Apostila

Revisado e atualizado em Jan/2016

ÍNDICE

1. INTRODUÇÃO

A comunicação entre pessoas e organizações tem vindo a ganhar cada vez mais importância no desempenho das mais variadas atividades, quer profissionais quer de lazer. Qualquer sistema de informação se beneficia com uma rede de comunicação, nomeadamente uma rede de computadores que permita a troca de dados ou de informação no formato digital.

As redes de computadores foram desenvolvidas para dar suporte a vários tipos de aplicações, como transmissão de dados, voz e vídeo, comunicações entre terminais e computadores. Independentemente da aplicação, existem vários fatores a ter em consideração como a dispersão geográfica, o tipo de informação transmitida e a fiabilidade exigida.

Na sociedade de informação, parte integrante dos tempos modernos, a probabilidade da ocorrência de falhas no acesso aos serviços é uma preocupação atual. Sendo assim, só é possível atingir os resultados pretendidos se existir a garantia de que os dados necessários estão disponíveis em tempo real e correspondem a informação fiável. Torna-se, por isso, necessário implementar métodos que armazenem a informação com segurança e disponibilidade. Para isso é necessário contornar as possíveis falhas de *hardware* e *software*, mantendo intacta a interação com o exterior e com o utilizador final.

O conceito de alta disponibilidade (HA – *High Availability*) visa manter o máximo de disponibilidade possível dos serviços prestados, recorrendo a *hardware* e *software* específico para o efeito. Torna-se assim possível contornar alguns tipos de falhas, desde as relacionadas com o armazenamento até à elaboração de procedimentos em caso de falhas mais graves (planos de *disaster recovery*).

Uma das formas de alcançar HA consiste em agrupar vários servidores independentes como um único sistema (*cluster*). Este tipo de sistemas permite ainda efetuar balanceamento de carga (*load balancing*), diminuindo a sobrecarga de cada servidor. No entanto, não deve ser confundido *cluster* de alta disponibilidade com *cluster* de alto desempenho, já que este tem como objetivo a execução, em paralelo, de tarefas computacionalmente complexas.

Atualmente, a maioria dos construtores de *hardware* e *software* desenvolvem soluções integradas de HA suportando aplicações críticas como transações financeiras, acesso a bases de dados e outras funções essenciais que devem estar permanentemente disponíveis. Construtores como a IBM, a HP ou a própria Microsoft disponibilizam soluções comerciais para estes fins. No entanto, estas soluções têm um custo associado e são dependentes de uma plataforma de sistema operativo proprietária. Os sistemas *OpenSource* têm reunido cada vez mais adeptos sendo de salientar o desenvolvimento nos últimos tempos de várias soluções de HA, a baixo custo, de código aberto e com uma atualização quase permanente. A sua distribuição, que inclui o código fonte do programa, é baseada na licença GNU – GPL.

2. CONCEITOS INICIAIS

A seguir serão apresentados alguns conceitos básicos sobre redes, serviços, redundância, disponibilidade, tolerância a falhas e recuperação de desastres que são de vital importância para compreendimento do conteúdo deste material. Alguns destes conceitos serão mais aprofundados ao longo do estudo, outros serão mais aprofundados em outras disciplinas pois não são o foco aqui.

2.1. Redes de computadores

Uma rede de computadores consiste num grupo de dois ou mais sistemas computacionais (computador, *software* e periféricos) interligados. As redes podem ser distinguidas através de vários fatores, entre eles a área geográfica. Às redes em que os computadores se encontram geograficamente próximos, por exemplo no mesmo edifício, designam-se por *Local Area Networks* (LANs) e implementam normalmente uma tecnologia que utiliza a técnica CSMA/CD para acesso ao meio de transmissão.

Segundo TANENBAUM (2003):

> *"...as redes de computadores são um conjunto de computadores autônomos interconectados por uma única tecnologia".*

Acontece, às vezes, ligeira confusão entre redes de computadores e sistemas distribuídos.

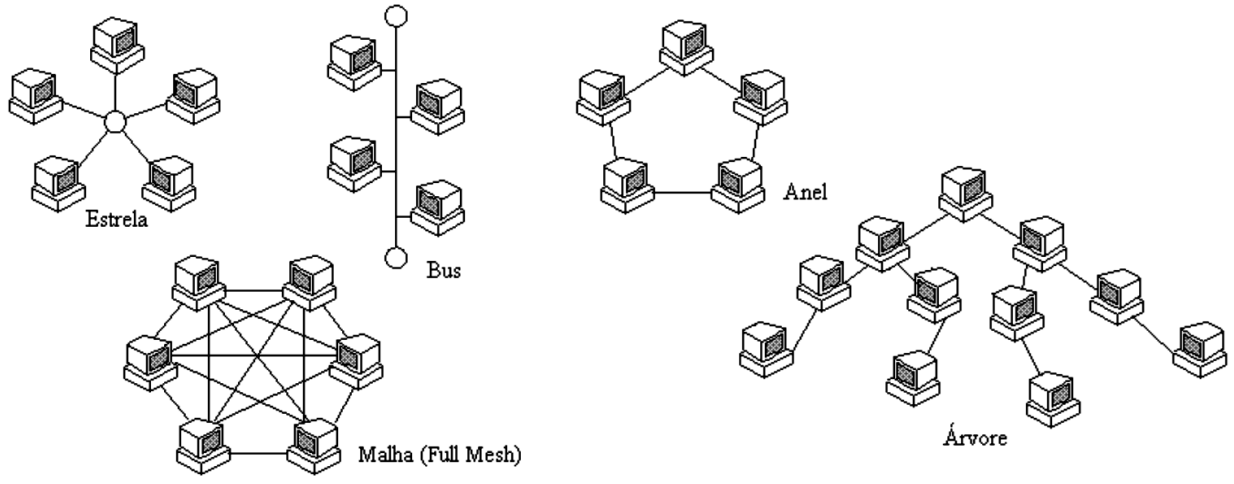
A diferença básica é que para os sistemas distribuídos a existência de vários processadores é transparente para o usuário, ou seja, o sistema oferece uma interface como se houvesse um único processador. Já para as redes de computadores a existência de vários computadores é visível para o usuário, que deve indicar explicitamente quais computadores devem participar de que tarefa.

Com relação ao uso das redes de computadores, podemos destacar ainda segundo TANENBAUM (2003) dois aspectos distintos. O primeiro está relacionado a aplicações comerciais, que se concentram no compartilhamento de recursos, e o objetivo é tornar todos os programas, equipamentos e dados ao alcance de todos os usuários a partir de políticas previamente estabelecidas, independente da distância física entre os recursos e o usuário. O segundo aspecto está relacionado ao uso das redes para aplicações domésticas, com o objetivo de acesso a informações remotas, comunicação entre pessoas, entretenimento interativo e comércio eletrônico.

Proporcionalmente à dimensão e dispersão geográfica da rede, podem ainda ser identificadas as *Metropolitan Area Networks* (MANs), que abrangem áreas semelhantes a uma cidade, ou *Wide Area Networks* (WANs), de maiores dimensões onde as comunicações são efetuadas via cabo ou ondas rádio. Uma WAN utiliza tecnologias de comutação de pacotes (por exemplo *Frame Relay*) ou de circuitos (por exemplo RDIS/ISDN). As ligações WAN são implementadas por operadores de telecomunicações ou por empresas de grandes dimensões com filiais dispersas por uma área geográfica grande.

Para desenvolver diferentes tipos de topologias, as redes são compostas por uma variedade de equipamentos que, juntamente com a cablagem, constituem a estrutura base de todo o processo de comunicação. Equipamentos simples como uma *bridge* ou um *hub* permitem estender o alcance de uma rede interligando várias estações sem influenciar os seus comportamentos. Para isolar tráfego entre segmentos e ligar diferentes tipos de redes locais são utilizados *switchs* que, além de otimizarem a largura de banda utilizável, permitem a redução de colisões na rede (domínios de colisão). Os *routers* são equipamentos usados para interligar redes distintas, definir domínios de *broadcast* e efetuar encaminhamento (*routing*) para determinada rede, utilizando protocolos de encaminhamento.

As redes informáticas também podem ser caracterizadas conforme a disposição física e/ou lógica dos elementos do sistema e a forma como são interligados. Este tipo de distinção denomina-se de topologia. Os tipos mais usuais de topologias estão exemplificados na figura a seguir.



Topologias de rede mais comuns.

Essa distinção pode ser alargada para o tipo de protocolo utilizado, que engloba um conjunto de regras que descrevem como os dados são transmitidos através da rede. Os protocolos de baixo nível definem os *standards* em termos físicos e a forma como é feita a transmissão dos dados bem como a detecção e correção de eventuais erros. A um nível superior, os protocolos tratam a formatação dos dados, incluindo a sintaxe das mensagens e a sua sequência.

2.2. Serviços

Os utilizadores de uma rede pretendem tirar partido de tudo o que ela lhe possa oferecer, direta ou indiretamente, nomeadamente ao nível da comunicação. Para que isso seja possível é necessário que as máquinas, interligadas em rede, suportem os serviços adequados. Além das funcionalidades disponibilizadas ao utilizador final (como processamento de informação ou transferência de arquivos), as aplicações (serviços) fornecem ainda a interface entre o utilizador e o sistema de comunicação. As aplicações desempenham funções como autenticação ou identificação de utilizadores ou tradução de nomes (por exemplo de máquinas) em valores numéricos perceptíveis aos diversos protocolos utilizados.

Em termos de transporte de informação, o modelo TCP/IP é o mais utilizado para interligar terminais a uma rede. Os dois protocolos de transporte usados são TCP e UDP. Enquanto que o primeiro é orientado à ligação (*connection-oriented*), permitindo alguma garantia de uma correta e ordenada recepção dos dados, o segundo é não orientado à ligação (*connection-less*), não oferecendo qualquer garantia a esse nível, sendo por isso menos robusto, mas mais leve. Isto deve-se ao fato de o conteúdo dos pacotes UDP ser mais pequeno, já que não é enviada muita informação para o destino.

