sistema como um todo ainda continua funcionando apenas com uma performance inferior devido ao nó que deixou de processar.

A interconexão entre os nós de um sistema em cluster pode ser feita usando redes de alta velocidade com protocolo standard como TCP/IP ou protocolos de baixo nível como Mensagens Ativas. O mais comum é que se use placa de rede do tipo Ethernet. Em termos de performance, latência e largura de banda, esta tecnologia está evoluindo. Uma conexão Ethernet simples não pode ser usada como base para uma ligação cluster pois a latência e a largura de banda não estão de acordo com o poder computacional que

temos nas atuais estações de trabalho. Devemos esperar que na interconexão de cluster tenhamos largura de banda superior a 10 Mbytes/s e tempo de latência menor que 100 µs. Algumas das tecnologias de rede de alta performance, segundo Buyya (1999) são apresentadas a seguir:

- **Ethernet, Fast Ethernet e Gigabit Ethernet** - A Ethernet tem se tornado um padrão em redes de estações de trabalho. Inicialmente tínhamos as placas de 10 Mbps, seguidas pelas placas Fast Ethernet com velocidade 100 Mbps usando Hubs ou Switch e atualmente, o estado da arte, é a placa Gigabit Ethernet que mantêm a mesma simplicidade da arquitetura Ethernet permitindo velocidades da ordem de Gigabit por segundo. Podemos também agregar múltiplas placas, formando um fluxo único com a transmissão de diversas placas simultaneamente, chamado TRUNKING, e teremos uma interconexão de alta velocidade entre as estações da rede.
- **Asynchronous Transfer Mode (ATM)** - ATM é uma tecnologia de circuito virtual chaveado, originalmente desenvolvida para a industria de telecomunicações. Baseado na transmissão de pequenos pacotes de dados chamados células pode ser usado tanto em LAN como WAN. As placas com tecnologia ATM têm um preço muito caro e não são largamente adotadas.
- **Scalable Coherente Interface (SCI)** - O objetivo do padrão SCI, ANSI/IEEE 1596-1992, é essencialmente prover um mecanismo de alta performance que suporte o acesso à memória compartilhada coerente, entre um grande número de máquinas. Esta transferência é feita com um tempo de espera (*delay*) de poucos µss. Esta tecnologia não é largamente empregada, seu custo é bastante alto e podemos encontrar em algumas arquiteturas proprietárias (DIETZ, 1998) de memória logicamente compartilhada e fisicamente distribuída, tais como HP/Convex Exemplar SPP e Sequent NUMA-Q 2000.
- **Myrinet** - É uma solução proprietária da empresa Myricom (2001), de alta performance para interconexão a 1.28 Gbps full duplex. Myrinet usa switches roteadores cut-through de baixa latência, oferecendo tolerância à falha pelo mapeamento automático da configuração de rede. Tem baixo tempo de latência (5 µs sentido único, ponto a ponto), excelente

performance e grande flexibilidade devido ao processador programável em sua placa.

Outras tecnologias de rede e formas de interligação de computadores tais como, ArcNet, CAPERS, FC (fibre Channel), FireWire (IEEE 1394), HiPPI e serial HiPPI, IrDA (Infrared Data Association), Parastation, PLIP, SCSI, ServNet, SHRIMP, SLIP, TTL_PAPERS, USB e WAPERS estão descritas em (DIETZ, 1998).

2.4 Software em Sistemas Distribuídos

O software em Sistemas Distribuídos seja aplicativo ou mesmo um sistema operacional, tem que estar de acordo com as características dos Sistemas Distribuídos. Diferentemente de um sistema monoprocessado no ambiente de Sistemas Distribuídos existem as necessidades de sincronismo, interação, controles, entre os diferentes ambientes de software, ou diferentes instancias, que compõem o sistema total. Para o usuário final tudo deve parecer como se fosse um único recurso de computação que está sendo usado para o processamento das necessidades do usuário. Neste contexto vamos fazer a seguir algumas considerações sobre os sistemas operacionais e sobre a programação de aplicativos.

2.4.1 Sistemas Operacionais

Segundo Buyya (1999), um sistema operacional moderno provê dois serviços fundamentais para o usuário. Primeiro ele permite utilizar o hardware de um computador mais facilmente criando uma máquina virtual que difere da máquina real facilitando o uso da mesma por usuários finais. Segundo, um sistema operacional compartilha recursos de hardware entre os usuários. Um dos mais importantes recursos é o processador. Um sistema operacional que trabalha com mais de um processo simultaneamente (multitask), como o Unix ou Windows NT, divide o trabalho que necessita ser executado pelo processador, dando a cada processo memória, recursos do sistema e uma fração de tempo do processador. O sistema operacional executa um processo (thread) por um curto espaço de tempo e depois muda para outro, executando um a um sucessivamente. Assim mesmo em um sistema com um único processador, temos a impressão de múltiplas execuções simultâneas, pois cada processo está sendo

atendido pelo processador em pequenas frações de tempo. Um usuário pode editar um documento, enquanto um relatório está sendo impresso e uma compilação está sendo executada. Para o usuário tudo se parece como se os três programas estivessem rodando simultaneamente.

De acordo com Tanenbaum (1992) podemos distinguir dois tipos de sistemas operacionais para sistemas com vários processadores: os sistemas operacionais fracamente acoplados e os fortemente acoplados. Sistemas fracamente acoplados são aqueles onde temos as estações independentes ligadas por rede local. Cada estação tem o seu sistema operacional. Os nodos têm baixo grau de interação e interagem quando necessário. Se a rede local cai, aplicações individuais em cada estação podem continuar. Sistema fortemente acoplado é aquele onde o software integra, fortemente cada nodo da rede. Se a interligação dos processadores for interrompida, a aplicação também será descontinuada. Segundo os conceitos já apresentados temos:

- **Sistemas Operacionais de Rede** – Software fracamente acoplado em hardware fracamente acoplado.
- **Sistemas Operacionais Distribuídos** – Software fortemente acoplado em hardware fracamente acoplado.
- **Sistemas Operacionais de Multiprocessadores** – Software fortemente acoplado em hardware fortemente acoplado.

Sistemas Operacionais (SO) de Rede permitem compartilhar diferentes recursos de uma determinada estação entre os demais membros da rede. Um dos recursos mais importantes para ser compartilhado é o disco, permitindo que múltiplas estações possam acessar uma única base de dados. Diversos produtos comerciais estão disponíveis, entre eles Rede Novell da Novell, Inc, Windows NT da Microsoft e NFS (Network File System) da SUN Microsystems. O NFS se tornou um padrão de mercado e hoje todos os UNIX têm uma implementação de NFS. Permite que uma determinada área de disco, um diretório ou conjunto de diretórios, seja exportado para outras máquinas da rede. No servidor existe um arquivo de configuração onde se dá permissão de acesso explicitando qual área e que modo de acesso cada máquina cliente vai estar autorizada nesta interligação. Na máquina cliente se monta um determinado diretório da máquina servidora sob o sistema de arquivos local e a área compartilhada para ser vista como se fosse parte da estrutura de arquivos do sistema local. Qualquer máquina com S.O. Unix pode ser cliente e/ou servidora de um sistema NFS e a Figura 10 (TANENBAUM,

1992) mostra a estrutura em camadas dos componentes envolvidos em uma ligação Cliente / Servidor.

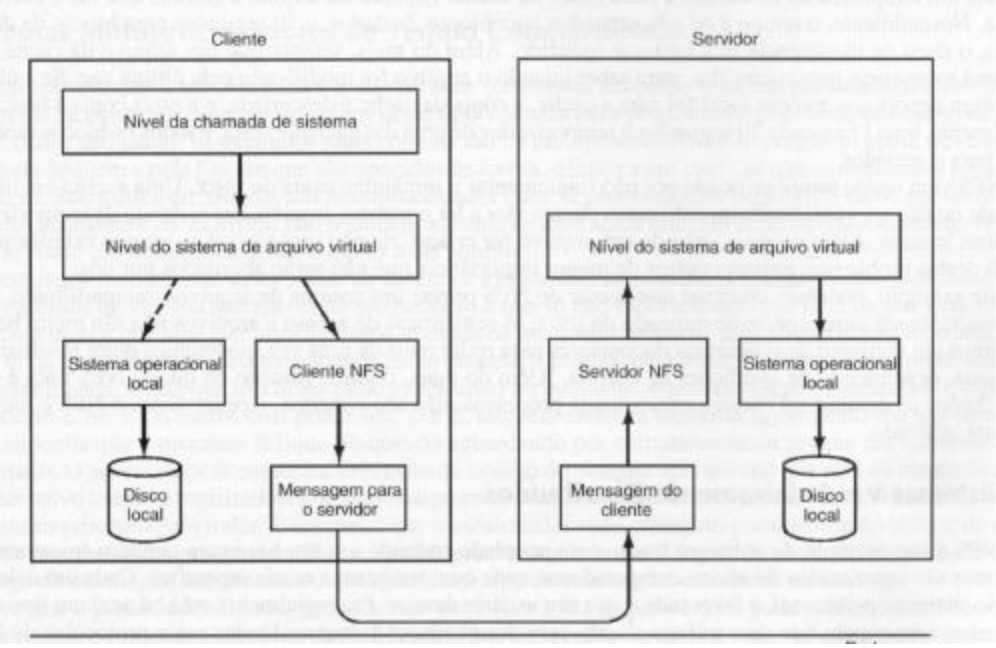

Figura 10 Estrutura em camadas do NFS

Sistemas verdadeiramente distribuído, vão alem do simples uso do NFS. "Um sistema distribuído é aquele que roda em um conjunto de máquinas sem memória compartilhada, máquinas estas que mesmo assim aparecem como um único computador para seus usuários" (TANENBAUM, 1992). Deve haver um sistema de comunicação entre processo, único e global que permita que qualquer processo fale com qualquer outro. Também deve haver um esquema de proteção global. A gerência dos processos também precisa ser a mesma em todo o sistema. A forma como um processo é criado, destruído, inicializado e finalizado, não pode variar de uma máquina para outra.

Sistemas Multiprocessadores de tempo compartilhado, combinação de software fortemente acoplado com hardware fortemente acoplado, são voltados para processamentos específicos. A característica fundamental desta classe de sistema é a existência de uma única fila de processos prontos: uma lista de todos os processos do sistema que não estão bloqueados, e que só não estão rodando por falta de processador disponível O sistema da Figura 11 (TANENBAUM, 1992) mostra uma situação com três processadores e cinco processos prontos para rodar.

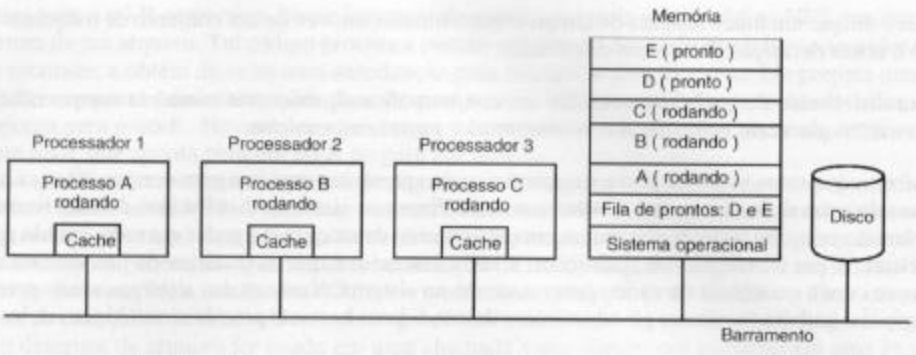

Fig. 9.12 Um multiprocessador com uma única fila de processos prontos.

Figura 11 Um multiprocessador com uma única fila de processos prontos

Segundo Stallings (1998) um sistema operacional de redes não é atualmente um sistema operacional, mas sim um conjunto distribuído de softwares de sistema para suportar o uso de servidores em uma rede. A máquina servidora provê serviços para a rede como gerenciamento de impressora e compartilhamento de arquivos. Cada computador da rede tem seu próprio sistema operacional. O sistema operacional de redes é simplesmente um complemento ao sistema operacional local que permite a interação entre as máquinas de aplicação com a máquina servidora. Já um sistema operacional distribuído é um sistema compartilhado por uma rede de computadores. Aparece para o usuário final como se fosse um único sistema operacional, mas fornece acesso transparente a recursos de diversas máquinas. Um sistema operacional distribuído pode aproveitar uma arquitetura de comunicação já existente, no entanto mais comumente um conjunto de instruções básicas para funções de comunicações é incorporado ao sistema operacional de forma a prover mais eficiência.

A implementação do sistema uniprocessador virtual só é possível se tivermos memória compartilhada para serem acessadas pelos multiprocessadores. Esta solução para redes de computadores fica muito difícil, senão impossível, devido à baixa velocidade e alta latência no processo de intercomunicação entre os processadores.