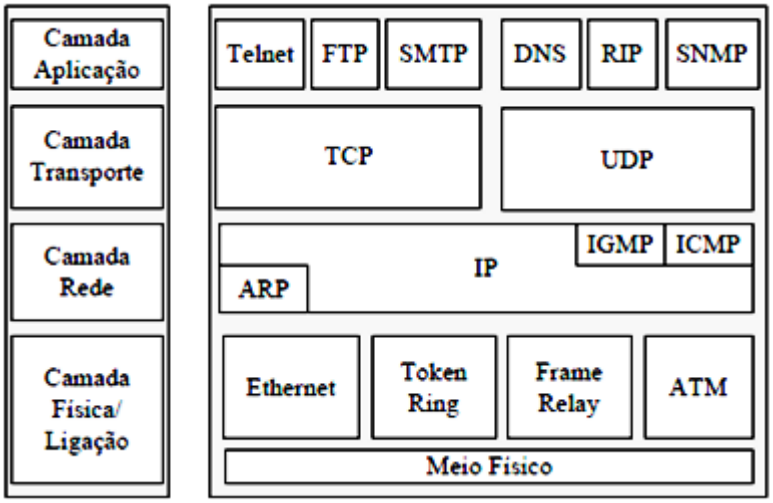
Um serviço de rede é um conjunto de operações implementado por um protocolo através de uma interface, e é oferecido à camada imediatamente superior. Ele define o que uma camada é capaz de executar sem se preocupar com a maneira pela qual as operações serão executadas.

Cada serviço é utilizado por aplicações diferentes, podendo uma aplicação utilizar vários serviços, como, por exemplo, um browser como o Mozilla Firefox. Este utiliza, por exemplo, HTTP, HTTPS, DNS.

Os serviços podem ser orientados a conexão ou não. Serviços relacionados à família TCP são orientados a conexão, enquanto serviços relacionados ao protocolo UDP são sem conexão.

- **Serviços orientados a conexão**: é o serviço TCP. Antes do envio de dados, um processo conhecido como *handshaking* cria uma conexão fraca entre os hosts. Basicamente, esse processo prepara o receptor para a recepção de pacotes. Esta conexão prévia possibilita verificar se todos os pacotes irão chegar corretamente ao destino, e em caso negativo, solicitar o reenvio dos mesmos (quando o receptor recebe um pacote, ele envia uma mensagem de confirmação ao transmissor. Se a confirmação não chegar, o pacote é reenviado), gerando uma transferência de dados confiável. Também pode fazer-se um controle de fluxo e congestionamento, para casos em que o receptor não suporta a velocidade de envio dos pacotes, ou quando algum roteador na rede está congestionado (é enviada uma mensagem ao transmissor, reduzindo ou interrompendo a velocidade de envio de pacotes). Como exemplo de serviços orientados a conexão, TCP, temos: HTTP, FTP, Telnet.
- **Serviços sem conexão**: é o serviço UDP (Protocolo de Datagrama de Usuário). Não há o processo de *handshaking*. Assim, uma aplicação apenas envia dados para um host, e com isso não há como saber se todos os pacotes chegaram. É mais rápido, mesmo por não haver a etapa da *handshaking*, mas é menos confiável, além de não possuir a possibilidade de controle de fluxo e congestionamento presentes no TCP. Algumas aplicações que usam o UDP: conferência de vídeo e telefone por internet.

Existem outros tipos de serviços, como o DHCP, que automaticamente determina um endereço IP válido a cada host conectado à Internet e o DNS, que possibilita que o utilizador use strings, ao invés de endereços IP para se conectar a outros servidores. O DNS mantém um banco de dados que relaciona cada string a um endereço IP.



Arquitetura TPC/IP

A um nível mais baixo na comunicação, é usual ter o protocolo IP sobre ligações e*thernet* em ambientes LAN. Numa WAN, uma alternativa consiste na utilização do protocolo IP sobre uma rede *Frame Relay*. Em ambos os casos podem ser usadas ligações físicas (como cabos UTP, cobre ou fibra óptica) ou ligações via rádio.

2.3. Cluster

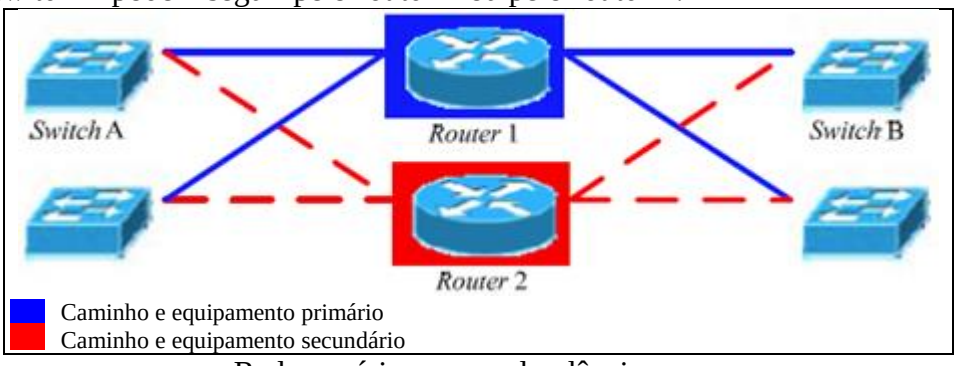
Um *cluster* é um conjunto de computadores independentes, designados de nós, que colaboram entre si para atingir um determinado objetivo comum, criando assim um sistema mais robusto. Para tal acontecer, os computadores devem comunicar entre si (através de ligações de alta velocidade) para coordenar todas as ações a serem executadas, sendo vistos pelo utilizador externo como um único sistema lógico. A comunicação é efetuada através de um mecanismo de *polling* – pedidos contínuos de envio de informação – para permitir que um servidor saiba se todos os outros nós, pertencentes ao *cluster*, estão operacionais.

A implementação de *clusters* tem como objetivos principais o aumento de disponibilidade e/ou de desempenho, quando se pretende que os serviços fornecidos estejam sempre disponíveis para novas solicitações ou que a velocidade de processamento seja aumentada recorrendo à divisão de tarefas pelos vários nós do *cluster*. Posteriormente será apresentada uma descrição mais aprofundada deste conceito que engloba os princípios, tipos e serviços de um *cluster*.

2.4. Redundância

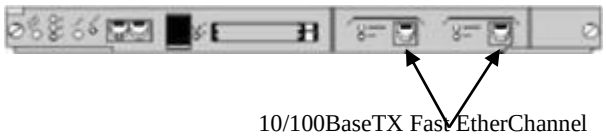
O termo redundância, associado às redes informáticas, pode significar a existência de vários métodos para efetuar uma determinada função ou de equipamento alternativo que, em caso de falha, assegura automaticamente o acesso aos serviços prestados.

Numa rede podem existir diferentes caminhos para chegar ao mesmo destino, com dois ou mais equipamentos idênticos a efetuar a mesma função. Na figura 2.3 estão representados vários caminhos possíveis para alcançar o mesmo destino: para os pacotes que passam pelo *switch* A alcançarem o *switch* B podem seguir pelo *router* 1 ou pelo *router* 2.



Rede genérica com redundância

Os equipamentos usados nas redes (como *hubs*, *switchs* e *routers*) são atualmente construídos a pensar em soluções ao nível da redundância, por exemplo equipamentos com duas fontes de alimentação. Os bastidores onde são instalados podem possuir UPS's e estabilizadores de corrente para garantir durabilidade e evitar quebras de serviço. Nas figuras 2.4 e 2.5 são visíveis dois exemplos de *hardware* redundante.



Uplinks redundantes

Routers redundantes com multiplas fontes de alimentação

2.5. Disponibilidade

A importância da disponibilidade (*availability*) de um serviço é visível em tarefas tão simples como a reserva de um lugar num avião ou o levantamento de dinheiro numa caixa Multibanco, que seriam difíceis de concretizar sem o suporte de um sistema informático. Por vezes, essa disponibilidade é tão crucial que se pode tornar a chave para a prosperidade de um negócio ou na sua total falência.

O conceito de disponibilidade numa rede refere-se ao período de tempo em que os serviços estão disponíveis ou ao tempo assumido como razoável para o sistema responder a um pedido do utilizador.

No entanto, podem ocorrer interrupções planejadas com custo para o utilizador, como o *upgrade* de um sistema operacional, podendo implicar a retomada dos serviços por parte de uma outra máquina destinada para o efeito (máquina em *standby*).

A disponibilidade de um serviço é calculada com base na percentagem que quantifica a probabilidade de encontrar o serviço operacional em determinado momento. A esta percentagem associa-se o *uptime* do serviço, isto é, o tempo em que este se encontra operacional. Já o *downtime* consiste no tempo em que se encontra fora de serviço. Esta percentagem é normalmente associada ao termo "número de noves de disponibilidade", onde uma solução de "5 noves" possui um *uptime* de 99.999%. A seguir são apresentadas duas formas de calcular a disponibilidade de um serviço.

$$Disponibilidade(\%) = \frac{Total\ Disponibilidade}{Total}$$

Por exemplo, para uma rede que se encontra disponível 165 horas por semana, o máximo possível seria 24horas/dia x 7dias/semana = 168 horas/semana. A percentagem de disponibilidade, neste caso, é igual a 165/168 = 98.21%.

Outra forma de identificar a disponibilidade de uma rede pode ser através da soma dos tempos de paragem e de recuperação de um determinado componente que lhe esteja associado

$$Disponibilidade = \frac{MTBF}{MTBF + MTTR}$$

em que MTBF (*Mean Time Between Failure*) é o tempo médio de paragem entre falhas de um componente e MTTR (*Mean Time To Repair*) é o tempo médio de reparação desse componente.

Aplicada a serviços de rede, a fórmula é modificada para a seguinte.

$$Disponibilidade = \frac{MTBSO}{MTBSO + MTTSR}$$

sendo MTBSO (*Mean Time Between Service Outage*) o tempo médio de interrupção de um serviço e MTTSR (*Mean Time To Service Repair*) o tempo médio de reparação desse serviço em caso de falha.

A tabela a seguir apresenta exemplos de medições da disponibilidade em percentagem de tempo útil. A medição é efetuada em termos dos períodos de tempo em que o sistema se encontra disponível (*uptime*) e indisponível (*downtime*), este último por ano e por semana.