2.4.2 Ambientes de Programação Paralela e Distribuída.

Do ponto de vista da aplicação, a forma como os programas podem ser desenvolvidos bem como o uso dos recursos existentes, estão relacionados com a arquitetura de processamento paralelo ou processamento distribuído. Segundo Radajewski (1998), temos basicamente duas formas de comunicação entre os processos durante a sua fase de execução:

1. Usando *threads* (linhas de execução, parte do código que roda independente), recurso do sistema operacional, em máquinas de memória compartilhada, onde as comunicações são feitas através da memória diretamente, em máquinas do tipo SMP;
2. Via mensagens, em máquinas onde o acesso à memória é feito através de mensagens que são trocadas entre os processos de cada máquina, exemplo Cluster ou interligadas em rede.

Tanto mensagens quanto *threads* podem ser implementadas em SMP, NUMA-SMP e cluster embora eficiência e portabilidade em cada caso, tenham as seguintes considerações:

- **Mensagens** - historicamente a tecnologia de troca de mensagens refletia o projeto dos primeiros computadores paralelos com memória local. Mensagens requerem que os dados sejam copiados enquanto *threads* usam os dados no mesmo local. A latência e a velocidade com as mensagens podem ser copiadas são os fatores limites neste modelo de troca de mensagens. Uma mensagem é bastante simples: alguns dados e um processador de destino. Alguns mecanismos comuns são interfaces de programas de aplicação tais como PVM (Parallel Virtual Machine) ou MPI(Message Passing Interface). O mecanismo de troca de mensagens pode ser eficientemente implementado usando *threads* e terá bom resultado tanto em máquinas SMP como entre clusters e máquinas em rede.
- ***Threads*** - as *threads* em sistema operacional foram desenvolvidas devido ao projeto de compartilhamento de memória o que permitiu uma comunicação em memória muito rápida e sincronização entre as partes de programas concorrentes. As *threads* funcionam bem em sistemas SMP por que a comunicação é feita através da memória compartilhada. O problema com *threads* é a dificuldade de estender o uso além de máquinas SMP pois sendo os dados compartilhados entre CPUs o overhead para atualizar o

cache de memória, coerência de cache, seria muito grande. Para estender *threads* eficientemente alem dos limites do SMP, seria necessário usar a tecnologia NUMA o que é muito caro.

A programação paralela ou programação distribuída terá realmente vantagem com esta arquitetura de sistemas paralelos e sistemas distribuídos se a aplicação for adequada para este ambiente. Inicialmente vamos considerar a diferença entre concorrência e paralelismo. Partes concorrentes são partes de um programa que podem ser executados independentemente. O paralelismo é quando temos partes concorrentes deste programa que podem ser executadas ao mesmo tempo em elementos de processamento separados. A distinção é muito importante pois concorrência é uma propriedade do programa e um eficiente paralelismo é uma propriedade do equipamento. A tarefa do programador é identificar que parte concorrente pode ser executada em paralelo e qual parte não pode.

2.4.3 DCE Ambiente de Computação Distribuída

O Ambiente de Computação Distribuída, (DCE – Distributed Computing Environment) é uma tecnologia de software desenvolvida pelo Open Group (OPEN, 1999) que permite o desenvolvimento de aplicações distribuídas usando sistemas heterogêneos. O Open Group é um consorcio de usuários e de fornecedores de computadores que trabalham em conjunto visando à tecnologia de sistemas abertos. A organização integra as melhores tecnologias e fornece para que sejam adotadas pelas industrias. Através do Open Group, diversos segmentos da empresas trabalham de forma cooperativa para a evolução do DCE. Algumas áreas criticas que são atendidas por esta tecnologia são:

- Segurança
- Internet / Intranet
- Objetos Distribuídos

A tecnologia DCE foi projetada para trabalhar independente de qual sistema operacional ou qual tecnologia de rede está sendo usada pela aplicação. Assim permite uma interação entre clientes e servidores em qualquer tipo de ambiente, mesmo heterogêneo, que a organização possa ter.

O Open Group fornece o código fonte do DCE os quais os fornecedores incorporam a seus produtos. Muitos fornecedores já agregam funções básicas do DCE junto com os seus sistemas operacionais. A tecnologia inclui serviços de software que

residem acima do sistema operacional, fornecendo interface (*middleware)* para os recursos de baixo nível do sistema operacional e recursos de rede. Estes serviços permitem que a organização possa distribuir o processamento e os dados através de toda a empresa. Atualmente DCE é o único ambiente integrado de serviços, não vinculado a um fornecedor específico, que permitem as organizações desenvolverem, usarem e manterem aplicações distribuídas através de redes heterogêneas.

Os servidos DCE incluem:

- RPC – Chamada de procedimento remoto, o que facilita a comunicação cliente-servidor, e então uma aplicação pode efetivamente acessar recursos distribuídos através da rede.
- Serviço de Segurança – Autentica a identidade dos usuários, autoriza o acesso a recursos em redes distribuídas e provê a contabilização de usuários e servidores.
- Serviço de Diretórios – Provê um modelo único de nome através do ambiente distribuído.
- Serviço de Relógio – Sincroniza os relógios dos sistemas através da rede.
- Serviços de *Threads* – Provê a capacidade de execução de múltiplas *threads*.
- Serviço de Arquivos Distribuídos – Provê acesso a arquivos através da rede.

Com relação à segurança, DCE é um dos mais seguros ambientes de computação distribuída. Incorpora a tecnologia Kerberos, uma forma, altamente confiável, bem gerenciada e bem entendida de proteger computação em rede.

Interagindo com a tecnologia hoje existente para internet/intranet a tecnologia para servidores seguros WEB prevê uma integração com os serviços de nome e serviços de segurança do DCE. Assim uma integração do DCE com Secure-http complementando as facilidades de segurança já existentes com autenticação de usuários para acesso a serviços WEB de informações restritas.

Com relação à tecnologia de objetos distribuídos, DCE lidera a interoperabilidade de diferentes estratégias para objetos distribuídos. DCE permite o uso de objetos distribuídos sem desconsiderar propostas como CORBA. De fato DCE tem os pré-requisitos das especificações CORBA como um ambiente específico Inter Orb Protocol

que já está disponível, testado e em uso atualmente. DCE é um componente chave para muitos vendedores implementarem tecnologias de orientação a objetos. Como exemplo a IBM tem acordo para suportar os serviços de segurança, nome e relógio do DCE em seu SOM/DSOM. Hewlett-Packard esta fornecendo DCE++, uma ferramenta orientada a objeto que provê um compilador C++IDL e biblioteca de classes para DCE. A Microsoft usa DCE como base para a comunicação com ActiveX. Digital usa a segurança do DCE em seu ObjectBroker, seu CORBA. Muitos fornecedores têm expressado a sua intenção em usar DCE RPC como um protocolo específico dentro da especificação CORBA para prover interoperabilidade.

2.4.4 Programação Distribuída Orientada a Objeto

A programação orientada a objeto é considerada atualmente um dos melhores modelos de programação para tratar com sistemas complexos enquanto provê facilidade de manutenção, novas implementações e reusabilidade. Um modelo contendo estes atributos é particularmente interessante para sistemas distribuídos, já que estes tendem a se tornar muito complexos. A base da programação orientada a objeto é o conceito de objeto, que é uma entidade que encapsula um estado e permite o acesso através de uma interface bem definida.

Programação distribuída orientada a objeto generaliza a programação orientada a objeto para sistemas distribuídos, estendendo o modelo de objeto com interação remota cliente/servidor entre objetos não locais. No modelo cliente/servidor, o servidor provê acesso aos clientes para serviços específicos através de mecanismos de intercomunicação de processos, IPC, tais como passagem de mensagens ou chamadas de procedimentos remotos. Na programação distribuída orientada a objeto um servidor de objetos encapsula um estado interno e prove o serviço para uma coleção de objetos clientes remotos. Assim como na pura programação orientada a objeto, este serviço é abstraído através da interface, ou seja, uma coleção de métodos que podem ser usadas para acessar o serviço. Clientes que estão residindo em diferentes maquinas podem invocar estes métodos através da invocação de métodos remotos.

Nos últimos anos tem surgido diversos ambientes de desenvolvimento baseados no paradigma da programação distribuída orientada a objeto. Estes ambientes, entre eles os mais notáveis exemplos são CORBA (OMG, 1998), Java Remote Method Invocation (RMI) (SUN, 1998) e DCOM (Distributed Component Objet Model) (GALLI,1998).

A seguir vamos discutir as principais características destes sistemas.

2.4.4.1 CORBA - Common Object Request Broker Architecture

Existe um consorcio internacional de industrias que promove a teoria e a prática de programação distribuída orientada a objetos que é chamado de OMG – Object Management Group (OMG, 1998). Seu objetivo é definir uma arquitetura padrão e comum para que através de diferentes plataformas de hardware e sistema operacionais, possa ser feita a intercomunicação entre objetos. O consorcio OMG foi fundado em 1989 por oito companhias e atualmente conta com cerca de 800 membros, com a participação dos maiores fabricantes de computadores. Esta organização não desenvolve projetos, trabalha com as tecnologias existentes oferecidas pelas empresas membros do consorcio. Sua forma de trabalho é divulgar RFPs – Request for Proposals em todos os aspectos da tecnologia de objetos solicitando especificações de novos componentes. Membros podem então propor uma especificação que é acompanhada pela implementação provendo o detalhamento dos conceitos inerentes a esta especificação. Um processo de revisão e votação é conduzido e assim que a especificação é aceita, qualquer fabricante pode implementar esta alternativa em seu produto para o mercado.

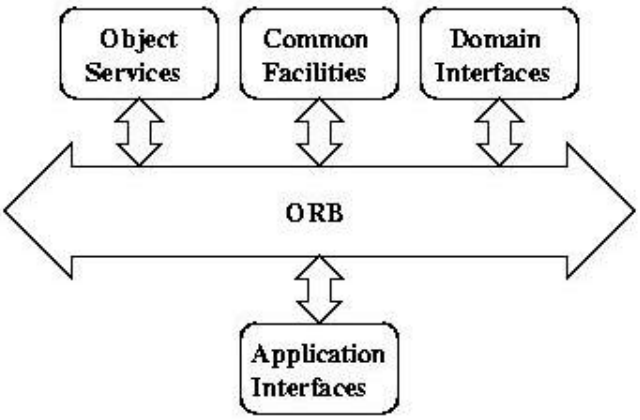

Figura 12 Modelo de Referencia OMG OMA (OMG, 1998)

O consorcio OMG tem desenvolvido um modelo conceitual, conhecido como *core object model*, e uma arquitetura de referencia, chamada OMA – *Object Management Architecture* sobre a qual as aplicações podem ser construídas. A OMG OMA se

propõem a definir em alto nível de abstração, as muitas facilidades necessárias para a computação distribuída orienta a objetos. Assim é proposto um modelo de referencia OMA com cinco partes principais, representados na Figura 12, como segue:

- ORB – Object Request Broker, (também comercialmente conhecido como CORBA) que é um barramento comum de comunicação para os objetos. Permite que objetos clientes de forma transparente e confiável, requisitem operações em objetos remotos e recebam respostas em um ambiente distribuído. A integração de objetos distribuídos está disponível através de plataformas, independente de sistemas operacionais e serviços de transporte da rede.
- Serviço de Objetos – são componentes de uso geral fundamentais para o desenvolvimento de aplicações CORBA. Um serviço de objetos é basicamente um conjunto de objetos CORBA que podem ser requisitados através do ORB. Serviços não estão relacionados com nenhuma aplicação especifica mas são blocos de construção básica, usualmente provida no ambiente CORBA, suportando funcionalidades básicas úteis para muitas aplicações. Diversos serviços, tem sido projetados e adotados como standard pela OMG (OMG, 1998), incluindo notificação de eventos, interfaces de processamento de transação, segurança, controle de autenticação, etc.
- Interfaces de Domínio – representam áreas verticais que provêem funcionalidades de interesse direto para usuários finais em específicos domínios de aplicação tais como financeiro ou saúde.
- Facilidades Comuns – provêem ao usuário final funções úteis em diversas aplicações que podem ser configuradas para uso específico em aplicações particulares. São facilidades relacionadas com o usuário final, tais como serviço de impressão, gerenciamentode documentos, e correio eletrônico.
- Interface de Aplicação – são específicas para aplicações de usuário final.Representam aplicações baseadas em componentes, executando tarefas especificas para o usuário. Uma aplicação é tipicamente composta de um grande numero de objetos, alguns dos quais são específicos da aplicação e outros parte dos serviços de objetos, facilidades comuns ou interface de domínios.

Todos os serviços OMG devem ser especificados usando OMG – IDL (Interface Definition Language). Uma linguagem puramente declarativa e não provê detalhes da implementação. É uma linguagem de programação neutra, independente de rede e assim usada como uma forma de descrever os tipos de dados. A sintaxe do OMG IDL é derivada do C++, removendo os construtores de uma linguagem de implementação e adicionando um numero de novas palavras chaves necessárias para a especificação de sistemas distribuídos.