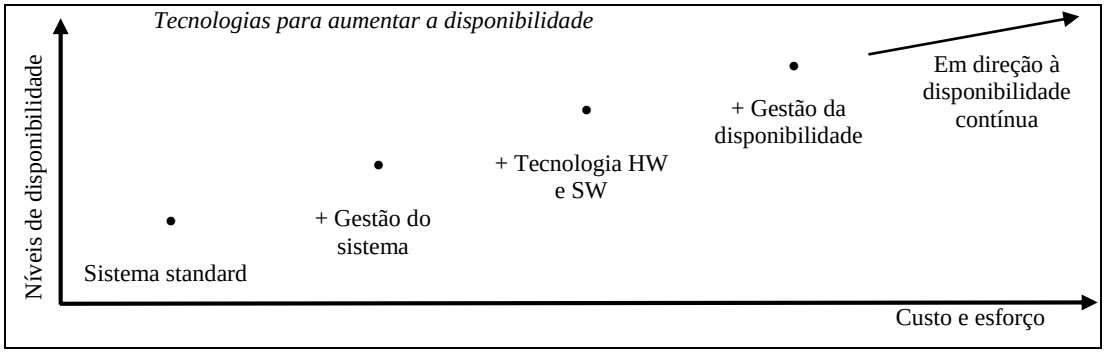
	Uptime	Downtime	Downtime por Ano	Downtime por Semana
1	90%	10%	36.5 dias	16 horas e 51 minutos
2	98%	2%	7.3 dias	3 horas e 22 minutos
3	99%	1%	3.65 dias	1 hora e 41 minutos
4	99.8%	0.2%	17 horas e 30 minutos	20 minutos e 10 segundos
5	99.9%	0.1%	8 horas e 45 minutos	10 minutos e 5 segundos
6	99.99%	0.01%	52.5 minutos	1 minuto
7	99.999%	0.001%	5.25 minutos	6 segundos
8	99.9999%	0.0001%	31.5 minutos	0.6 segundos

Medindo a disponibilidade

Alguns valores desta tabela podem ser associados a determinados serviços:

- Computadores pessoais e sistemas experimentais;
- Sistemas de acesso;
- (5) Servidores de Internet;
- (6) CPD (*Collaborative Product Development*) e sistemas de negócios;
- (7) Sistemas de telecomunicações, de saúde e bancários;
- (8) Sistemas de defesa militar.

A disponibilidade pode ser representada através de uma sequência em que cada ponto ao longo do eixo possui um custo associado e cada melhoria pode ser obtida através de investimento em recursos e/ou tecnologias adicionais. O conceito de disponibilidade pode ser dividido em vários níveis: Disponibilidade Básica (*Base Availability*), Disponibilidade Melhorada (*Improved Availability*), Alta Disponibilidade (*High Availability*) e Disponibilidade Contínua (*Continuous Availability*).



Sequência da disponibilidade

A figura demonstra que para aumentar os níveis de disponibilidade de um serviço é inevitável um aumento do custo e do esforço que lhe estão associados. Assim, a disponibilidade contínua não pode ser obtida apenas através de um produto comercial, mas sim utilizando um planeamento rigoroso e um conjunto de tecnologias envolvendo *hardware* e *software*.

Sendo assim, a regra fundamental para alcançar os níveis mais altos de disponibilidade passa por uma sólida implementação das técnicas base e de uma rigorosa gestão do sistema. Caso não seja aplicada a correta estratégia, o investimento efetuado na compra de *hardware* e *software* poderá ser em vão.

2.6. Tolerância a falhas

A noção de tolerância a falhas (*fault tolerance*) surgiu em 1967 por Avizienis[1]. Um sistema de tolerância a falhas tem a capacidade de continuar um serviço apesar da existência de uma falha de *hardware* ou *software*, mas não em caso de erro humano. Este tipo de sistemas pode ser construído recorrendo à redundância ao nível do *hardware* (CPU, memória e subsistema de I/O) e do *software* (várias aplicações desenvolvidas por diferentes equipes com as mesmas especificações, mas com diferentes implementações). A tolerância a falhas passa pela utilização de equipamento redundante que entra automaticamente em funcionamento após a detecção de uma falha no equipamento principal.

Um exemplo pode ser uma UPS (*Uninterruptible Power Supply*) que garante a continuidade do serviço em caso de falha ao nível da alimentação de um servidor.

Exemplo de equipamento para tolerância a falhas (*datacenter* UPS).

2.7. Disaster recovery

Uma organização está constantemente vulnerável à ocorrência de vários incidentes, como ataques de *hackers*, vírus nos computadores, falhas na energia elétrica, falhas ou cortes na cablagem, ou desastres naturais. Estes incidentes não são, na sua maioria, previsíveis e podem destruir parcial ou totalmente informação vital para o funcionamento da empresa.

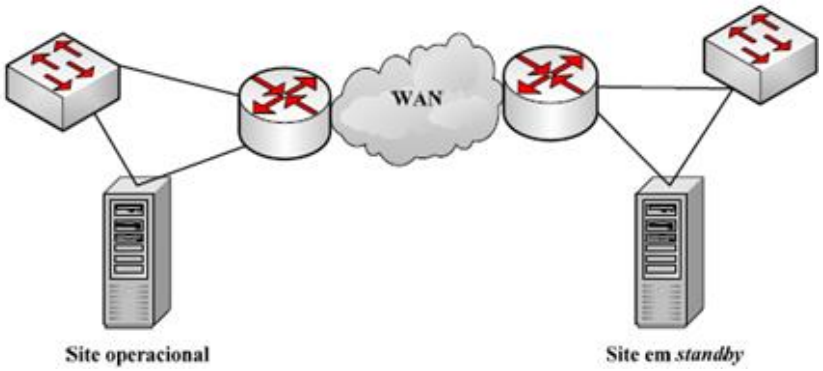
Para que essa informação possa ser recuperada no mais curto espaço de tempo a empresa deverá possuir um plano de *disaster recovery* (DR), cuja função se resume à recuperação de grande parte da informação perdida durante o acidente. A aplicação desse plano passa pela redundância empregue a nível de *hardware*, como diversos discos rígidos em diversos locais com a mesma informação, e de *software*, como aplicações que efetuam periodicamente atualizações nos diversos discos.

Normalmente os planos de DR focam, numa primeira abordagem, a recuperação de recursos de grandes dimensões, nomeadamente grandes sistemas de computação e armazenamento de dados. Estes planos incluem processos para armazenamento de dados em *backup* num ou mais lugares onde é improvável acontecer um desastre, bem como um processo de conversão para tecnologias de *backup* caso as tecnologias principais sejam afetadas pelo desastre.

Embora não exista um método totalmente eficaz, o recurso a alguns procedimentos simples pode ajudar a evitar possíveis perdas em caso de acidentes. Os mais utilizados são o uso de

equipamentos *Layer3* como pontos de decisão para encaminhamentos diferentes, a introdução de redundância e a utilização de ligações ponto-a-ponto como ligações de *backup*.

Nessa figura é representada uma possível rede onde a informação é armazenada em diferentes locais, em que um dos locais funciona como a parte ativa da rede e o outro como *standby*, sendo ativado em caso de falha do primeiro.

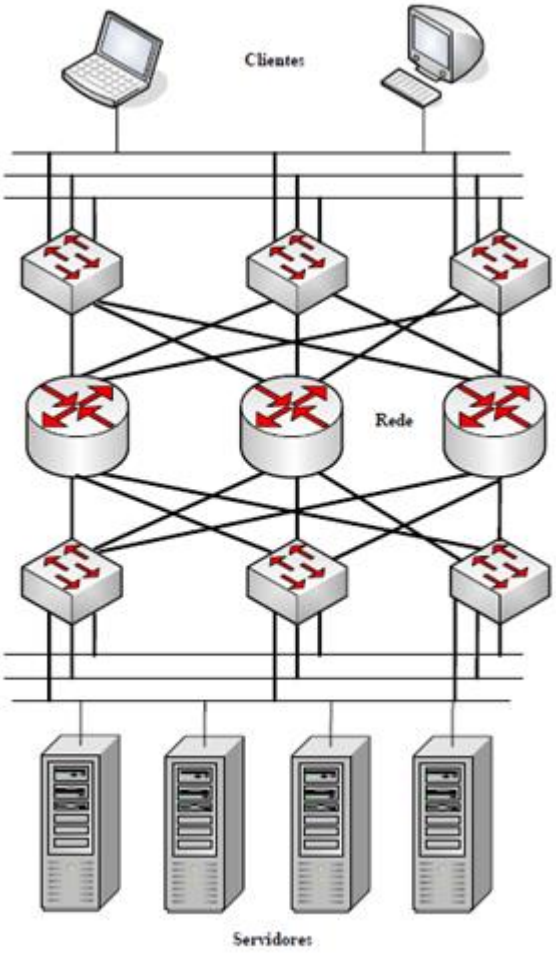


Possível cenário para *disaster recovery*

2.8. Disponibilidade contínua

Este conceito envolve um serviço permanente sem nenhuma falha o que implica HA constante de 24 horas/dia. A disponibilidade contínua (*continuous availability*) combina as características da operação contínua e da alta disponibilidade representando um estado ideal. É geralmente usada quando se pretende um elevado nível de disponibilidade, onde o *downtime* é o mínimo possível.

Neste nível de disponibilidade, o sistema nunca falha na entrega do serviço, sendo usual a execução simultânea de uma tarefa (processamento paralelo) em duas máquinas distintas. Este tipo de sistemas tenta fornecer disponibilidade a 100% ao utilizador final através da aplicação de equipamentos redundantes e da capacidade de recuperação total de erros. Nessa figura é apresentado um cenário possível de disponibilidade contínua.



Possivel cenário de disponibilidade contínua

3. SISTEMAS CENTRALIZADOS

Um **sistema de informação centralizado** (**SIC**) é aquele executado em uma coleção de máquinas, que se utiliza de seus recursos individuais e possui uma máquina servidora que centraliza todas as informações. São sistemas que possuem pouco poder de processamento sequencial (tempo compartilhado) e necessitam de um *mainframe* para que possa funcionar com qualidade. Porém, por maior que seja a velocidade de processamento de um mainframe, ele jamais conseguirá alcançar o poder de processamento de vários microcomputadores interligados, como se fosse um único sistema.

3.1. Modelos de sistemas centralizados

Existem três modelos de sistemas centralizados: **Monousuário**, **Cliente-servidor** (duas camadas) e **Multicamadas**.

O Sistema Monousuário é um sistema que funciona apenas em uma máquina. Onde a aplicação e o banco de dados estão armazenados em um mesmo computador.

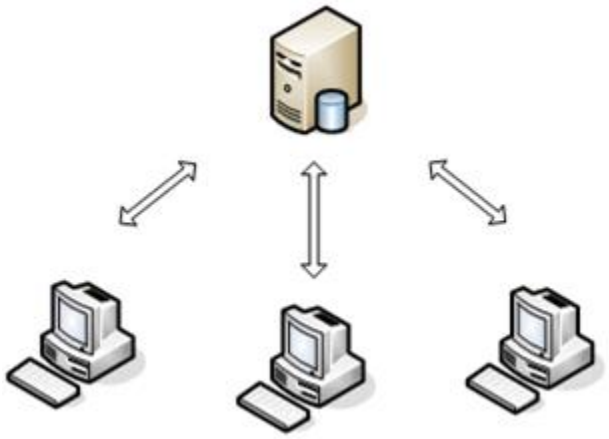
O Sistema Cliente Servidor é o tipo de sistema onde existe a separação de uma máquina servidor, que fica à disposição das requisições realizadas pelas máquinas clientes.

Como características do cliente servidor, geralmente, são utilizadas as máquinas clientes com pouca capacidade e um servidor robusto. Uma outra forma de especificar essa característica é utilizando a expressão "servidor gordo e cliente magro". As regras de negócio ficam armazenadas no próprio banco de dados por "*views*" e "*storage procedures*". No cliente, fica somente a interação do sistema com o usuário, as chamadas interfaces, como as telas e os relatórios.

Já o sistema Multicamadas é uma melhoria do cliente servidor (duas camadas), em que são separadas as regras de negócio, a interface e o banco de dados.

A segurança é um ponto fundamental em qualquer sistema de informação e uma das vantagens dos sistemas centralizados é que eles possuem um único host que fornece alto grau de segurança, concorrência e controle de cópias de segurança e recuperação. Ao contrário dos sistemas distribuídos que é mais fácil acessar os dados, o que dificulta garantir a segurança dos dados existentes e a privacidade dos dados secretos.

Além disto, em um sistema centralizado não há necessidade de um diretório distribuído, já que todos os dados estão localizados em um único host. Porém todos os acessos aos dados realizados por outro que não seja o host, onde o banco de dados está, gera alto custo de comunicação. O host em que o banco de dados está localizado pode ainda criar um "gargalo" (um elemento é o gargalo quando limita o desempenho do sistema), dependendo da quantidade de acessos simultâneos. Podem acontecer também problemas de disponibilidade dos dados, se o host, onde os dados estão armazenados, sair do ar.



Exemplo de um sistema centralizado

4. SISTEMAS DISTRIBUIDOS

Os **sistemas de informação distribuídos** (**SID**) ficaram mais populares depois da explosão da Internet em 1993 e, desde então, estes sistemas não param de crescer, tanto no meio acadêmico como no meio comercial. A principal motivação na construção de um sistema distribuído é o compartilhamento de recursos tais como: impressoras, arquivos, páginas web, acesso a banco de dados distribuídos, etc., porém, é muito mais do que isto. Um SID é um conjunto de processos concorrentes acessando recursos distribuídos, os quais podem ser compartilhados ou replicados, através de troca de mensagens em um ambiente de rede. Durante décadas, pesquisadores e profissionais enfatizaram a importância destes sistemas, muito utilizados atualmente, principalmente em ambientes que necessitam ter escalabilidade, alto desempenho, tolerância a falhas e heterogeneidade.

Um sistema distribuído é aquele executado em uma coleção de máquinas, porém que aparece como um único computador para seus usuários, onde um software de sistema distribuído permite computadores coordenarem suas atividades e compartilhar recursos do sistema (hardware, software e dados), através da troca de mensagens.

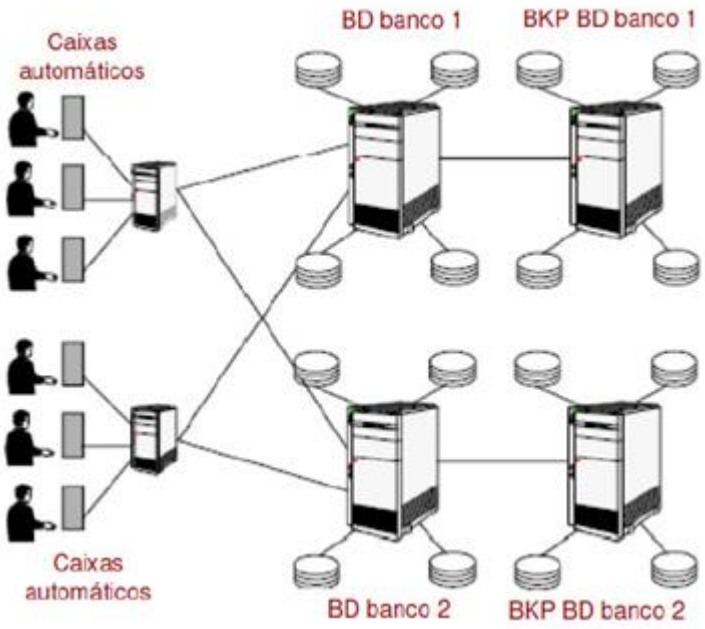
Como vimos, existem diversas definições de SID. Se referindo ao conceito de transparência, TANENBAUM (2002) define sistemas distribuidos como:

"uma coleção de computadores autônomos conectados por uma rede de comunicação que é percebida pelos usuários como um único computador que provê um serviço ou resolve um problema"

A segurança das informações em qualquer sistema é extremamente importante, pois é ela que garante que o sistema conseguirá fazer aquilo pelo qual foi projeto, de maneira correta e para os usuários corretos. Muitas informações mantidas ou que trafegam em sistemas distribuídos são sensíveis e sigilosas, portanto sua segurança é fundamental e consiste em três aspectos: confiabilidade, integridade e disponibilidade.

Uma característica marcante dos sistemas de informação distribuídos, é que são construídos a partir de uma variedade de redes, sistemas operacionais, hardwares e linguagens de programação variadas, é a transparência, cujo objetivo é tornar certos aspectos da distribuição e da funcionalidade do sistema invisíveis ao usuário, parecendo não existir, quando na verdade existem. A transparência é o atributo que esconde de usuários/aplicativos detalhes de funcionamento do sistema distribuído, de tal forma que exista a impressão de se tratar de um sistema centralizado.

Os SIDs possuem grande tolerância a falhas, ou seja, grande capacidade do sistema sobreviver a falhas em algum dos seus elementos. Uma falha em algum tipo de componente poderá ocorrer o isolamento do mesmo e dos computadores dependentes, mas não impede que o resto do sistema continue funcionando normalmente.



Exemplo de um sistema distribuido

Abaixo é apresentado um quadro comparativo entre os sistemas centralizados e os sistemas distribuidos:

Centralizado	Distribuido
Controle central	Autonomia
Consistência global	Consistência fraca
Execução sequencial	Execução concorrente
Vulnerabilidades	Tolerência a falhas
Informação local	Informação remota
Localização fixa	Migração
Homogeneidade	Heterogeneidade
Custo alto	Boa relação custo x benefício
Acesso direto	Transparência
Rigidez na expansão	Escalabilidade

4.1. Características dos Sistemas Distribuidos

As principais características dos sistemas distribuidos, segundo SOUZA apud PRADO (2010) e COULOURIS (2012) são:

- **Concorrência**: Caracterizado pelo compartilhamento de recursos com uma melhor utilização da carga de processamento entre todas as máquinas. Os recursos de um sistema distribuído devem ser compartilhados, tanto hardware (discos, impressoras, unidade de cd) como software (arquivos, banco de dados, aplicativos). Onde esses recursos são fisicamente encapsulados em um computador e acessados via comunicação. Todo tipo de recurso requer alguns métodos e políticas de gerenciamento. Isto inclui um esquema de nomes, mapeamento de nomes dos recursos para endereços concorrentes.
- **Heterogeneidade:** Diversidade de elementos computacionais. Como plataforma de *hardware*, sistema operacional, rede, linguagens de programação, padrões de representação de dados (IDL, XML), bibliotecas (DLL, POSIX), invocação de serviços (COM, CORBA, RMI, SOAP), implementações diferentes do mesmo conjunto de protocolos para diferentes tipos de rede (IP, TCP, UDP, SMTP), plataformas de execução (JVM - JAVA, CLR - .NET)
- **Abertura:** É a característica que determina se um sistema pode ser entendido em diversas maneiras. Essa abertura pode ser em relação tanto para software como hardware. Sistemas distribuídos abertos são baseados no fornecimento de um mecanismo de comunicação inter-processos uniforme e interfaces publicadas para acesso aos recursos compartilhados. Exemplo: Unix, BDS Unix.
- **Escalabilidade:** É a facilidade de inclusão de novos componentes em um sistema distribuído sem acarretar problemas, ou seja, a inclusão de novos usuários ou componentes é transparente para o usuário e para o sistema distribuído.
- **Tolerância à falhas:** É a capacidade de um sistema computacional resolver problemas de hardwares ou de softwares sem a intervenção humana e sem a percepção do que está utilizando. O design de sistemas computacionais é baseado em duas abordagens: redundância de hardware e recuperação de software.
- **Transparência:** De maneira geral qualquer processo pode ser executado em qualquer máquina da rede de maneira transparente ao utilizador. É a capacidade de mudança de um caminho normalmente traçado para um de contingência sem a possibilidade de percepção de quem está utilizando o sistema. Estas transparências tanto podem ser de hardware como de software, destas transparência podemos citar: Transparência de Acesso; de Localização; de Concorrência; de Replicação; à Falhas; de Migração; de desempenho; e, de Escala.

Além de todas estas características dos sistemas de informação distribuídos, podemos destacar a economia que este sistema oferece. Isto porque, um sistema de informação centralizado de grande porte, necessita de um *mainframe* para que possa funcionar perfeitamente. Quando se utiliza um sistema distribuído em substituição a um sistema centralizado, pode-se substituir o *mainframe* por vários microcomputadores, onde é acrescentado um poder computacional de processamento de baixo custo, sendo mais viável economicamente. Isto porque o custo de um mainframe é bem mais alto que o custo de processamento por servidores distribuídos.

5. PARADIGMAS DE SISTEMAS DISTRIBUIDOS

Os sistemas distribuídos podem ser classificados de acordo com cinco paradigmas: hierárquico, cache de cpu, cliente-servidor, conjunto de processadores e orientado ao fluxo de dados.

- **Paradigma "Hierárquico"**

O paradigma “hierárquico” caracteriza-se por dispor os vários processadores de acordo com uma organização em árvore. É organizado de maneira que a capacidade computacional dos processadores seja maior a medida que esses processadores estejam mais próximos da raiz da

árvore, sendo que as funções tratadas pelos vários processadores são distribuídas de acordo com a capacidade computacional de cada processador, de modo que os processadores mais distantes da raiz tratam de serviços mais específicos e especializados, enquanto que os processadores mais próximos da raiz tratam de serviços mais globais.

- **Paradigma "Cache de CPU"**

O paradigma "cache de cpu" caracteriza-se pelo fornecimento, ao usuário, de dois níveis de capacidade computacional: o primeiro possui menor capacidade e o segundo maior. Isso é conseguido conectando-se o usuário a um computador de menor capacidade, sendo que esse estará conectado a um computador central de grande capacidade. O sistema operacional decidirá em que computador determinado serviço será realizado, de acordo com alguns dados, tais como, adequabilidade de cada máquina, custo relativo de cada máquina, taxa de transmissão entre as máquinas e carga de trabalho de cada máquina.

- **Paradigma "Cliente-Servidor"**

O paradigma "cliente-servidor" caracteriza-se pela existência de vários computadores clientes e vários computadores servidores conectados através de um subsistema de comunicação. Os computadores clientes comandam a execução da aplicação, adonando, quando necessário, os computadores servidores, a fim de que esses realizem algumas funções específicas. Servidores de arquivos, servidores de banco de dados e servidores de impressão são exemplos típicos de processos que residiriam em processadores servidores.