Como descrevemos CORBA (COULOURIS, 2001) "é um *middleware* que permite aos programas de aplicação se comunicarem entre si, independente de suas linguagens de programação, sua plataforma de hardware e de software e a rede sobre a qual eles se comunicam". As aplicações são construídas com objetos CORBA os quais implementam interfaces definidas na linguagem de definição de interfaces (IDL). Clientes acessam os métodos nos IDL interfaces dos objetos CORBA por meio de invocação método remota (RMI). O componente de *middleware* que suporta RMI é chamado de distribuidor de requisição de objetos (ORB – Object Request Broker). A especificação do CORBA foi patrocinada por membros do grupo OMG – Object Management Group. Muitos diferentes ORBs foram implementados segundo a especificação, suportando uma variedade de linguagens de programação. Os serviços CORBA provêem facilidades genéricas que podem ser usadas em uma grande variedade de aplicações.

2.4.4.2 JAVA Remote Method Invocation

Java Remote Method Invocation (RMI) é um modelo de objeto distribuído para a linguagem Java que mantém a semântica do modelo de objeto Java, tornando a distribuição de objetos fácil de implementar e de usar (EMMERICH, 1999). Na terminologia standard Java RMI, um objeto remoto é um cujos métodos podem ser chamados de outra maquina virtual Java. Um objeto deste tipo é descrito por um ou mais interfaces remotas, que são interfaces Java que declaram qual método do objeto remoto pode ser chamado remotamente. RMI refere-se a ação de invocar um método de uma interface remota em um objeto remoto.

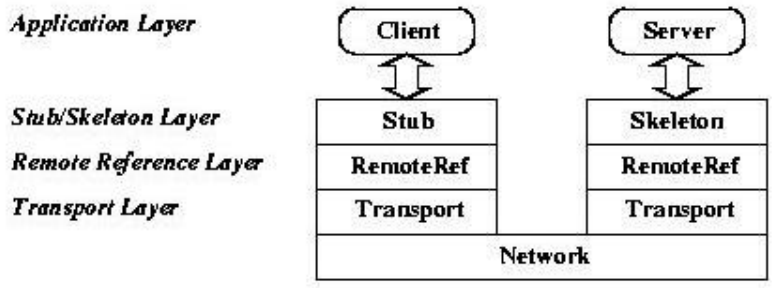

Figura 13 Arquitetura Java RMI

A arquitetura Java RMI é ilustrada na Figura 13. Usa o mecanismo padrão (derivado de RPC) para comunicação com objetos remotos: *stubs* e *skeletons*. Um *stub* para um objeto remoto atua como um cliente local, representativo para o objeto remoto. Clientes nunca interagem diretamente com objetos remotos, mas somente com objetos *stub* os quais são responsáveis por fazer a chamada em seu remoto objeto destino. Um *skeleton* para um objeto remoto é uma entidade no lado do servidor que contem o método que dispara a chamada para a implementação do atual objeto remoto. *Stubs* e *skeleton* são gerados pelo compilador RMIC fornecido com a distribuição padrão do Java RMI.

Um *stub* para um objeto remoto implementa o mesmo conjunto de interfaces remotas implementado pelo objeto remoto. Quando um método do *stub* é chamado, ele faz o seguinte:

- Inicia a conexão com a maquina virtual Java remota que contem o objeto remoto.
- Escreve e transmite os parâmetros para a maquina virtual Java.
- Espera pelo resultado destas chamadas.
- Lê o valor de retorno ou exceção retornada.
- Retorna o valor a quem chamou.

Na maquina virtual Java, cada objeto remoto pode ter um *skeleton* correspondente.(na versão JDK 1.2 *skeletons* não são obrigatórios) O *skeleton* é responsável por disparar a chamada para a atual implementação do objeto remoto. Quando o *skeleton* recebe uma chamada faz o seguinte:

- Lê os parâmetros para o método remoto

- Chama o método na implementação do atual objeto remoto.
- Transmite o resultado para quem chamou.

A implementação da camada de seção no Java RMI suporta a ativação de objetos sob demanda através da interface de ativação. Assim evita que no Java RMI objetos servidores estejam em memória todo o tempo permitindo que sejam carregados dinamicamente, quando necessário. A ativação é completamente transparente e está representada na Figura 14 (EMMERICH, 1999).

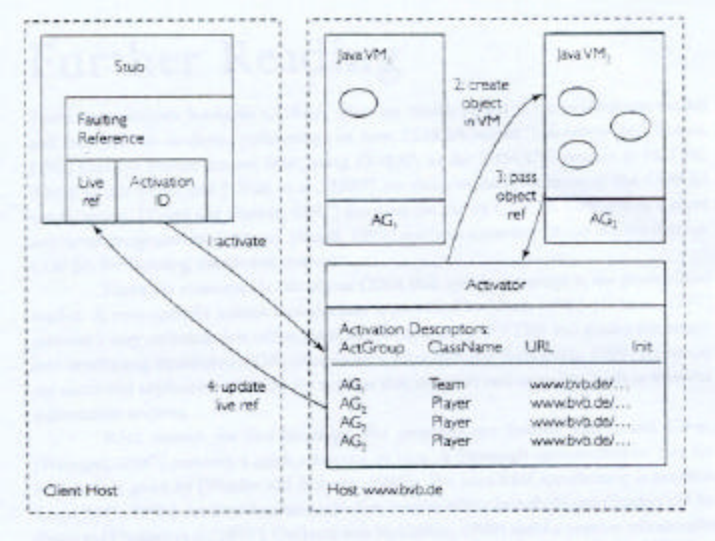

Figura 14 Ativação de objetos remotos em Java RMI

2.4.4.3 DCOM - Distributed Component Object Model

O modelo de objetos componentes distribuídos (Distributed Component Objet Model – DCOM) é o padrão adotado pela Microsoft para distribuição de objetos (GALLI, 1998). Estabelece um modelo de programação, padrão binário, bem como um padrão de interoperabilidade para a computação usando objetos distribuídos. DCOM está disponível desde o Windows NT 4.0 e tinha versões que poderiam ser obtidas na internet para o Windows 95. Está incluído no Microsoft Internet Explorer 4.0, no windows 98 e disponível para diversas plataformas UNIX fornecido por uma empresa chamada Software AG. Embora iniciado pela Microsoft, COM e DCOM não é mais

ambiente proprietário da Microsoft. Atualmente o consorcio independente ActiveX é responsável pelo gerenciamento deste padrão.

O modelo básico COM inclui a habilidade de permitir que elementos lógicos sejam considerados independentes. Também permite componentes binários flexíveis para adaptação a diferentes configurações e maquinas. COM é mais largamente conhecido como a tecnologia básica do ActiveX. Qualquer ferramenta de software que suporta componentes COM, automaticamente suporta DCOM. Existem quatro serviços no ambiente do servidor ActiveX que são particularmente úteis para DCOM: transações, incluindo a capacidade de recuperação e volta ao estado original; enfileiramento com filas confiáveis de armazenamento e envio que permitem operações em redes sujeitas a intermitentes falta de acesso; uso de script no servidor para permitir fácil integração com aplicações de internet baseadas em HTML; e acesso a sistemas de produção legados. DCOM permite que a tecnologia COM possa se comunicar diretamente através de redes. Alguns pontos importantes do DCOM são:

- Independência de transporte – DCOM permite que os componentes se comuniquem, seja com ou sem orientação a conexão. Suporta TCP/IP, UDP/IP , IPX/SPX, Apple Talk e HTTP.
- Tecnologia aberta – DCOM é uma tecnologia aberta e está disponível para UNIX, Apple, Windows e alguns ambientes mais antigos. Contudo é mais comum no ambiente Windows.
- Integrado com servidor e browser WEB – Como DCOM inclui ActiveX e os componentes ActiveX podem ser incluídos em aplicações baseadas em browser DCOM permite distribuir aplicações internet que suportam tecnologias de browser.
- Segurança – DCOM pode integrar segurança na internet baseada em certificados.
- Extensão do mecanismo RPC do Ambiente de computação distribuída (DCE) – DCOM usa e estende este mecanismo de RPC do DCE.

A arquitetura básica DCOM permite que uma aplicação possa ser desenvolvida de maneira que automaticamente permita uma distribuição futura e com escalabilidade. Assim se uma demanda maior de uma aplicação acontece podemos aumentar a capacidade do servidor para atender esta necessidade. Entretanto se instalarmos um

novo servidor e a aplicação estiver no padrão DCOM poderemos distribuir a execução sendo que parte da aplicação poderá rodar no servidor antigo e parte no novo servidor. Para se comunicar com um componente que não seja local, DCOM emprega um mecanismo de comunicação inter processos que é completamente transparente para a aplicação. Especificamente DCOM substitui o mecanismo de comunicação local por um protocolo de comunicação em rede. Desta forma DCOM permite independência e transparência de local para a aplicação.

2.4.4.3.1 Monikers

Os nomes de instância em DCOM são referenciados como monikers. São por si só objetos e permitem extrema flexibilidade. Identificadores que vão desde um nome de banco de dados, um servidor, até um endereço URL ou uma pagina HTML podem ser caracterizados como uma instancia de um objeto. Monikers contem as informações necessárias e lógica para localizar uma corrente instancia que está sendo executada do objeto que ele está se referindo. As instancias que estão sendo executadas, são listadas na tabela ROT – Running Object Table. Esta tabela é usada pelo Monikers para encontrar instancias de objetos que estão em execução, rapidamente.

Monikers armazenam informações de referencia sobre o estado de um objeto. Estas informações estão tipicamente armazenadas em um banco de dados. Um objeto pode indicar para o moniker qual é a atividade quando o estado do objeto for armazenado. Assim uma operação de acesso a dados pode prevenir um possível gargalo se o objeto rodar no mesmo servidor onde estão os dados. Se múltiplos clientes necessitam acessar o objeto, o objeto precisa executar em um lugar acessível a todos os clientes.

2.4.4.3.2 Chamadas a Métodos Remotos

DCOM executa uma chamada a método remoto quando um cliente quer chamar um objeto em outro espaço de endereçamento. No DCOM a linguagem de descrição de interface (IDL) foi construída em cima do padrão DCE RPC. Como é usual em RPC todas as informações parâmetros são disponibilizadas após serem concatenadas em um único espaço de memória, sendo passado o endereço desta área. O cliente faz o processo

inverso e a partir do endereço, recria os dados que o processo vai receber. O retorno desta chamada volta com parâmetros segundo o mesmo processo anterior. O lado do cliente é conhecido como *proxy* e o lado do servidor e conhecido como *stub.* Um tipo adiciona de dados não incluído em DCE RPC é o ponteiro de interface. Estes tipos de ponteiros podem aparecer como resultado da CoCreateInstance ou como parâmetro para a chamada de um método. Para tratar este novo tipo de dado a criação de um par proxy/stub é feita com a capacidade de tratar este novo tipo de dados em todos os métodos da interface.

2.4.4.3.3 Coleta de Lixo

O mecanismo primário para controle do tempo de vida de um determinado objeto é um contador de referencia. Este contador incrementa quando se usa AddRef e decrementa com o Release. Como nem todos os clientes terminam normalmente, um processo de Ping é empregado. Cada objeto exportado tem um tempo de PingPeriod e um contador de numPingsToTimeOut. A combinação destes valores determina o tempo total conhecido como período de ping. Se este tempo passa sem que ocorra um ping para o referido objeto, a referencia remota é considerada expirada. O contador de referencia é decrementado como se o processo tivesse terminado normalmente. Quando um objeto não tem referencia ativa, um processo de coletor de lixo se encarrega de liberar a memória então ocupada por este objeto.

2.4.4.3.4 Modelo de Thread Suportado em DCOM

COM e DCOM operam com a capacidade de *multithread* nativa do sistema operacional. Assim a comunicação inter processo no DCOM requer DCE RPC, que por sua vez requer que mensagens RPC sejam processadas em uma *thread* arbitraria. DCOM provê sincronização automática do método de chamada para uma *thread* simples. DCOM usa o modelo apartamento com relação ao tratamento de *threads*. Com relação aos parâmetros podemos pensar em cada *thread* como um processo separado. Acesso na mesma *thread* é acesso direto enquanto de diferentes *thread* é indireto, possivelmente através do proxy/stub. Um objeto pode suportar qualquer dos modelos de *thread* que segue:

- Apartamento Thread única, *thread* principal somente – Neste modelo todas as instancias são criadas na mesma *thread* única associada com este objeto.
- Apartamento Thread única – Neste modelo uma instancia é amarrada a uma *thread* única contudo diferentes instancias podem ser criadas em diferentes *threads*.
- Apartamento Multithread – Neste modelo instancias podem ser criadas em múltiplas *threads* e podem ser chamadas em *threads* arbitrarias.

Qualquer objeto implementa como o servidor local controla o tipo de apartamento usado para seu objeto. Quando um objeto está sendo criado, COM necessita saber se o objeto é compatível com o modelo de *thread* do objeto pai. Se o objeto filho não é compatível, COM tenta carregar o objeto em um apartamento diferente que seja compatível.