Esse paradigma apresenta algumas vantagens em relação ao "cache de cpu", tais como: não existe um computador central; existe a facilidade para o compartilhamento de periféricos caros; e existe economia em termos de armazenamento em memória secundária, pois dispositivos com alta capacidade de armazenamento apresentam um custo por byte armazenado inferior ao custo em dispositivos com baixa capacidade.

- **Paradigma "Conjunto de Processadores"**

O paradigma "conjunto de processadores" caracteriza-se pela existência de um conjunto de processadores disponíveis a todos os usuários, os quais estão conectados diretamente ao subsistema de comunicação. Os serviços dos usuários serão designados dinamicamente a um dos processadores. Nesse paradigma, poderão existir ainda processadores com funções específicas.

Esse paradigma é mais otimizado que o "cliente-servidor", pois evita que um processador seja dedicado exclusivamente a um usuário ou a um conjunto de usuários. Dessa forma, é possível uma distribuição balanceada dos serviços requisitados pelos usuários entre os vários elementos processadores, evitando que exista grande diferença de carga entre esses elementos processadores.

- **Paradigma "Orientado ao Fluxo de Dados"**

O paradigma "orientado ao fluxo de dados" é muito mais radical que os demais apresentados. Nesse paradigma, um programa é descrito por um grafo e cada nó do grafo (instrução do programa) é designado a um elemento processador. Os resultados gerados por um elemento são enviados a outros elementos processadores como mensagens. As instruções são executadas assincronamente, dependendo apenas da disponibilidade dos dados; isso significa dizer que uma máquina a fluxo de dados otimizada apresenta o máximo desempenho. Esse paradigma objetiva alcançar um paralelismo de granularidade fina, ou seja, os processos que executam em paralelo são aproximadamente de tamanho de uma instrução de uma máquina convencional.

6. COMUNICAÇÃO E SINCRONIZAÇÃO EM SISTEMAS DISTRIBUÍDOS

Comunicação entre processos está no coração de todo sistema distruído. Não tem sentido estudar sistemas distribuídos sem examinar cuidadosamente os modos pelos quais processos em máquinas diferentes podem trocar informações. Esta comunicação é sempre baseada em troca de mensagens de baixo nivel como a oferecida pela rede subjacente.

Sistemas distribuídos modernos frequentemente consistem em milhares ou até milhões de processos espalhados por uma rede cuja comunicação não é confiável, como a Internet. A menos que os recursos de comunicação oferecidos pelas redes de computadores sejam substituidos por

alguma outra coisa, o desenvolvimento de aplicações distribuidas em grande escala é extremamente difícil.

Antes de iniciar nossa discussão sobre comunicação em sistemas distribuídos, primeiro faremos uma recapitulação de algumas questões fundamentais relacionadas com comunicação. Protocolos de comunicações em rede formam a base para qualquer sistema deste tipo.

Para se aprofundar mais nestes conceitos de comunicação, verifique o capítulo 4 de STEEN & TANENBAUM, 2012 (http://feuc.bv3.digitalpages.com.br/users/publications/9788576051428/pages/69).

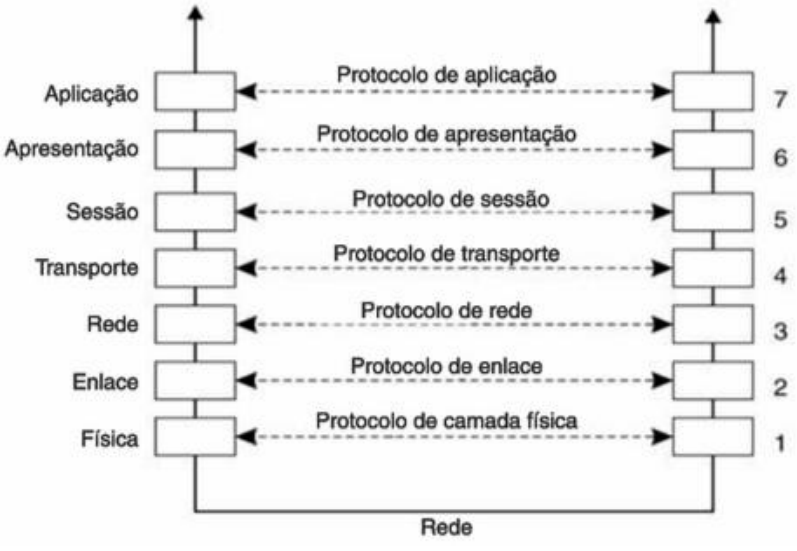
6.1. Protocolos em camadas

Devido a ausência de memória compartilhada, toda comunicação em sistemas distribuidos é baseada no envio e recebimento de mensagens (de baixo nível). Quando o processo A quer se comunicar com o processo B, em primeiro lugar ele monta uma mensagem em seu próprio espaço de endereço. Depois, executa uma chamada de sistema que faz com que o sistema operacional envie a mensagem pela rede até B. Embora essa ideia básica pareça bem simples, para evitar caos, A e B têm de concordar com o significado dos bits que são enviados. Se A enviar uma mensagem codificado segundo o código de caracteres EBCDIC da IBM e B estiver esperando em ASCII, a comunicação não será ótima.

Vários acordos diferentes são necessários. Quantos volts devem ser usados para sinalizar um bit 0, e quantos para sinalizar um bit 1? Como o receptor sabe que é o último bit da mensagem? Como ele pode detectar se a mensagem foi danificada ou perdida e o que deve fazer se descobrir que isso aconteceu? Qual o comprimeiro dos números, cadeias e outros itens de dados, e como eles são representados? Em resumo, são necessários acordos em uma variedade de níveis que vão de detalhes de baixo nível de transmissão de bits a detalhes de alto nível sobre como a informação deve ser expressa.

Para ficar mais fácil lidar com os vários níveis e questões envolvidos em comunicação, a *International Organization Standardization* (*ISO*) desenvolveu um modelo de referência que identifica claramente os vários níveis envolvidos, dá-lhes nomes padronizados e indica qual nível deve fazer tal serviço. Esse modelo é denominado **modelo de referência para interconexão de sistemas abertos** (*Open System Interconnection Reference Model*) (TANENBAUM, 2009), usualmente abreviado para ISO OSI ou, às vezes apenas **modelo OSI**.

O modelo OSI é projetado para permitir que sistemas abertos se comuniquem. Um sistema aberto é o que está preparado para se comunicar com qualquer outro sistema aberto usando regras padronizadas que regem o formato, o conteúdo e o significado das mensagens recebidas. Essas regras estão formalizadas no que denominamos **protocolos**. Se um grupo de computadores quiser se comunicar por uma rede, todos eles têm de concordar com os protocolos que serão utilizados. É feita uma distinção entre dois tipos gerais de protocolos.



Camadas, interface e protocolos no modelo OSI

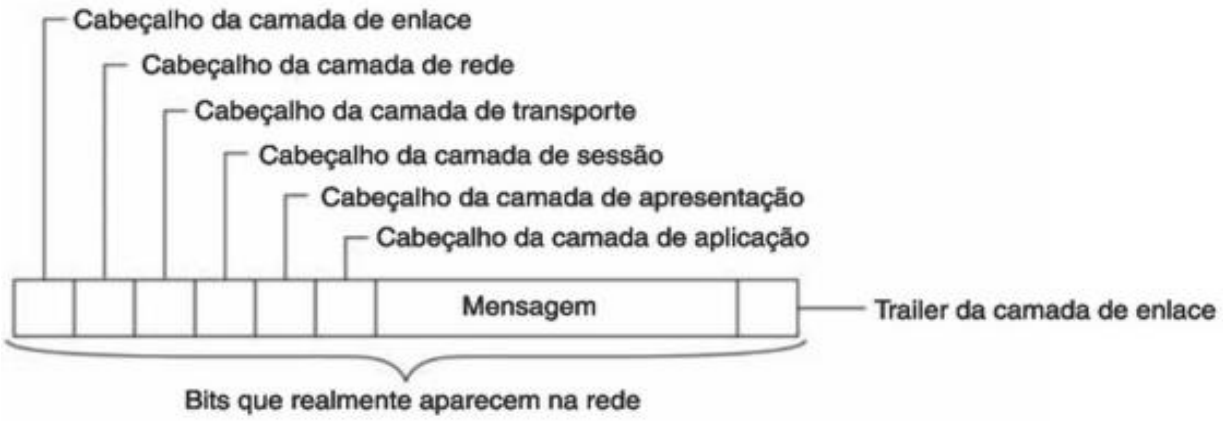
Com protocolos **orientados a conexão**, antes de trocar dados, o remetente e o receptor primeiro estabelecem explicitamente uma conexão e possivelmente negociam o protocolo que usarão. Após concluírem, devem liberar a conexão. O telefone é um sistema de comunicação orientado a conexão. Quando o protocolo é **sem conexão**, não é preciso estabelecer nada antecipadamente. O remetente apenas transmite a primeira mensagem quando estiver pronta. Um exemplo de comunicação sem conexão é colocar uma carta em uma caixa de correio. Em computadores, ambos os tipos de comunicação são comuns.

No modelo OSI, a comunicação é dividida em até 7 níveis ou camadas, como mostra a figura.

Cada camada lida com um aspecto específico da comuicação. Desse modo, o problema pode ser dividido em porções gerenciáveis, e cada uma delas pode ser resolvida independentemente das outras. Cada camada fornece uma interface para a camada que está acima dela. A interface consiste em um conjunto de operacões que, juntas, definem o serviço que a camada está preparada para oferecer a seus usuários.

Quando o processo A na máquina 1 quer se comunicar com o processo B na máquina 2, ele constroi uma mensagem e passa essa mensagem para a camada de aplicação em sua própria máquina. Em seguida, o software de camada de aplicação adiciona um **cabeçalho** à frente da mensagem e passa a mensagem resultante para a camada de apresentação por meio da interface entre as camadas 6 e 7. Por sua vez, a camada de apresentação adiciona seu próprio cabeçalho e passa o resultado para a camada de sessão, e assim por diante.

Algumas camadas não se limitam a adicionar um cabeçalho à frente da mensagem, adicionam também um ***trailer*** ao final. Quando a mensagem chega ao nível mais baixo, a camada física transmite a mensagem (cujo aspecto poderia estar parecido com o mostrado a seguir) colocando-o no meio físico de transmissão.



Mensagem típica tal como aparece na rede

Quando a mensagem chega à máquina 2, ela é passada para cima e cada camada retira e examina seu próprio cabeçalho. Por fim, a mensagem chega ao receptor, o procesos B, que pode respondê-la usando o caminho inverso. A informação contida no cabeçalho da camada *n* é usada para o protocolo da camada *n*.

Como dito anteriormente, dispor de facilidades poderosas e flexíveis para comunicação entre processos é essencial para qualquer sistema distribuído. Em aplicações tradicionais de rede, a comunicação costuma ser baseada nas primitivas de troca de mensagem de baixo nível oferecidas pela camada de transporte. Uma questão importante em sistema middleware é oferecer um nível mais alto de abstração que facilitará expressar comunicação entre processos mais do que o suporte oferecido pela interface com a camada de transporte.

Uma das abstrações mais amplamente utilizadas é a chamada de procedimento remoto (RPC). A essência de uma RPC é que um serviço é implementado por meio de um procedimento cujo corpo é executado em um servidor. O cliente recebe apenas a assinatura do procedimento, isto é, o nome do procedimento junto com seus parâmetros. Quando o cliente chama o procedimento a implementação do lado do cliente, denominada apêndice, fica encarregada de embrulhar os valores dos parâmetros em uma mensagem e enviá-la ao servidor. Este chama o procedimento propriamente dito e retorna os resultados, mais uma vez em uma mensagem. O apêndice do cliente extrai os valores do resultado da mensagem de retorno e a passa de volta à aplicação cliente chamador.

RPCs oferecem facilidades de comunicação síncrona, pelas quais um cliente é bloqueado até que o servidor tenha enviado uma resposta. Embora existam variações de qualquer um dos mecanismos pelos quais esse modelo síncrono estrito é amenizado, ocorre que os modelos de uso geral de alto nível orientados a mensagens muitas vezes são mais convenientes.

Em modelos orientados a mensagem, as questões giram em torno de se uma comunicação é ou não persistente e se uma comunicação é ou não síncrona. A essência da comunicação

persistente é que uma mensagem apresentada para transmissão é armazenada pelo sistema de comunicação pelo tempo que for necessário para entregé-la. Em outras palavras, nem o remetente nem o receptor precisam estar ligados e funcionando para que a transmissão da mensagem ocorra. Em comunicação transiente, nenhuma facilidade de armazenamento é oferecida, de modo que o receptor deve estar preparado para aceitar a mensagem quando ela for enviada,

Em comunicação assíncrona, o remetente tem permissão de continuar imediatamente após a mensagem ter sido apresentada para transmissão, possivelmente antes mesmo de ela ter sido enviada. Em comunicação síncrona, o remetente é bloqueado no mínimo até que uma mensagem seja recebida. Alternativamente, o remetente pode ser bloqueado até ocorrer a entrega da mensagem ou mesmo até que o receptor tenha respondido, como acontece com as RPCs.

Modelos de middleware orientado a mensagem em geral oferecem comunicação assíncrona persistente e são usados onde RPCs não são adequadas. Esses sistemas costumam ser utilizados para ajudar a integração de conjuntos de bancos de dados a sistemas de informações de grande escala. Entre outras aplicações estão e-mail e fluxo de trabalho.

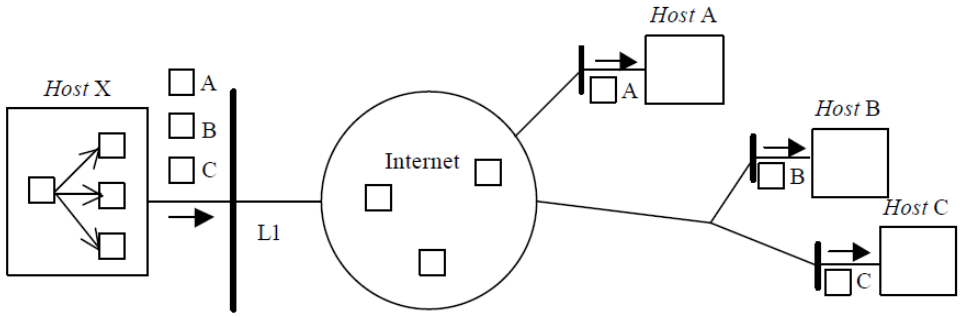
Uma forma muito diferente de comunicação é a comunicação por fluxos, na qual a questão é se duas mensagens sucessivas têm ou não uma relação temporal. Em fluxos contínuos de dados, um atraso fim-a-fim máximo é especificado para cada mensagem. Além disso, também é requerido que as mensagens sejam enviadas sujeitas a um atraso fim-a-fim mínimo. Exemplos típicos desses fluxos contínuos de dados são os fluxos de áudio e vídeo. Em geral, é difícil específicar e implementar quais são, exatamentem as relações temporais ou o que se espera do subsistema de comunicação subjacente em termos de qualidade de serviço. Um fator complicador é o papel da variância do atraso. Ainda que desempenho médio seja aceitável, variações substanciais no tempo de entrega podem resultar em desempenho inaceitável.

Por fim, uma importante classe de protocolos de comunicação em sistema distribuidos é o *multicasting*. A ideia básica é disseminar informações de um remetente para vários receptores.

6.2. Multicasting

Em comunicações tradicionais envolvendo vários pontos simultaneamente, para cada pacote do *host* de origem é feita uma replicação para o número de *hosts* destinos, e cada pacote é enviado para seu destino separadamente. Este modelo impõe uma limitação no número de máquinas que poderiam estar envolvidas na comunicação, pois o tráfego gerado na rede e as necessidades computacionais do *host* de origem – que gera cópias de cada pacote – aumentam linearmente com o número de *hosts* destinos envolvidos.

A figura a seguir exemplifica uma transmissão tradicional. No exemplo, um pacote deve ser enviado para os *hosts* destinos A, B e C. Ele é replicado no *host* de origem e três cópias são enviadas, consumindo maior desempenho no *host* X e maior largura de banda na rede, mesmo que alguns dos segmentos de rede sejam comuns aos três.

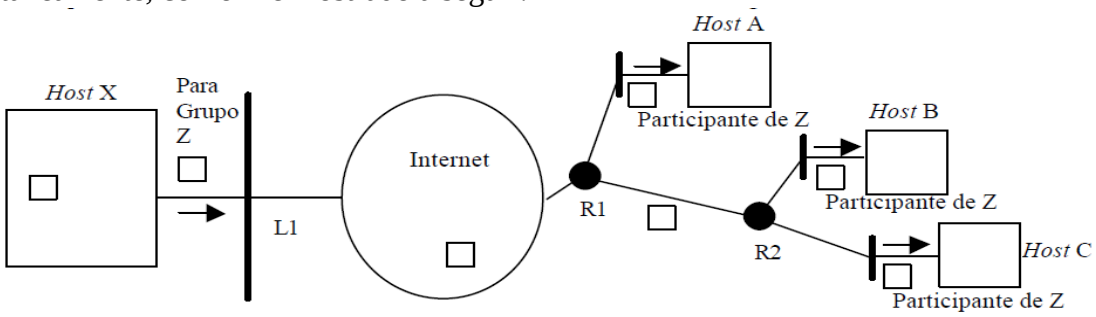


Transmissão com replicação do datagrama[1].

Existem, porém, tecnologias que tratam estas limitações, sendo chamadas soluções escalares de rede. O uso da difusão seletiva IP é uma destas soluções. Utilizando difusão seletiva,

[1] Pacote, trama ou datagrama é uma estrutura unitária de transmissão de dados ou uma sequência de dados transmitida por uma rede ou linha de comunicação que utilize a comutação de pacotes. A informação a transmitir geralmente é quebrada em inúmeros pacotes e então transmitida.

apenas uma cópia de cada pacote é enviada por enlace, sendo copiada apenas quando houver um braço na árvore lógica dos destinos. A utilização de difusão seletiva fornece um ganho de processamento de CPU e de largura de banda, quando vários *sites* estão envolvidos simultaneamente, conforme mostrado a seguir.



Transmissão sem replicação do datagrama

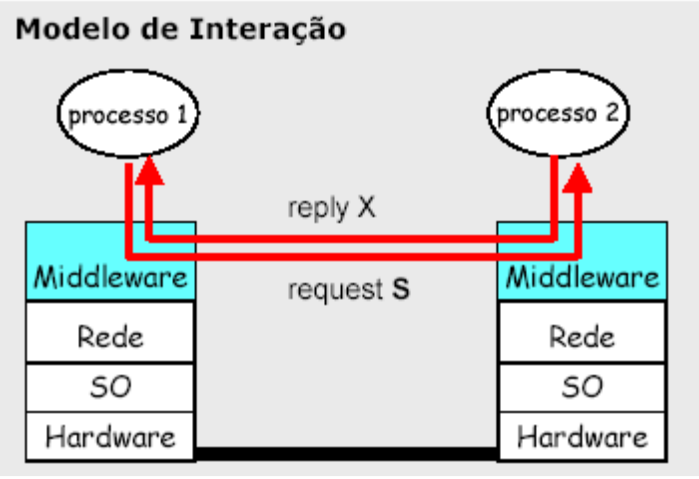
Na figura anterior, o *host* X não mais replica os datagramas para cada *host* destino, participante do grupo Z (A, B e C). Este envia apenas uma cópia possuindo o endereço classe D do grupo. O tráfego sobre o enlace L1 permanece sempre o mesmo, independentemente do número de participantes do grupo. O datagrama será replicado apenas quando houver destinos separados. Neste caso o datagrama será replicado no último roteador que suporte difusão seletiva comum aos dois caminhos. Os roteadores de difusão seletiva que são responsáveis por replicar os datagramas estão representados por R1 e R2.

7. CONCEITOS DE MIDDLEWARE

O termo *middleware* caracteriza uma camada de software que possibilita comunicação entre aplicações distribuídas - tendo por objetivo diminuir a complexidade e heterogeneidade dos diversos sistemas existentes -, provendo serviços que realizam a comunicação entre esta categoria de aplicações de forma transparente às mesmas. A adaptação entre sistemas heterogêneos é necessária, por exemplo, quando um sistema atual deve interoperar com sistemas obsoletos ou com diferentes empresas. Um *middleware* é dividido em componentes do ambiente de programação e do ambiente de execução.

A figura ao lado ilustra o processo de comunicação entre aplicações distribuídas através de um *middleware*. Todo o processo é transparente para as aplicações, e a heterogeneidade existente é tratada também pelo *middleware*.

Inicialmente, um *middleware* tinha a função básica de unir os componentes de um programa distribuído, ditando a maneira pela qual estes componentes interoperavam. Atualmente, sua função é integrar aplicações completas entre e dentro de organizações.