2.4.4.3.5 Segurança em DCOM

Assim como qualquer sistema que suporta computação distribuída DCOM também tem seus aspectos de segurança:

- **Segurança de acesso** – Assegura que um objeto possa ser chamado somente por objetos que tenham a permissão apropriada.
- **Segurança no lançamento** – Assegura que somente objetos apropriados possam criar novos objetos em um novo processo.
- **Identidade** – O principio de como um objeto identifica a si mesmo.
- **Políticas de conexão** – Se uma mensagem pode ser alterada e se uma mensagem é capas de ser interceptada por outro objeto.

2.4.4.4 Pontos Chaves de CORBA, JAVA e DCOM

Segundo Emmerich (1999) quando avaliamos Corba, Java e Dcom encontramos os seguintes pontos chaves:

- CORBA, DCOM e JAVA/RMI permitem que objetos clientes requisitem a execução de operações de objetos servidores distribuídos. Esta operações

são parametrizadas e detalhados mecanismos de passagem de parâmetros são especificados.

- CORBA, DCOM e JAVA/RMI todos usam uma forma de referência para identificar objetos servidores de forma transparente quanto à localização. Objetos CORBA são identificados por referencia ao objeto, objetos COM sãi identificados por ponteiros de interface e objetos Java/RMI são identificados por referencias de falta.
- JAVA/RMI é diferente de CORBA e DCOM pelo fato que integra objetos não remotos ao modelo objeto. Isto é obtido modificando o mecanismo padrão de passagem de parâmetros no modelo e passando objetos não remotos por valor.
- O modelo objeto do CORBA, DCOM e JAVA/RMI, todos separam a noção de interface e implementação. A separação é mais específica em CORBA e DCOM, onde existem linguagens separadas para definir interfaces e onde a implementação pode ser escrita em diferentes linguagens de programação.
- Os modelos de objetos dos três ambientes suportam herança. Todos especificam uma classe primária que é usada para derivar propriedades comuns a qualquer objeto referenciado. Esta classe principal é Object para CORBA, IUnknown im DCOM e Remote in Java/RMI.
- Nos três enfoques, atributos são tratados como operações. Pode ser implícito como em CORBA ou explicitamente incorporado pelo projetista usando DCOM ou JAVA/RMI.
- Todos os modelos de objeto têm suporte para tratar com falhas durante a requisição de objetos.
- Todos os modelos de objeto são estaticamente de um tipo (tipo de variável) e suportam uma forma restrita de polimorfismo. Em todos os modelos é possível associar um objeto a uma variável de diferente tipo desde que o tipo da variável estática seja um supertipo do tipo do objeto dinâmico.
- Em todos os três sistemas de *middleware* temos geradores para os clientes e *stubs* no servidor para implementar a camada de apresentação. CORBA usa *stubs* no cliente e implementa *skeletons.* DCOM usa interface *proxie* e

interface *stub.* Em JAVA/RMI eles são referenciados como *stubs* e *skeletons.*

- CORBA, COM e JAVA/RMI todos suportam a ativação sob demanda de objetos servidores. CORBA tem esta facilidade através de adaptador de objeto; em DCOM é implementado pelo gerenciador de controle de serviço; em JAVA/RMI o ativador implementa esta ativação.

2.5 Comunicação nos Sistemas Distribuídos

A comunicação em sistemas distribuídos se torna mais complexa, pois a troca de informações entre dois processos localizados em máquinas distintas depende primeiramente da interligação destas máquinas. Quando em um único equipamento dois processos podem se comunicar através da memória compartilhada que é comum e acessível aos dois processos simultaneamente. Segundo Stallings (1998) a interligação dos equipamentos hoje pode ter vários enfoques desde o tratamento de um computador pessoal como um simples terminal até um alto grau de integração entre aplicações em computadores pessoais e servidores de banco de dados.

Uma das arquiteturas que se tornou padrão de mercado é o TCP/IP (Transmission Control Protocol/Internet Protocol), amplamente usado nos dias atuais e que será visto com mais detalhes na seqüência.

2.5.1 Arquitetura do Protocolo TCP/IP

Segundo Stallings (1998) TCP/IP é o resultado de um protocolo pesquisado e desenvolvido na rede experimental ARPANET, fundada pela DARPA (Defense Advanced Research Projects Agency) e genericamente conhecido como conjunto de protocolos TCP/IP. Este conjunto de protocolos foi transformado em padrões internet pelo IAB (Internet Architecture Board). Não há um modelo oficial de protocolo TCP/IP como no caso do modelo OSI. Contudo, baseados nos protocolos padrões que foram desenvolvidos nos podemos organizar as tarefas de comunicação para o TCP/IP em cinco níveis independentes:

- **Camada de Aplicação** – contém a lógica necessária para suportar as varias aplicações de usuários. Para cada tipo diferente de aplicação, como

exemplo a transferência de arquivos, um módulo separado é necessário para esta aplicação peculiar.

- **Camada de transporte** – Independente da natureza da aplicação, os dados que estão sendo trocados entre as aplicações precisam ser confiáveis. O controle da ordenação dos pacotes para garantir que sejam recebidos na mesma ordem que são transmitidos é uma tarefa que independe da natureza da aplicação. O protocolo de controle de transmissão (TCP) é o mais comum nesta camada.
- **Camada de internet** – Quando temos a interligação de diferentes redes é necessário o uso de rotinas que tratam o roteamento para esta conexão. Os procedimentos para tratar estas funções estão na camada internet e o protocolo IP é usado nesta camada para prover as funções de roteamento através de múltiplas redes.
- **Camada de acesso à rede** – Esta relacionada com a troca de dados entre um sistema final e uma rede ao qual está ligada.O computador que envia precisa prover o endereço da rede de destino. O computador que envia pode requisitar certos serviços, tal como prioridade, que precisam ser atendidos pela rede. O software específico usado nesta camada depende do tipo de rede a ser usado; diferentes padrões foram desenvolvidos para o chaveamento de circuitos, chaveamento de pacotes (X.25), rede local (Ethernet) e outros.
- **Camada física** – atende a interface de ligação física entre a estação de transmissão de dados (computador, estação de trabalho) e o meio de transmissão ou rede. Esta camada está relacionada com características dos meios de transmissão, a natureza dos sinais, a taxa de transferência e outros itens relacionados.

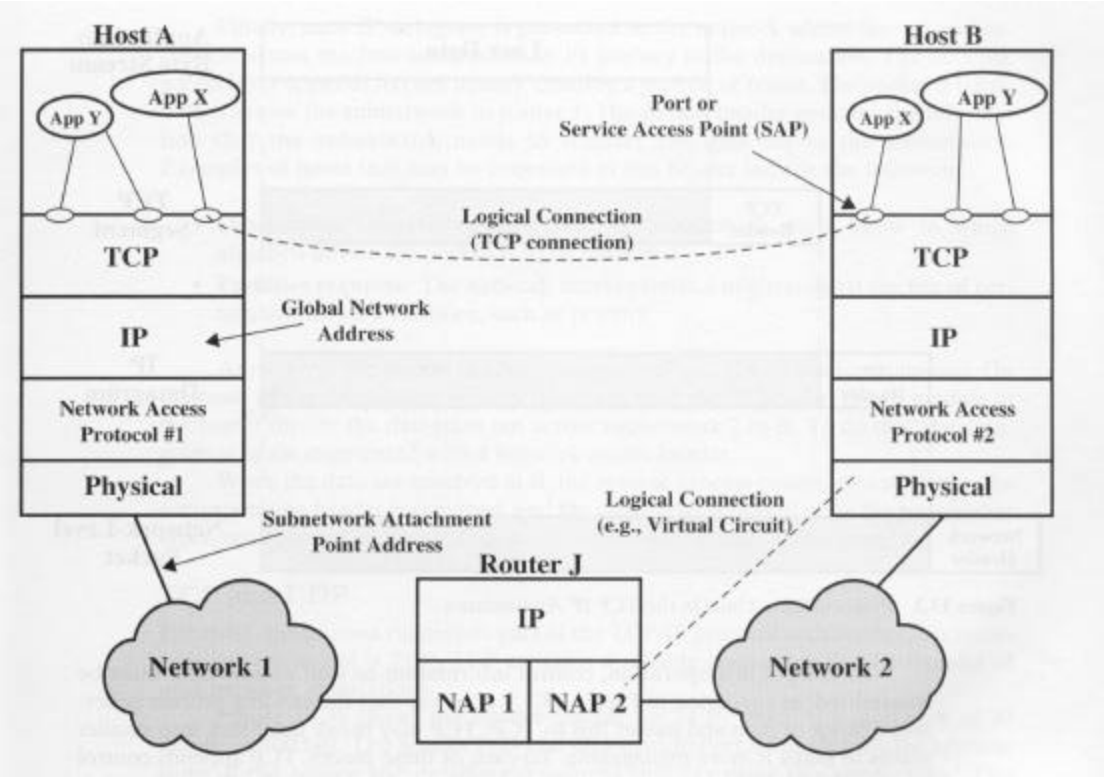

Figura 15 Conceito TCP/IP

Em termos de operação, podemos ver na Figura 15 como estes protocolos são configurados para comunicação. Para ficar claro que as facilidades de comunicação abrangem múltiplas redes a rede onde estamos conectados é chamada de subrede. Assim um protocolo de acesso como o Ethernet é usado para conectar o computador a subrede. Este protocolo permite enviar dados para outro computador na mesma subrede ou a um roteador nesta subrede. O protocolo IP está implementado em todos os sistemas finais e nos roteadores. Atua como um retransmissor para mover os pacotes da origem ao destino final passando por roteadores que interligam as subredes. TCP é implementado somente nos sistemas finais; responsável por controlar os blocos de dados de forma a que sejam entregues corretamente para as aplicações apropriadas.

Para o sucesso da comunicação, cada entidade no sistema como um todo, precisa ter um endereço único. Atualmente dois níveis de endereçamento são necessários. Cada computador em uma subrede precisa ter um endereço global único o que permite que o dado seja enviado ao computador específico. Este endereço é usado pelo IP para roteamento e entrega dos pacotes. Em um computador, cada aplicação precisa ter um endereço único neste computador. Este endereço único relativo a aplicação, também

chamado de porta, permite ao protocolo TCP fazer a entrega das informações ao processo específico.

2.5.2 Modelo Cliente Servidor

Segundo Tanenbaum (1992) uma das estruturas mais comuns para os sistemas distribuídos é a do cliente-servidor. A idéia por trás deste modelo é a de estruturar o sistema operacional como um grupo de processos cooperantes, denominados servidores, que oferecem serviços a processos usuários, denominados clientes. As máquinas clientes e máquinas servidoras normalmente rodam o mesmo microkernel, onde tanto os clientes quanto os servidores rodam como processos usuários. Uma máquina pode rodar um único processo, vários clientes, vários servidores, ou uma mistura dos dois.

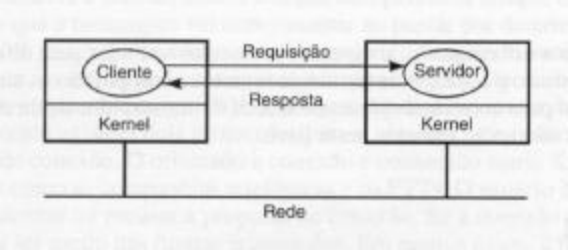

Figura 16 O modelo Cliente Servidor

Para evitar o overhead considerável dos protocolos orientados a conexão tal como o OSI ou o TCP/IP, o modelo cliente servidor é baseado em um protocolo muito simples, sem conexão, do tipo solicitação/resposta. A vantagem deste modelo, Figura 16 é a simplicidade. O cliente envia uma solicitação ao servidor e recebe dele uma resposta. Nenhum tipo de conexão deve ser estabelecido antes do envio da solicitação, nem desfeito após a obtenção da resposta. A própria mensagem de resposta serve como uma confirmação do recebimento da solicitação. Devido a esta estrutura extremamente simples, os serviços de comunicação fornecidos pelo microkernel podem por exemplo, ser reduzidos a duas chamadas de sistema, uma para envio de mensagens e outra para a recepção das mesmas.