Comunicação através de *middleware*

No que se refere à integração entre aplicações escritas em diferentes linguagens de programação, poucos middlewares suportam esta característica. Um bom exemplo de integração entre diferentes linguagens é o *Commom Object Request Broker Architecture* (*CORBA*), pois neste *middleware* é possível fazer o mapeamento de uma *Interface Definition Language* (*IDL*) em muitas linguagens de programação.

Um ponto importante na utilização e funcionalidade dos *middlewares* é a sua padronização, o que causa problemas relacionados à integração, facilidade de uso e gerenciamento. A

padronização, entretanto, é importante para auxiliar os consumidores a selecionarem middlewares baseados em qualidade.

Duas características importantes na construção de um *middleware* são sua flexibilidade e performance. Porém, estes dois pontos são mutuamente exclusivos, já que o alto nível de flexibilidade acaba impedindo um alto nível de performance, sendo o ideal um balanceamento entre os dois.

Outro ponto a ser destacado na utilização dos *middlewares* é a redução na complexidade do sistema, pois, ao utilizar um *middleware*, o sistema terá sua complexidade reduzida, sendo que este é um dos objetivos principais na utilização desta camada intermediária.

Atualmente, o principal desafio enfrentado pelo *middleware* é a facilitação da integração entre aplicações para Internet. Assim como a integração entre consumidores e a Internet está acontecendo a cada dia que passa, é importante que haja integração entre aplicações via este mesmo meio de acesso, vencendo, assim, as limitações dos *browsers* no que se refere ao modelo de aplicações de duas camadas.

7.1. Serviços do Middleware

O serviço oferecido pelo *middleware* é um serviço de propósito geral, situado entre plataformas (serviços de baixo nível) e aplicações, sendo caracterizado pelas APIs (*Application Programmer Interface*) e pelos protocolos que suporta.

Devido às propriedades dos componentes de um *middleware*, seus serviços não são dependentes de plataforma ou aplicação, sendo genéricos entre as diversas aplicações dos diversos fabricantes, suportando interfaces e protocolos padrões.

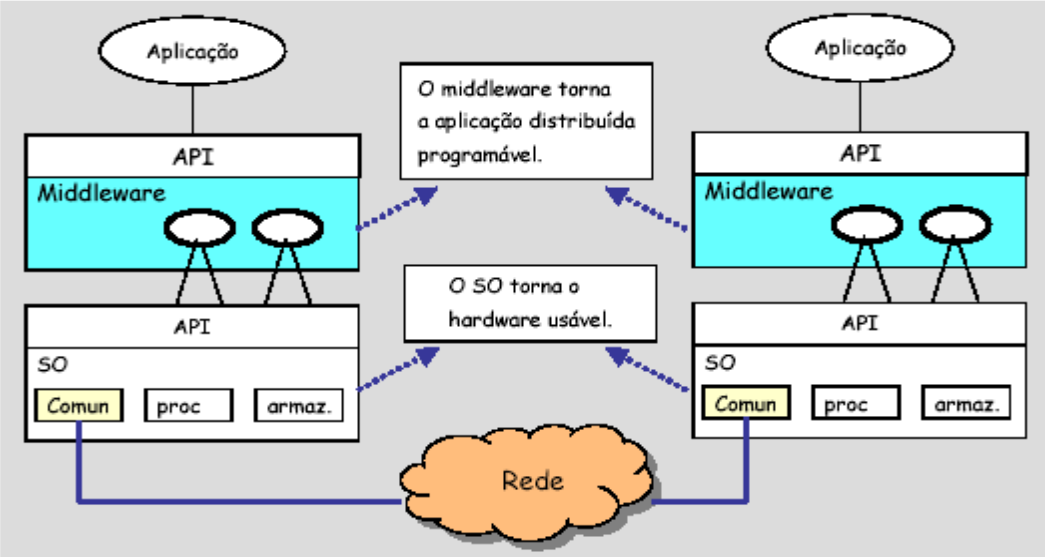
Para que seja considerado um *middleware*, o serviço oferecido deve atender às necessidades de uma gama de aplicações de um determinado domínio, não se concentrando apenas em serviços fornecidos para uma aplicação específica, além de ter implementação independente de plataforma. Esta independência de plataforma é conseguida no momento da implementação dos serviços, quando se busca atingir a portabilidade, o que significa que o *middleware* poderá ser transferido para outra plataforma com o mínimo esforço.

Um serviço fornecido por um *middleware* deve dar suporte a pelo menos uma API padrão, sendo considerado transparente em relação a esta API se puder ser acessado por ela sem a modificá-la. Porém, o que se encontra são serviços implementados sobre API's proprietárias ou protocolos que ainda não foram oficialmente publicados, fato que dificulta a padronização entre os diferentes vendedores. Os *middlewares* comumente fornecem os seguintes serviços:

- **Gerenciamento de Apresentação:** Gerenciamento de formulários, gráficos, impressora e hipermídia.
- **Computação:** Ordenação, dispositivos matemáticos, serviços internacionalizáveis, conversores de dados e serviço de tempo.
- **Gerenciamento de informação:** Servidor de diretórios, gerenciador de *log*, gerenciador de arquivos.
- **Comunicação:** Mensagens *Peer-to-Peer*, chamada remota de procedimento, fila de mensagens, mensagem eletrônica e exportação eletrônica de dados.
- **Controle:** Gerenciamento de transações, de *threads*.
- **Gerenciamento do Sistema:** Serviço de notificação de eventos, serviço de contas, gerenciamento de configuração, detector de falhas.
- **Sistema de entrega:** Um padrão proposto é o *Java Virtual Machine* (JVM) para o sistema de entrega. Outro exemplo é o *Remote Procedure Call* (RPC).
- **Comunicação entre processos:** É o coração do *middleware*. Como exemplo pode-se citar o *Object Request Broker* (ORB) do CORBA, que tem por objetivo a integração entre aplicações remotas.
- **Interface com o usuário:** Como exemplo tem-se o HTML, e os formatos multimídia aceitos pela Internet.
- **Em direção de uma utilização mais universal de *middlewares* comuns:** Os esforços para a criação de *middleware* estão centrados, atualmente, em soluções proprietárias e

direcionadas a algum interesse. Os resultados disso ainda são funcionalidades caras e proibitivas. Uma das razões pelas quais o reuso ainda é utilizado é a facilidade de agrupar soluções direcionadas, que permanece invisível a todos menos ao desenvolvedor. Uma solução para este problema é a conscientização dos programadores em relação ao custo do ciclo de vida de desenvolvimento que está por trás da interconexão de componentes individuais.

- **Soluções comuns suportando tanto variabilidade quanto controle:** Os sistemas tendem a trabalhar de uma maneira adequada se tiverem todos os recursos necessários para o seu correto funcionamento. Há pouca ou nenhuma flexibilidade em seus comportamentos. O que é necessário, então, é uma reconfiguração do sistema em relação aos recursos disponíveis a eles, que tem dois focos: o comportamento individual e o agregado.



Contexto do *middleware*

A figura anterior ilustra o contexto do *middleware* na comunicação entre aplicações distribuídas e o sistema operacional. Um *middleware* utiliza API's fornecidas pelo sistema operacional (S.O) para que a comunicação distribuída seja facilitada. A responsabilidade do S.O é apenas gerenciar o funcionamento do *hardware*.

Outros serviços incluídos são gerenciamento de qualidade de serviço (QoS – *Quality of Service*) e segurança da informação. Apesar da integração entre QoS e *middleware* ser importante, é necessário que haja uma padronização para que este procedimento possa ser realizado.

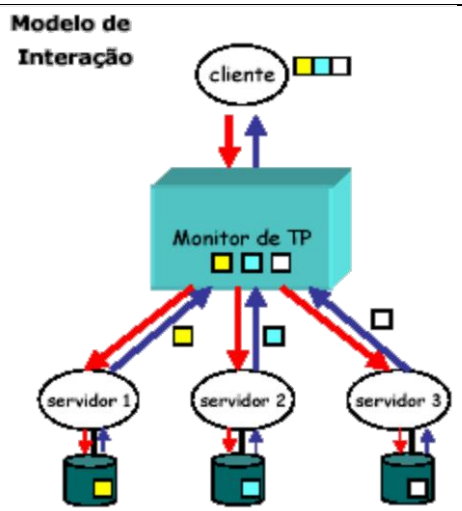
7.2. Classificação dos Middlewares

A utilização de *middlewares* não é transparente ao desenvolvedor. Alguns fatores indicam esta falta de transparência:

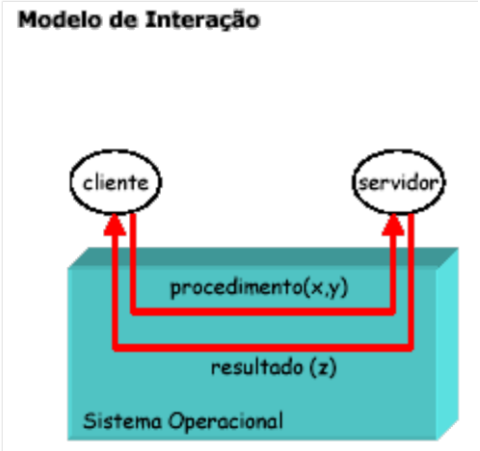
- A comunicação entre objetos distribuídos é lenta se comparada à comunicação entre objetos locais;
- A ativação e desativação de componentes gera a necessidade da implementação de persistência destes componentes;
- Os componentes devem ser desenvolvidos de maneira que possam lidar com as interações concorrentes que ocorrem em componentes distribuídos;
- Os componentes podem escolher entre as primitivas de sincronização que são oferecidas pelo *middleware* e a necessidade da exploração correta destas.

Segundo TALARIAN apud MACIEL e ASSIS (2004), os *middlewares* são classificados nas seguintes categorias:

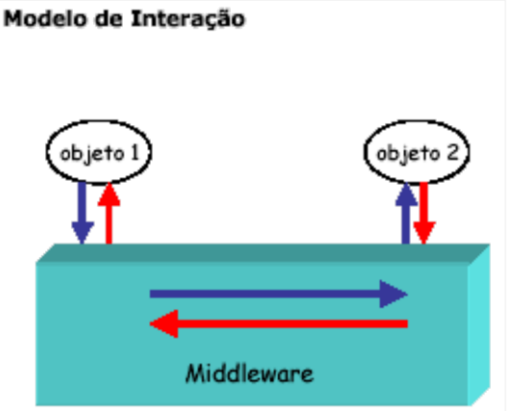
- **Monitor de Processamento de Transação:** Provê um ambiente completo para aplicações de transação que acessam banco de dados relacionais. O *overhead* de comunicação neste modelo fica reduzido ao mínimo devido à troca de mensagens se basear em simples *request/reply*. Vemos o Modelo de interação do *middleware* orientado a transação.