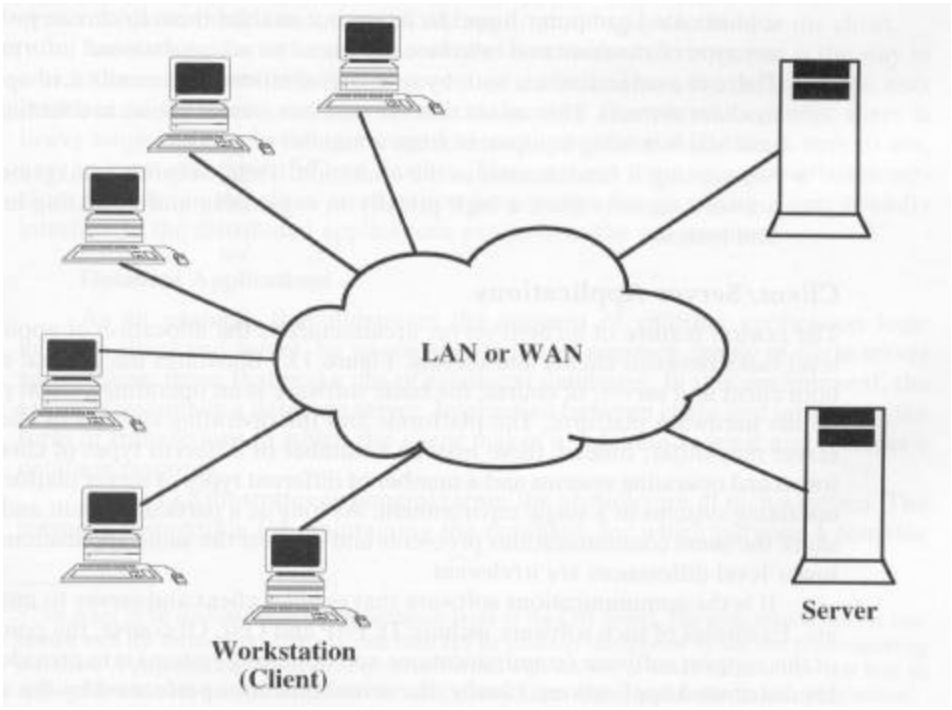

Figura 17 Ambiente genérico para cliente servidor

Segundo Stallings (1998) um exemplo de ambiente genérico para cliente servidor é mostrado na Figura 17. A máquina cliente, geralmente um PC ou estação de trabalho, que tem uma interface muito amigável, geralmente gráfico, para o usuário final. Cada máquina servidora disponibiliza um conjunto de serviços compartilhados para os clientes.

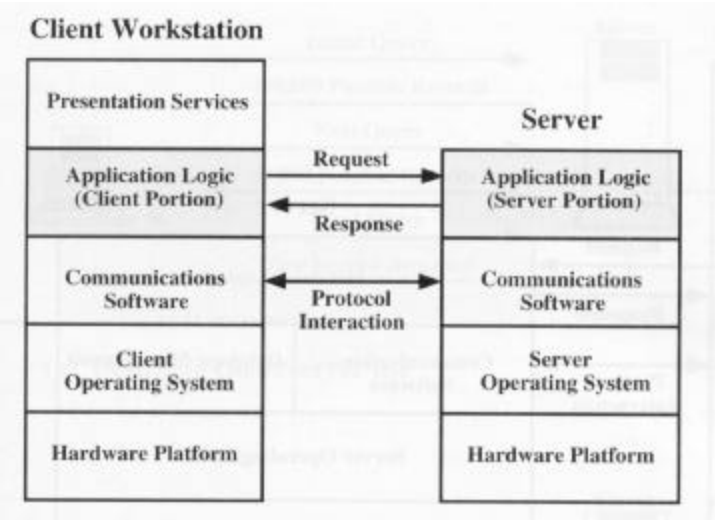

Figura 18 Arquitetura genérica cliente/servidor

A principal característica da arquitetura cliente/servidor é a alocação de tarefas no nível de aplicação entre clientes e servidores. A Figura 18 ilustra o caso geral. Em ambas, cliente e servidor, o software básico é o sistema operacional rodando na plataforma de hardware. A plataforma e o sistema operacional do cliente pode ser diferente do servidor. Assim o importante é que um servidor e um particular cliente compartilhem os mesmos protocolos de comunicação. É o software de comunicação que permite a interoperabilidade entre o cliente e o servidor. Exemplos destes softwares incluem TCP/IP e OSI. Um exemplo que bem ilustra o processamento cliente/servidor é o caso de aplicações que acessam bancos de dados. Usando linguagem SQL (Structured Query Language) um aplicativo cliente pode solicitar dados de tabelas contidas em um servidor de banco de dados. Após processar a requisição o Banco de Dados devolve os resultados para o cliente que ira tratar as informações e mostrar o resultado ao usuário, geralmente em um ambiente de telas gráficas, dando uma melhor apresentação ao resultado final com os dados obtidos.

2.5.3 RPC Chamada Remota de Procedimentos

RPC (Remote Procedure Calls) é um método comum e largamente aceito para o encapsulamento de mensagens em um sistema distribuído (STALLINGS, 1998). A essência desta técnica é permitir que programas em diferentes máquinas possam interagir usando simples semântica de procedimentos call/return, como se os dois programas estivessem na mesma máquina. Isto é a chamada de um procedimento é usada para acessar serviços remotos. A popularidade deste enfoque é devido as seguintes vantagens:

- A chamada de um procedimento é amplamente aceita, usada e de abstração entendida.
- O uso de chamadas remotas de procedimentos permite que interfaces remotas sejam designadas como um conjunto de operações nomeadas com tipos designados. Assim esta interface pode ser claramente documentada e os programas distribuídos podem ser conferidos estaticamente, em tempo de compilação, para validar os erros de tipos (permite conferir os tipos de dados que estão sendo usados nas chamadas, em tempo de compilação)

- Devido à especificação de uma interface padronizada e precisa o código de comunicação para uma aplicação pode ser gerado automaticamente.
- Devido à especificação de uma interface padronizada e precisa o programador pode escrever módulos clientes e módulos servidores que podem ser usados em diferentes computadores e sistemas operacionais com pequenas modificações e pouca recodificação.

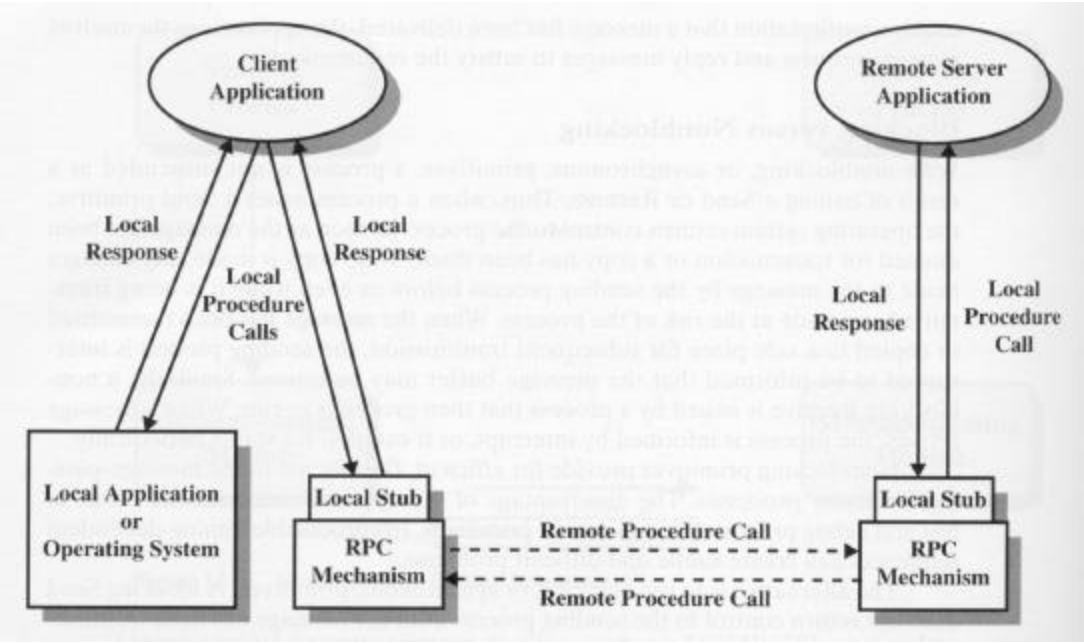

Figura 19 Mecanismo de chamada remota a procedimento

Na Figura 19 temos uma visão detalhada do mecanismo de RPC. O programa que chama faz uma chamada comum de procedimentos com parâmetros em sua própria máquina. Por exemplo a chamada CALL P(X,Y) onde P = Nome do procedimento, X = Parâmetro enviado e Y = Parâmetro recebido

Isto pode ou não ser transparente ao usuário que a intenção é chamar um procedimento remoto em outra máquina. Um pequeno procedimento, chamado de STUB, P deverá ser incluído no procedimento que chama ou ser ligado dinamicamente na hora da chamada. Este STUB cria uma mensagem que identifica o procedimento sendo chamado e inclui os parâmetros. Envia a mensagem ao sistema remoto e aguarda a resposta. Quando uma resposta é recebida este STUB P retorna ao programa chamador passando os valores que retornaram.

Na máquina remota, um outro STUB é associado com o procedimento chamado. Quando uma mensagem chega, ela é examinada e uma chamada local CALL P(X,Y) é gerada. Este procedimento remoto é assim chamado localmente e tratado como uma chamada local.

São características básicas que podemos observar no mecanismo RPC:

- Passagem de parâmetros – Geralmente por valor e não por referência pois os dados dos parâmetros são copiados dentro da própria mensagem enviada.
- Representação dos parâmetros – Se temos ambientes heterogêneos podemos ter diferença na representação de números e até de texto (tipos inteiros, tipos float, ASCII/EBCDIC).
- Forma de ligação – A forma de ligação entre o cliente e o servidor pode ser de forma persistente e não persistente. Dizemos persistente quando é estabelecido a comunicação na primeira chamada e esta ligação permanece após a primeira resposta. De outra forma quando a cada mensagem é estabelecido uma comunicação, enviada a mensagem, recebido a resposta e desfeita a comunicação, chamamos de ligação não persistente.
- Sincronismo – O tradicional é a chamada de procedimentos de forma síncrona, ou seja o procedimento que chamou, passa a mensagem e fica aguardando a resposta do procedimento chamado. Para prover flexibilidade, várias facilidades de chamadas de procedimentos assíncronas foram implementadas visando um maior grau de paralelismo enquanto mantêm a simplicidade das chamadas de procedimentos remotos.

Segundo Tanenbaum (1992) o objetivo da chamada remota de procedimentos é manter escondido do usuário todos os procedimentos relativos à comunicação remota. Algumas dificuldades se apresentam quando temos a ocorrência de erros no servidor, no cliente ou na rede gerando situações como segue:

- **O cliente não é capaz de localizar o servidor**: Devido a uma falha de hardware e o servidor efetivamente não estar disponível.
- **Perda de mensagem solicitando serviço**: Usando um contador de tempo, se o tempo expira e a resposta não veio, retransmite a mensagem.
- **Perda de mensagem com resposta:** Uma solução obvia é um temporizador e se depois de determinado tempo o cliente não obtém a

resposta, faz a requisição novamente. O problema é que nem sempre podemos simplesmente pedir para repetir a operação. Temos situações onde a requisição pode ser feita repetidas vezes e teremos sempre o mesmo resultado. Este tipo de operação é chamado **idempotente**. Outra situação é se repetimos a solicitação e os dados são alterados no servidor a cada chamada refeita. Exemplo, adicionar um valor ao saldo de uma conta corrente. Nesta situação temos uma operação **não idempotente**.

- **Quedas do servidor:** Este tipo de problema também está relacionado com a operação idempotente mas não pode ser resolvida com o uso de números seqüenciais nas mensagens. Num servidor a ordem normal dos eventos é receber a solicitação, executar o procedimento e responder. Assim uma parada no servidor pode acontecer após a execução ou antes da execução do procedimento. Existe a técnica de chamada de no mínimo uma vez, e garante que a chamada remota ao procedimento será executada no mínimo uma vez. Outro método é chamado de semântica de no máximo uma vez e garante que a chamada remota foi executada no máximo uma única vez.
- **Queda do cliente:** É a situação quando o cliente manda uma requisição e antes de receber a resposta, sai do ar.

Na chamada remota a procedimentos o protocolo empregado é muito importante. Em princípio qualquer protocolo que permita enviar os bits do cliente para o servidor e vice-versa, poderia ser usado, porém o aspecto de performance é muito importante e está diretamente relacionado com o protocolo adotado. Podemos usar protocolo orientado a conexão ou protocolo não orientado a conexão, que não necessitam estarem sempre interligados para a troca de mensagens. Quando usamos um protocolo orientado a conexão, antes de enviar a primeira mensagem entre dois procedimentos, é estabelecida uma conexão a qual ficará ativa a partir deste ponto. Assim facilita muito o processo de retransmissão, pois o tratamento dos erros será feito pelo protocolo em um nível abaixo e não pela aplicação que troca mensagens. Em especial em ligações de longa distância é usado um protocolo orientado a comunicação. Entretanto para redes locais o nível de erros é muito baixo e o overhead devido ao uso de um protocolo orientado a conexão é muito alto. Neste caso um protocolo sem conexão será empregado na maioria das vezes. Outra consideração é se devemos usar um protocolo padrão, de propósito geral, ou devemos desenvolver um específico para chamada remota. Alguns sistemas distribuídos usam o IP, ou o UDP que é construído em cima do

IP (TCP/IP), como protocolo básico. A decisão de implementação a partir de um protocolo padrão como TCP/IP é devido a facilidades existentes tais como: já estar desenvolvido e testado; padrão em todos sistemas UNIX; disponível em diferentes sistemas operacionais e adequado a redes locais e redes remotas.

Com relação à performance, o IP não foi projetado para ser um protocolo para usuário final. Seu projeto baseia-se no fato de que é possível estabelecer conexões TCP confiáveis entre redes.

Outro aspecto muito importante a ser considerado é o fluxo de controle. Muitos chips de interface de rede são capazes de enviar pacotes consecutivamente sem quase nenhum intervalo entre eles mas estes mesmos chips normalmente não têm como receber um número ilimitado de pacotes, devido à sua capacidade de armazenamento ser finita.

Com relação a confirmações de mensagens temos a situação do servidor eventualmente perder a confirmação de uma mensagem. Na chamada remota de procedimentos o protocolo consiste em uma requisição, uma resposta e uma confirmação. Esta última é necessária para fazer com que o servidor possa desfazer-se das respostas que tenham sido recebidas corretamente pelo receptor. Na prática o servidor pode inicializar um temporizador ao enviar a resposta e livrar-se dela quando este temporizador expirar. Outra forma é interpretar uma nova requisição do cliente, como sendo também uma confirmação da requisição anterior, pois se a anterior não fosse recebida, não estaria fazendo uma nova requisição.

Alguns aspectos sobre gerencia de tempo devem ser considerados na chamada remota de procedimentos. Todos os protocolos são desenvolvidos para permitir a troca de mensagens sobre algum tipo de meio de comunicação. Em todos os sistemas, algumas mensagens podem ser ocasionalmente perdidas, devido ao ruído na comunicação ou ao overflow no buffer de recepção. Em conseqüência, a maioria dos protocolos inicializa um temporizador sempre que uma mensagem é expedida e uma resposta é esperada. Se a resposta não chegar em determinado tempo, o temporizador sinaliza, e a mensagem original é retransmitida.

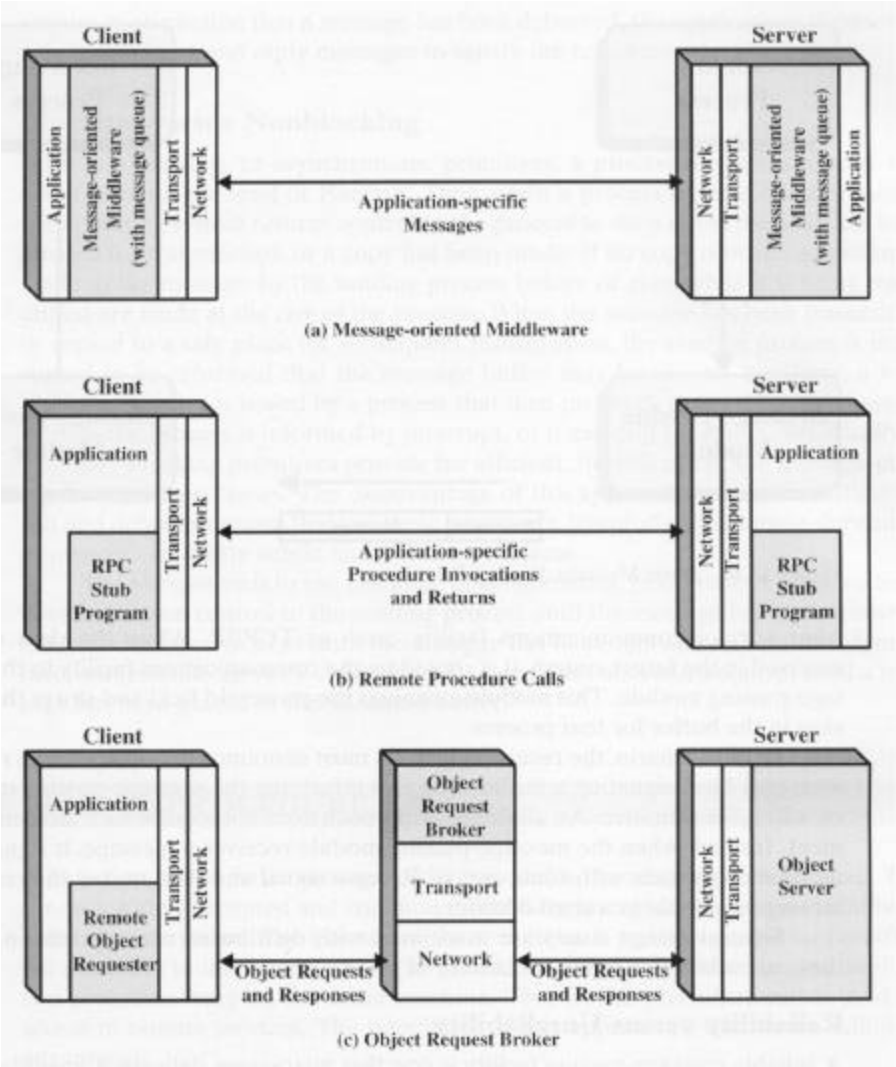

Figura 20 Servidor de requisição de Objetos

Em um nível superior a chamada de procedimentos remotos, encontramos o mecanismo de comunicação de objetos. Nos mecanismos de comunicação com orientação a objetos podemos ter simples mensagens ou objetos completos sendo trocados entre os processos que se comunicam. Segundo Stallings (1998) um cliente que necessita um serviço envia uma requisição para um provedor de requisição de objetos que atua como um diretório de todos os serviços remotos disponíveis na rede, Figura 20. Uma padronização neste mecanismo de troca de objetos ainda não ocorreu. Encontramos concorrentes como Microsoft com COM/OLE (Common Object Model / Object Linking ans Embedding) e outros como IBM. Apple, Sun que adotaram CORBA (Common Object Request Broker Architecture).

3 COMUNICAÇÃO EM GRUPO

Destacamos neste capítulo o embasamento conceitual relativo ao tema principal deste trabalho que é a Comunicação em Grupo. Todas as características deste paradigma usado em sistemas distribuídos se encontram conceituadas neste capitulo.

3.1 Introdução

Em sistemas distribuídos é fundamental a comunicação entre os diversos computadores que compõem este ambiente de sistemas distribuídos. Como já vimos anteriormente a Chamada Remota de Procedimentos – RPC se aplica em especial para a comunicação que envolve dois processos. Em algumas situações é desejável que um processo possa se comunicar com diversos outros processos. Com RPC não podemos ter um único transmissor que envia de uma única vez para múltiplos receptores. Este mecanismo de comunicação que trata múltiplas conexões é chamado de Comunicação em Grupo e segundo Tanenbaum (1992) tem as peculiaridades e características que serão apresentadas neste capítulo. Um grupo é uma coleção de processos que interagem entre si em algum sistema. A propriedade chave que todos os grupos tem é que quando uma mensagem é enviada para o grupo, todos os membros do grupo recebem esta mensagem. Esta forma é chamada Comunicação Um-Para-Muitos, um transmissor para muitos receptores, em contraste com a Comunicação Ponto-a-Ponto, conforme ilustrado na Figura 21.

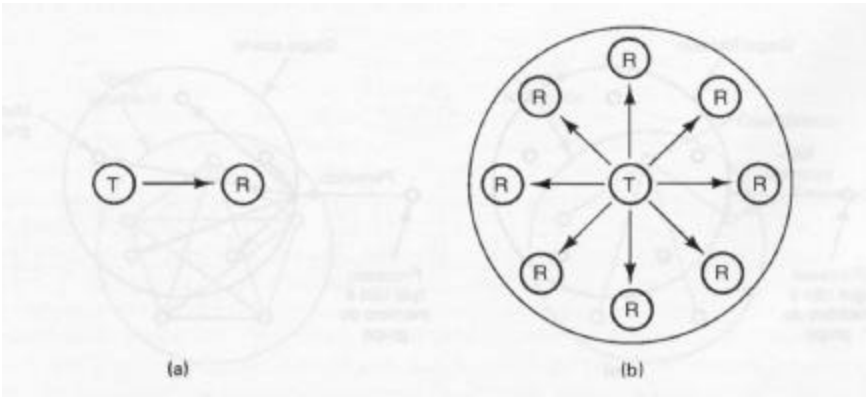

Figura 21 (a) Comunicação Ponto-a-Ponto (b) Comunicação Um-Para-Muitos

Grupos são dinâmicos e assim novos grupos podem ser criados e grupos antigos podem ser destruídos. Um processo pode juntar-se a um grupo ou deixar um grupo. Um

processo pode ser membro de diversos grupos ao mesmo tempo. Assim mecanismos são necessários para gerenciar grupos e os processos participantes dos grupos. Quando usamos mecanismos de comunicação em grupo os processos tratam com coleções de outros processos de forma abstrata. Se um processo envia uma mensagem para um grupo de servidores, este processo não precisa se preocupar quantos são os servidores nem onde eles estão localizados. Inclusive já na chamada seguinte o numero de servidores deste grupo e suas localizações podem mudar sendo esta alteração transparente para o processo que envia a mensagem para o grupo.

Como será implementada a comunicação em grupo, depende das características do hardware usado na interligação dos equipamentos. Em algumas redes é possível se criar um endereço especial de rede onde múltiplas máquinas podem escutar neste endereço. Quando um pacote é enviado para um destes endereços, automaticamente ele é recebido por todas as máquinas que estão escutando este endereço. Esta técnica é chamada de "*multicasting*". Implementar comunicação de grupos usando "*multicasting*" é simples, basta atribuirmos um endereço de "*multicasting*" para cada grupo. Em redes que não temos o "*multicasting*", ainda podemos ter o "*broadcasting*", onde os pacotes que tem um determinado endereço são escutados por todas as máquinas. O "*broadcasting*" pode ser usado para implementar comunicação em grupo, embora não seja tão eficiente, pois todos na rede vão receber as mensagens e cada processo terá que avaliar se aquela mensagem recebida é para o grupo ao qual ele pertence. Se não for vai descartar a mensagem mas neste caso o processo já gastou tempo e recursos. Mesmo assim ainda tem a vantagem que um único pacote enviado vai atingir a todos os equipamentos da rede. Se a rede não suporta "*multicasting*" nem "*broadcasting*" ainda é possível se implementar um sistema onde o transmissor vai enviar um pacote separado para cada um dos receptores do grupo. Para um grupo com N membros é necessário o envio de N pacotes ao invés de um único pacote como no "*broadcasting*" ou "multicasting". Quando a comunicação é um a um é chamada de "*unicasting*". Além das características normais encontradas nos mecanismos de troca de mensagens, tais como, buferização, blocagem, no tratamento de grupos temos novas características de organização, endereçamento e outras que serão detalhadas na seqüência.

3.2 Organização dos Grupos

Os sistemas que suportam comunicação em grupo podem ser divididos em duas categorias, dependendo que quem está enviando para quem. Alguns sistemas suportam o conceito de grupo fechado, no qual somente membros do grupo podem enviar mensagens para o grupo. Membros de fora do grupo não podem enviar mensagens para o grupo como um todo, embora possam enviar mensagens para membros individuais. Outro conceito, grupo aberto, onde qualquer processo pode enviar mensagem para qualquer grupo. Na Figura 22 temos uma ilustração de grupo aberto e grupo fechado.

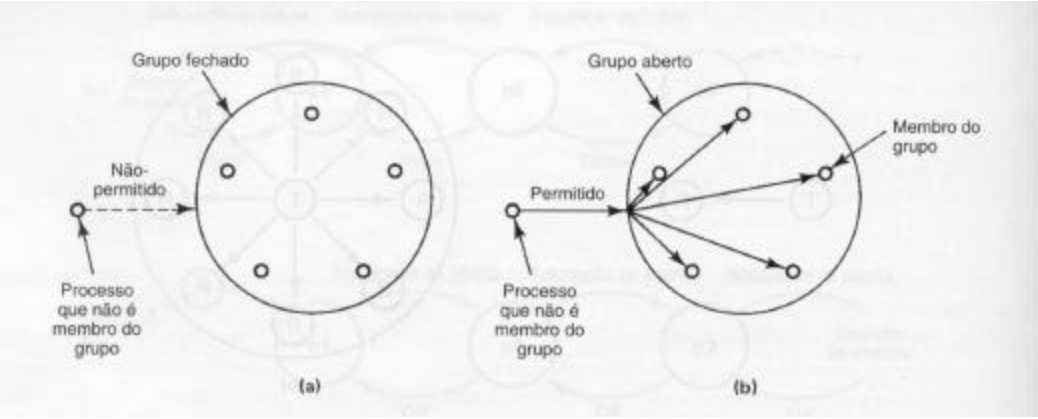

Figura 22 (a) Grupo Fechado (b) Grupo Aberto

A decisão de usar grupo fechado ou grupo aberto, normalmente está relacionada com o tipo de aplicação que estará sendo executada, usando estes mecanismos de comunicação em grupo. Se for uma aplicação típica de processamento paralelo, uma coleção de processos trabalhando juntos para a solução de um problema, neste caso não interessa outro processo de fora interagir com algum processo do grupo, sendo típico para um grupo fechado. De outro lado, em uma aplicação de servidores replicados, um processo não membro do grupo, um cliente, pode enviar uma mensagem para o grupo fazendo alguma requisição aos servidores.

Ainda quanto à organização temos grupos de semelhantes e grupos hierárquicos. Esta característica está relacionada à estrutura interna do grupo. Em alguns grupos os processos são iguais. Nenhum processo é gerente e todas as decisões são feitas coletivamente. Em outros grupos existe um processo que é o coordenador e os demais fazem o que é determinado por este coordenador. Neste caso quando uma requisição é feita, seja por um cliente externo ou por um dos membros do grupo, a requisição é enviada ao coordenador. O coordenador decide qual dos processos do grupo é o mais

indicado para executar esta solicitação. Este padrão de comunicação está ilustrado na Figura 23.

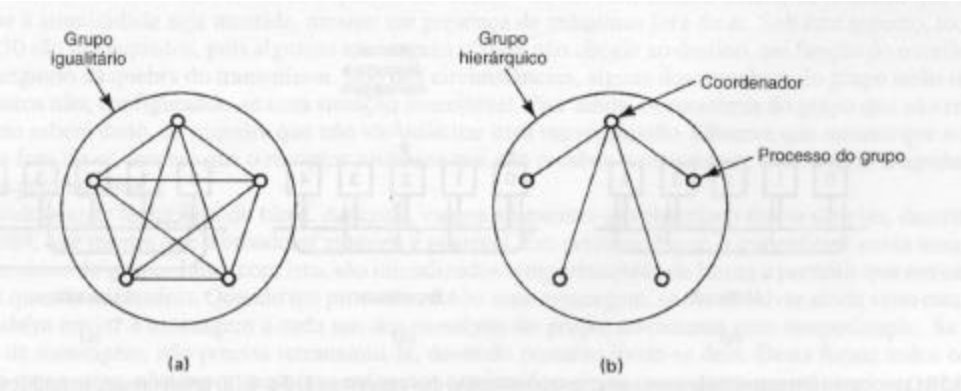

Figura 23 (a) Comunicação em Grupo de Semelhantes
(b) Comunicação em Grupo Hierárquico.

Cada uma destas organizações tem suas próprias vantagens e desvantagens. Na comunicação em grupos de semelhantes não tem um ponto único de falha. Se um dos processos falhar, o grupo continua, apenas se torna menor. A desvantagem é que a tomada de decisão é mais complicada. Para alguma decisão, tem que envolver todos do grupo, gerando uma demora e um overhead. No grupo hierárquico temos o oposto sendo as decisões tomadas pelo coordenador. Qual processo do grupo vai executar qual atividade e quando um processo termina, ele avisa ao coordenador que está pronto para outra atividade. A perda do coordenador em um grupo hierárquico causa uma parada nas atividades do grupo.

3.3 Gerenciamento na Comunicação em Grupo

No mecanismo de comunicação em grupo é necessário um controle sobre a criação e destruição de grupos, bem como permitir que processos se integrem ao grupo ou deixem o grupo. Um enfoque é ter um servidor de grupo para o qual todas as requisições possam ser feitas. Este servidor de grupo pode manter um controle total de todos os seus grupos e quais participantes de grupo pertencem a qual grupo. Este método é fácil de entender, eficiente e de fácil implementação. Infelizmente ele compartilha com a maior desvantagem das técnicas centralizadas, um único ponto de

falha. Se o servidor de grupos falhar, o gerenciamento do grupo deixa de existir. Provavelmente a maioria dos grupos terá que ser reconstruídos, e certamente interrompendo o trabalho que estava em andamento.

O enfoque oposto é gerenciar os grupos de uma forma distribuída. Em um grupo aberto, um processo de fora do grupo, pode enviar uma mensagem para todos do grupo anunciando a sua presença. Em um grupo fechado, ao menos com relação à solicitação de participar do grupo, também será enviada uma mensagem para todos. Este grupo fechado, só aceita uma mensagem de algum processo fora do grupo, quando esta mensagem é de solicitação para participar do grupo. Uma mensagem direta sem antes estar participando do grupo fechado, não seria aceita. Para um processo deixar de fazer parte de um grupo, basta enviar uma mensagem de adeus a todos do grupo.

Temos ainda duas características relativas ao controle dos participantes do grupo. Se o membro do grupo falhar ele efetivamente deixa de pertencer ao grupo. O problema é que pode não existir um aviso a todos os outros membros do grupo, como quando o processo deixa o grupo de forma padrão. Os demais membros têm que descobrir por conta própria que aquele participante não está respondendo e o excluir do grupo. Outro ponto é o sincronismo necessário entre as mensagens sendo enviadas e a entrada ou saída de um membro no grupo. Todo processo que for integrado ao grupo no momento t deverá receber a partir deste momento t, todas as mensagens enviadas. Da mesma forma quando deixa o grupo no momento z, todas as mensagens enviadas após z já não são mais endereçadas a este processo que deixou o grupo. Uma forma de resolver esta situação é transformar a operação de inclusão no grupo ou retirada do grupo em uma mensagem especial de inclusão ou retirada que será transmitida de forma síncrona na seqüência das demais mensagens para o grupo. Uma úlltima preocupação relacionada a controle dos participantes do grupo é quando muitas máquinas param de forma anormal, por exemplo quando ocorre um particionamento da rede, deixando o grupo inoperante. Algum protocolo é necessário para reconstruir o grupo com as máquinas que restaram.

Para que um processo envie uma mensagem a um grupo é necessária uma forma de identificar o grupo. Uma forma é atribuindo a cada grupo um endereço único assim como temos um numero único de identificação para cada processo. Se a rede suportar *multicast*, o endereço do grupo pode ser associado com o endereço de *multicast*, assim cada mensagem enviada ao grupo pode ser direcionada especificamente para aquele grupo identificado pelo endereço de *multicast* e só vai ser recebida pelas máquinas que fazem parte daquele endereço de *multicast*. Se o hardware não suporta *multicast* mas

aceita *broadcast*, as mensagens podem ser enviadas por *broadcast*. Cada kernel receberá a mensagem e vai avaliar o endereço do grupo. Se algum processo que está rodando nesta máquina faz parte do grupo ele recebe a mensagem, caso contrario, a mensagem será descartada. Em ultimo caso se nem *multicast* nem *broadcast* for suportado na rede, o kernel na máquina que envia deverá ter uma lista de todas as máquinas que tem processos pertencentes a este grupo da mensagem a ser enviada. O kernel então envia em uma conexão ponto-a-ponto a cada um dos participantes do grupo uma de cada vez, a mensagem a ser transmitida. Estas três implementações são vistas na Figura 24 onde um processo 0 envia uma mensagem para um grupo consistindo dos processos 1, 3 e 4.. Um ponto importante é que mesmo nos três casos o processo que envia manda apenas uma mensagem para o endereço do grupo e a mensagem vai para todos os membros do grupo. A forma como será transmitida é problema do sistema operacional. O processo que envia não tem que se preocupar com o tamanho do grupo ou se a comunicação vai ser por *multicasting*, *broadcasting* ou *unicasting*.

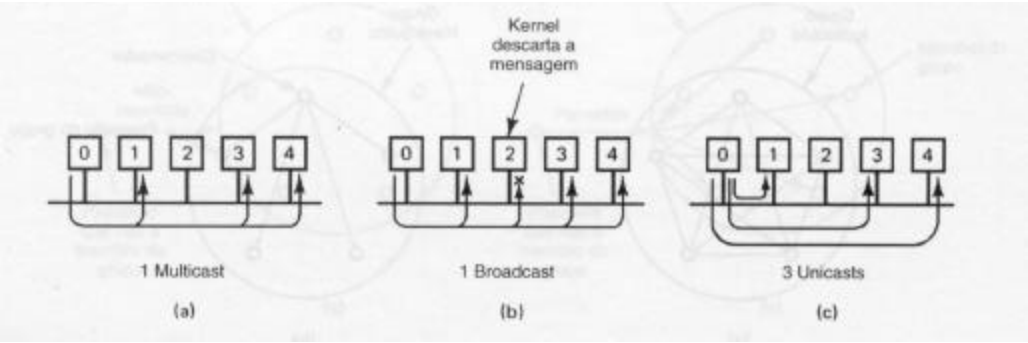

Figura 24 (a) *Multicasting* (b) *Broadcasting* (c) *Unicasting*

Um segundo método de endereçamento do grupo é quando o processo do usuário final que envia tem uma lista explicita de todos os destinos, por exemplo, endereços IP. Quando este método é usado, um parâmetro na chamada de envio da mensagem é um ponteiro para uma lista de endereços. Este método tem um sério inconveniente que é forçar o processo membro do grupo de saber precisamente quem faz parte do grupo. Assim deixa de ser transparente. Sempre que existir uma mudança de participante no grupo o processo do usuário final precisa atualizar sua lista de membros do grupo. No exemplo da Figura 24 esta administração é feita pelo kernel e o processo do usuário final não se preocupa com isto. Um terceiro método de endereçamento é chamado de endereçamento com predicado. Neste sistema cada mensagem é enviada para todos os

membros do grupo porém contendo um predicado, uma expressão booleana, para ser avaliada. Este predicado pode envolver o numero da máquina receptora, suas variáveis locai ou outros fatores. Se este predicado após ser avaliado for verdade então a mensagem é aceita.

Seria ideal se pudéssemos ter um conjunto de primitivas comuns, tanto para comunicação ponto-a-ponto como para comunicação em grupo. Quando usamos RPC temos o envio e a recepção de mensagens usando duas simples primitivas tais como "send" e "receive", respectivamente. Para a comunicação em grupo uma mensagem enviada pode gerar diversas respostas. Se usarmos uma biblioteca de procedimentos poderíamos ter uma semântica única para envio de mensagens, com dois parâmetros. Se fosse RPC um parâmetro era endereço do processo e o segundo endereço da mensagem. Se for para grupo o primeiro parâmetro é o endereço do grupo e segundo endereço da mensagem. A chamada pode ser usando buffer ou não, bloqueada ou não, confiável ou não tanto para ponto-a-ponto como para grupo. Em geral estas escolhas são feitas pelos projetistas do sistema e ficam sem flexibilidade de parametrização. De maneira similar o "receive" indica a intenção de receber uma mensagem, podendo o processo ficar bloqueado até o recebimento da mensagem. Se usarmos a mesma semântica o "receive" deveria funcionar para comunicação ponto-a-ponto e para grupos. Devido a estas dificuldades para manter a mesma semântica com formas tão diferentes de processamento, alguns projetistas incluíram novas primitivas como "group_send" e "group_receive". Se necessário associar a resposta a uma chamada podemos usar "getreply" onde após o envio de uma mensagem podemos chamar "getreplay" repetidamente para coletar todas as respostas associadas.

3.4 Propriedades da Comunicação em Grupos

Uma propriedade importante no mecanismo de comunicação em grupo, é a propriedade do tudo-ou-nada. A maioria dos sistemas que implementam os mecanismos de comunicação em grupo, são projetados para que as mensagens enviadas a um grupo cheguem corretamente a todos os membros deste grupo ou não cheguem a nenhum deles. Esta propriedade do tudo-ou-nada é também conhecida como atomicidade ou *broadcast* atômico. A atomicidade é importante pois torna mais fácil a programação de um sistema distribuído. Quando qualquer processo enviar uma mensagem para um

grupo ele não deve preocupar-se se algum outro membro do grupo não recebeu esta mensagem. Com esta propriedade da atomicidade o processo que envia a mensagem para um grupo irá receber uma mensagem de erro se um ou mais integrantes do grupo teve problema no recebimento sendo que os demais vão ignorar esta mensagem. Do ponto de vista de programação, tudo se passa como se fosse enviado para apenas um receptor.

Outra propriedade importante é o ordenamento das mensagens. Alem do conceito tradicional de envio e recebimento de mensagens, quando tratamos de comunicação em grupo devemos ter em mente o conceito de entrega da mensagem. Assim uma mensagem recebida para um destino final passa por um processo de entrega que eventualmente precisa ser controlado para que a seqüência de entrega seja a mesma seqüência de envio. Alguns algoritmos que tratam estas características de ordenamento em ambiente de multicast podem ser encontrado em Coulouris, 2001 no capitulo 11.

Diretório Mensagens do Usuário U3

Seq	Usuário	Assunto
1	U1	Novo modelo?
2	U2	Re: Novo modelo ?
3	U5	Outro assunto

Diretório Mensagens do Usuário U4

Seq	Usuário	Assunto
1	U2	Re: Novo modelo ?
2	U5	Outro assunto
3	U1	Novo modelo?

Figura 25 Diretório de mensagens dos usuários U3 e U4

Para exemplificar esta propriedade de ordenamento de mensagens, vamos considerar uma situação que ocorre em um grupo de assinantes de uma lista de discussão na internet. Uma lista de assinantes pode ser considerada como um grupo onde todos os membros do grupo estão interessados no assunto que é tratado na lista. Assim quando alguém envia uma mensagem para o grupo todos recebem esta mensagem e alguns podem responder a referida mensagem, resposta esta que também será propagada para todos os membros do grupo. Vamos observar o diretório de

mensagens recebidas de dois usuários deste grupo, usuário U3 e usuário U4, conforme Figura 25. O usuário U3 recebe uma mensagem com o assunto “novo modelo?” enviada por U1 e logo em seguida uma outra mensagem “Re: novo modelo?” enviada por U2 onde uma resposta para a pergunta anterior estava sendo fornecida. Se observarmos o diretório de mensagens do usuário U4 veremos que este primeiro recebeu a resposta “Re: novo modelo?” , uma outra mensagem enviada por U5 e depois é que recebeu a pergunta “novo modelo?” caracterizando uma troca de ordem nas mensagens recebidas. Esta troca ocorreu pois o servidor de mensagens trabalha de forma assíncrona e a entrega das mensagens não tem nenhuma garantia de ordenamento. Certamente a mensagem enviada por U1 com a pergunta “novo modelo?” não pode ser entregue na primeira tentativa e quando foi reenviada posteriormente para o usuário U4 este já tinha recebido antes a resposta da pergunta.

Um mecanismo de comunicação em grupo precisa ter uma semântica bem definida com relação à ordem em que as mensagens serão entregues. A melhor garantia para isto é fazer com que as mensagens sejam entregues na mesma ordem em que foram enviadas. No nosso exemplo de lista de mensagens anterior esta garantia não é implementada pois o custo seria muito grande e os sistemas de lista se valem dos recursos já existentes de envio de mensagens via internet e que não contemplam ordenamento.

A ordenação de mensagens é um ponto muito importante na comunicação em grupo e assim vamos detalhar mais este assunto baseado nos conceitos de ordenação de mensagens em serviços de *broadcast* (ATTIYA,1998) (MULLENDER, 1993) e também no sistema AMOEBA (KAASHOEK, 1993). Temos quatro tipos diferentes de ordenação que normalmente estão implementadas nos mecanismos de comunicação de grupo:

- **Sem Ordem** – Onde as mensagens são enviadas ao grupo sem a preocupação de ordenamento. Tem o menor overhead pois não necessita controle algum de seqüência porém pode não ser adequado para muitas aplicações.
- **Ordenamento FIFO** – Garante que todas as mensagens de um membro sejam entregues aos demais membros do grupo na ordem em que elas foram enviadas. Para todas as mensagens M1 e M2 e todos os processos Pi e Pj, se Pi envia M1 antes de enviar M2, então M2 não é recebida em Pj antes que M1 seja recebida.

- **Ordenamento CAUSAL** – No ordenamento causal introduzimos o conceito de mensagem dependente de outra. Um exemplo para ilustrar, é quando recebemos mensagens de uma lista na internet. Um membro da lista envia uma pergunta para a lista, esta mensagem é enviada a todos os membros da lista. Na seqüência alguém responde a solicitação e esta resposta também é enviada a todos. Na caixa postal de um participante da lista chega primeiro a mensagem com a resposta e mais tarde a mensagem com a pergunta. No ordenamento causal as mensagens estão em ordenamento FIFO e se um membro após receber a mensagem A envia uma mensagem B, é garantido a todos os membros do grupo que vão receber A antes de B. Para todas as mensagens M1 e M2 e cada processo Pi, se M1 acontece antes de M2 então M2 não é recebida em Pi antes que M1 seja. A Figura 26 apresenta um exemplo gráfico de ordenamento causal. Nesta execução duas mensagens são controladas em termos de entrega. Primeiro a mensagem de p1 (0,2,0) é atrasada até que a mensagem anterior (0,1,0) chegue vinda de p1. Depois a mensagem (1,3,0) vinda de p0 é atrasada até que a mensagem (0,3,0) vinda de p1 (que aconteceu antes) tenha chegado.

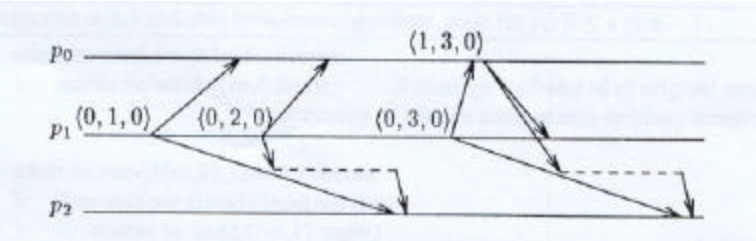

Figura 26 Ordenamento Causal

- **Ordenamento Total** – No ordenamento total cada membro do grupo recebe todas as mensagens na mesma ordem. Para todas as mensagens M1 e M2 e todos os processos Pi e Pj, se M1 é recebida em Pi antes que M2 seja, então M2 não é recebida em Pj antes que M1 seja. Este ordenamento total é o melhor de todos aqui apresentados e torna a programação fácil, mas é difícil de ser implementado.

Uma propriedade de sobreposição de Grupos está associada ao fato que um processo pode ser membro de diversos grupos ao mesmo tempo. Esta situação pode levar a que tenhamos inconsistências. Podemos observar na Figura 27 que mostra dois grupos 1 e 2. Os processo A, B e C são membros do grupo 1. Os processo B,C e D são membros do grupo 2. Supondo que os processos A e D decidam simultaneamente enviar uma mensagem a seus respectivos grupos, e que o sistema use a ordenação em tempo global dentro de cada grupo. Considerando que estamos usando *unicast*, a ordem das mensagens é mostrada na Figura 27, através da numeração de 1 a 4. Novamente temos uma situação em que dois processos B e C, recebem mensagens em ordem diferente. O processo B primeiro recebe uma mensagem de A, seguida por uma vinda de D. O processo C recebe na ordem oposta. O problema é que embora tenhamos ordenação em tempo global, dentro do grupo, não há necessariamente qualquer coordenação entre os diversos grupos. Alguns sistemas suportam uma ordenação no tempo muito bem definida entre grupos sobrepostos, mas outros não.

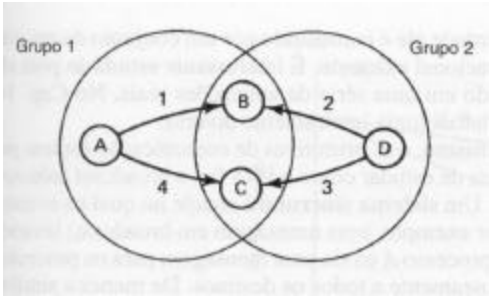

Figura 27 Quatro processos A,B,C e D e quatro mensagens.
Os processos B e C recebem as mensagens de A e D em ordem diferente

Outra propriedade importante nos mecanismos de comunicação em grupo se refere a escalabilidade. Muitos algoritmos funcionam bem enquanto os grupos possuem poucos elementos e não houver muitos grupos. Podemos ter problemas se o grupo tiver centenas ou milhares de componentes, ou se tivermos milhares de grupos e se o sistema for muito grande.

10 REFERÊNCIAS BIBLIOGRÁFICAS [1]

AMIR, Y. **Replication using Group Communication over a Partitioned Network** Ph.D. Thesis., Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, 1995.

AMIR, Y.; STANTON, J. **The SPREAD Wide Area Group Communication System.** Technical report, Center of Networking and Distributed Systems, Johns Hopkins University, Baltimore, Mariland, 1998. Disponível na internet URL: http://www.cnds.jhu.edu/publications/ capturado em agosto de 2001.

ATTIYA H.; WELCH J, **Distributed Computing: Fundamentals, Simulations and Advanced Topics**, McGraw-Hill, 1998

BERKET, K., **The InterGroup Protocols: Scalable Group Communication for the Internet.** Dissertation, University of California – USA 2000. Disponível na internet URL http://www-itg.lbl.gov/InterGroup capturado em dezembro de 2001.

BIRMAN, K; RENESSE, R. van, **Reliable Distributed Computing with the ISIS Toolkit**, IEEE Computer Society Press, 1994.

BIRMAN, K **Pagina do professor BIRMAN**, em http://www.cs.cornell.edu/Info/Department/Annual95/Faculty/Birman.html e http://www.cs.cornell.edu/Info/Projects/ISIS/ acessada em março de 2002.

BUYYA, R., **High Performance Cluster Computing: Architecture and Systems**, Vol 1. Pags 3-45, 1999

COULOURIS G.; DOLLIMORE J.; KINDBERG T., **Distributed Systems Concepts and Design**, Addison Wesley, Third Edition, 2001

[1] Seguiu-se a NBR 6023/2000 que substitui a NBR 6023/1989. Nas citações seguiu-se NBR 10520/2001 que substitui a NBR 10520/1992.

DIETZ, Hank **Linux Parallel Processing HOWTO**, 1998 [online]. Disponível na internet URL http://yara.ecn.purdue.edu/~pplinux/PPHOWTO/pphowto.html capturado em outubro de 2001.

DOLEV, D.; MALKI, D.; **The Transis Approach to High Availability Cluster Communication.,** Communications of the ACM, 39(4), April 1996

EMMERICH W., **Engineering Distributed Objects**, John Wiley & Sons Ltd, 1999

EZHILCHELVAN, P.; MACEDO, R.; SHRIVASTAVA, A. **NEWTOP: A Fault-tolerante Group Communication Protocol**. Proceedings of the 15th IEEE International Conference on Distributed Computing Systems, pag 296-306, Vancouver, Canada, maio 1995.

GALLI, D.L., **Distributed Operating Systems Concepts and Practice**, Prentice Hall, 1998

HAYDEN, M. **The Ensenble System,** PhD thesis, Department of Computer Science, Cornell University, January 1998

KAASHOEK, M.F.; TANENBAUM, A.S. **Group Communication in AMOEBA and its applications**, Distributed Systems Engineering Journal, vol 1, pp 48-58, julho1993[online]. Disponível na Internet URL: http://www.cs.vu.nl/vakgroepen/cs/amoeba_papers.html capturado em janeiro de 2002.

KOCH, R.R. **The Atomic Group Protocols: Reliable Ordered message delivery for ATM networks** PhD Dissertation, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, December 2000

MALLOTH, C; **Conception and Implementation of a Toolkit for Building Fault-Tolerant Distributed Applications in Large-Scale Networks,** PhD thesis , Ecole Polytechnique Federale de Lausanne, 1996.

MONTRESOR, A., **System Support for Programming Object-Oriented Dependable Applications in Partitionable Systems**, Technical Report UBLCS-2000-10 Department of Computer Science, University of Bologna, Bologna – Italy, 2000. Disponível na internet URL http://www.cs.unibo.it capturado em dezembro de 2001.

MOSER, L. E.; AMIR, Y.; MELLIAR-SMITH, P.M.; AGARWAL, D.A. **Extended Virtual Synchrony.** Proceedings of the 14th IEEE International Conference on Distributed Computing Systems, Pag 56-65, Poznan, Polonia, junho 1994. IEEE Computer Society Press, Los Alamitos, CA, 1994

MOSER, L. E.; MELLIAR-SMITH, P.M.; AGARWAL, D.A.;BUDHIA, R.;LINGLEY-PAPADOULOS, C. **Totem: A Fault-Tolerant Group Communication System.** Communications of the ACM, 39(4) , April 1996.

MULLENDER S., **Distributed Systems**, Addison-Wesley, second edition, 1993

MYRICOM **Myrinet Overview**, 2001 Disponível em http://www.myri.com/myrinet/overview/index.html Acesso em 01 fev 2002.

OMG – Object Management Group. **The Common Object Request Broker: Architecture and Specification**, *Rev 2.3* OMG Inc., Framingham, Mass., March 1998

OPEN Group **DCE Overview**, 1999. [online] Disponível na internet URL http://www.opengroup.org/dce/info/papers/tog-dce-pd-1296.htm
capturado em janeiro de 2002.

RADAJEWSKI, J., EADLINE D. **Beowulf HOWTO,** 1998. [online] Disponível na internet URL http://www.linuxvoodoo.com/howto/HOWTO/Beowulf-HOWTO/Beowulf-HOWTO-4.html capturado em outubro de 2001.

RENESSE, R. van; BIRMAN, K.P.; MAFFEIS, S. **Horus: A Flexible Group Communication System**, Communications of ACM, 39(4):76-83, April 1996.

SILBERSCHATZ, A., GALVIN, P.B., **Sistemas Operacionais Conceitos**, Quinta Edição, Prentice-Hall, 2000.

STALLINGS, W., **Operating Systems Internals and Design Principles**, Third Edition. Prentice-Hall, 1998.

SUN Microsystems. **Java Remote Method Invocation Specification**, *Rev 1.50*. 1998, Sun Microsystems, Inc., Mountain View, California, October 1998.

TANENBAUM A. S., **Modern Operating Systems**. Prentice-Hall, 1992.

VERÍSSIMO, P.; RODRIGUES, L., **Distributed Systems for System Architects**, Kluwer Academic Publishers, 2001

WHETTEN, B.; MONTGOMER, Y., T.; KAPLAN, S., **A High Performance Totally Ordered Multicast Protocol**. Proceedings of the International Workshop on Theory and Practice in Distributed Systems, pag 33-57, Dagstuhl Castle, Alemanha, Setembro 1994.