- ***Remote Procedure Call* (RPC):** Uma das primeiras formas de comunicação entre processos remotos, operando em baixo nível. Não se aplicam bem a aplicações grandes e em tempo real. Ao lado vemos o Modelo de interação do *middleware* RPC.

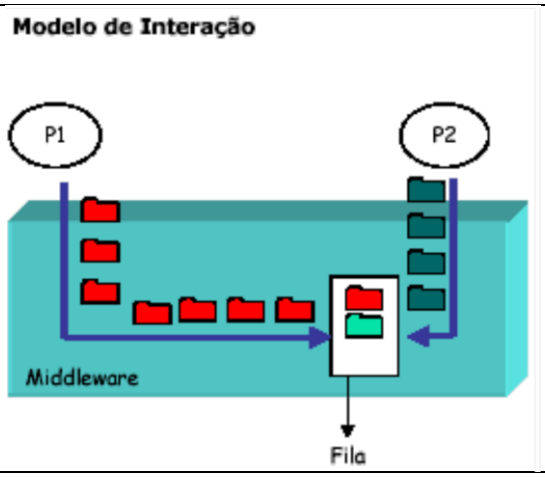


- ***Object Request Broker (ORB):*** Pode ser considerado um RPC orientado a objetos. Existem dois concorrentes: CORBA, da *Object Management Group* (OMG), e DCOM, da *Microsoft*. A lado temos o Modelo de interação do *middleware* orientado a objetos.



- ***Homegrown Middleware***: Este tipo de *middleware* se destina a aplicações específicas, ou seja, que são feitas para resolver um problema específico. Sua constante atualização e personalização o tornam caro e com flexibilidade afetadas.

- ***Orientado à mensagem (MOM):*** Funciona com base na troca de mensagens entre programas de maneira assíncrona. Ao lado temos o Modelo de interação do *middleware* orientado a mensagem.



- ***Serviço de mensagem Java (JMS):*** Especifica um conjunto de interfaces pelas quais programas em Java podem acessar *softwares* orientados a mensagem.

7.3. Tipos de middleware

Alguns tipos de middleware que serão descritos abaixo são: reflexivo, adaptativo e *Distributed Object Computing* (*DOC*).

7.3.1. *Middleware* Reflexivo

Um *reflective middleware* utiliza o conceito de refletividade computacional, ou seja, uma aplicação pode acessar algumas partes do estado do sistema logo abaixo e modificá-lo dinamicamente, modificando, então, sua própria semântica.

O uso da refletividade de maneira indisciplinada pode resultar em quebras inesperadas do sistema, principalmente se a modificação resulta em mudanças incompatíveis para uma determinada parte da aplicação.

Segundo ROMÁN apud MACIEL e ASSIS (2004), um *middleware* reflexivo explora o protocolo meta-objeto de Gregory Kiczales, combinado às ideias de refletividade computacional e orientação a objetos, sendo dividido em *base-level* e *metalevel*.

A ideia base do *base-level* é atingir a funcionalidade das aplicações, enquanto que o *metalevel* designa coleções de componentes que constituem a arquitetura interna da plataforma do *middleware*. A propriedade refletiva permite que o comportamento destes objetos seja monitorado, possibilitando mudanças no comportamento do *middleware*.

A aplicabilidade de um middleware reflexivo se concentra em aplicações de computação onipresente em tempo real, visto que a própria natureza deste tipo de aplicação difere em relação à das já existentes. A flexibilidade introduzida por um middleware reflexivo é capaz de suprir as necessidades da computação onipresente, característica essa não existente em plataformas de middleware convencionais, por serem grandes e inflexíveis.

O *middleware* reflexivo é implementado como um conjunto de componentes colaborativos que podem ser configurados ou reconfigurados pela aplicação, tendo sua interface imutável e podendo suportar aplicações desenvolvidas para *middlewares* tradicionais.

A importância dos metadados e do uso da refletividade na estrutura do middleware é que estes armazenam informações sobre o estado das aplicações que estão rodando sobre o middleware, de maneira que cada aplicação específica possui um perfil próprio, o qual, ao ser modificado, atingirá a própria aplicação e o comportamento do *middleware*. De maneira mais direta, a função dos metadados é ditar ao middleware o seu comportamento quando uma determinada ação for executada, enquanto que a refletividade é utilizada para armazenar perfis específicos para cada aplicação.

7.3.2. *Middleware* Adaptativo

O objetivo desta categoria de *middleware* é ser dinamicamente personalizável, o que possibilita às aplicações móveis a flexibilidade necessária.

Segundo YAU apud MACIEL e ASSIS (2004), apesar da natureza da adaptação de aplicações ser específica por aplicação, cabe ao middleware prover um *framework* flexível para o desenvolvimento de aplicações adaptativas e sensíveis ao contexto. O *middleware* adaptativo deve ser reconfigurável, permitindo que novos serviços sejam acoplados. Esta categoria de *middleware* deve ter as seguintes características:

- **Suporte a aplicações sensíveis ao contexto e adaptativas:** Facilita na construção de aplicações adaptativas e sensíveis ao contexto por um *middleware*.
- **Suporte em tempo de execução de comunicações *ad hoc* entre as aplicações:** Considerando que o custo da comunicação em ambientes sem fio é maior que o custo da computação neste mesmo ambiente, cabe ao *middleware* utilizar de maneira inteligente o recurso da comunicação baseado no contexto e nas necessidades reais das aplicações.
- **Serviços adaptativos específicos da aplicação:** Além dos serviços disponibilizados pelo middleware, uma aplicação pode ter serviços específicos, que envolvem segurança e gerenciamento de grupo.
- **Interoperabilidade com *middleware* de outros domínios:** Com o objetivo de fornecer a interoperabilidade necessária com *middlewares* de outros domínios, o *middleware* adaptativo deve ter suporte à interoperabilidade, o que permite que aplicações móveis se comuniquem com as aplicações já existentes sem muito esforço.

7.3.3. Distributed Object Computing Middleware (DOC)

Este tipo de middleware se destina a aplicações distribuídas, e, como em um protocolo de rede que se divide em múltiplas camadas, também possui uma arquitetura decomposta em camadas que são, de acordo com SCHANTZ apud MACIEL e ASSIS (2004):

- ***Host Infrastructure middleware:*** Encapsula a comunicação pertencente ao SO e o mecanismo de concorrência, criando, então, componentes reusáveis da programação distribuída. Estes componentes, além de ajudar na encapsulação de individualidades do sistema operacional, também auxilia na obtenção de aplicações menos susceptíveis a erro e com uma maior portabilidade entre plataformas diferentes. Alguns exemplos de *middlewares* pertencentes à esta camada são: a máquina virtual Java da Sun (JVM), a plataforma .NET da Microsoft para Web Services.
- ***Distribution Middleware:*** Define modelos de programação de auto nível dos quais as API's e os componentes podem automatizar e estender a capacidade de programação já existente no sistema operacional. *Middlewares* pertencentes à esta camada proporcionam a construção de aplicações distribuídas como se fossem aplicações locais. Dentre estes *middlewares* se encontram: CORBA, RMI, DCOM e SOAP.
- ***Commom Middleware Services:*** Define serviços de alto nível que podem ser utilizados pelos programadores para a construção de aplicações distribuídas. Estes serviços permitem que o programador não se preocupe com a forma com que sua aplicação irá se comunicar com sua parte distribuída, focando mais a lógica desta aplicação. Exemplos destes serviços são: os serviços CORBA, *Sun's Enterprise Java Beans* (EJB), .NET Web Services.
- ***Domain-specific middleware services***: São serviços específicos requeridos por determinados domínios. Ao contrário das outras camadas do DOC, que provêm mecanismos de reuso e serviços horizontais, este tipo de serviço possui seu alvo no mercado vertical. Os serviços oferecidos por esta camada possibilitam um maior crescimento da qualidade do sistema e diminuem o esforço e o ciclo de vida necessários para o desenvolvimento de determinadas aplicações distribuídas.

Os DOC *middlewares* provêm algumas funcionalidades para o desenvolvimento de aplicações distribuídas. Estas funcionalidades serão apresentadas a seguir, segundo o mesmo autor.

- **Foco na integração ao invés da programação:** A origem do middleware se dá devido ao problema em relação à integração e construção de partes separadas de uma determinada aplicação. Middlewares distribuídos, a exemplo do CORBA e do Java RMI, possibilitam que pedaços de aplicações sejam interconectados, independentemente da localização e da tecnologia utilizada. Estas funcionalidades permitem que middlewares DOC reduzam os esforços do ciclo de vida do software por considerar experiências de desenvolvimento anteriores.
- **Demanda para suporte fim-a-fim de QoS, não apenas componente QoS:** A construção de aplicações que utilizam QoS não é uma tarefa fácil, visto que há uma necessidade de implementar as propriedades não funcionais do QoS, tais quais latência previsível, *throughput*, escalabilidade, dependência, flexibilidade e recursos integrados entre e dentro dos pedaços da aplicação. Existem duas premissas que vão de encontro ao suporte fim-a-fim do QoS, que são os diferentes níveis de serviços possíveis e desejáveis, e o nível do serviço em uma propriedade deve ser coordenado para que o serviço desejado seja atingido.
- **A viabilidade crescente dos sistemas abertos:** a característica distribuída permite que os sistemas com este propósito sejam mais abertos que os sistemas monolíticos, facilitando a implementação de componentes a partir de uma múltipla escolha e do conceito de engenharia aberta.
- **Crescente demanda por tecnologias divididas que tendem ao aumento da competição global:** o custo do ciclo de vida do desenvolvimento de softwares pode ser amenizado pela consideração de conhecimento anterior e pela focalização no esforço para aumentar a qualidade do software. Quando os desenvolvedores não precisam se preocupar com os detalhes de baixo nível, eles podem se concentrar nos detalhes específicos da aplicação que estão desenvolvendo, como, por exemplo, gerenciamento de recursos distribuídos e dependência fim-a-fim.
- **Cobertura da complexidade potencial para sistemas complexos da próxima geração:** com a crescente complexidade dos sistemas distribuídos, fica difícil sustentar a integração e composição destes sem o uso contínuo de pesquisas. O que pode acontecer é a consideração de um conhecimento falho, levando a resultados de alto risco a problemas comuns.

7.4. Considerações sobre o Middleware

Os objetivos principais de um *middleware* são a integração entre sistemas heterogêneos e a intermediação entre as aplicações e o sistema operacional. Para que estes objetivos sejam alcançados, um *middleware* deve fornecer serviços que atendam ao domínio de aplicações para o qual foi construído, sendo importante que este serviço tenha sua base em uma das API's ou protocolos padrões.

A facilidade de uso dos *middlewares* é outro fator importante na sua elaboração, o que induz a criação de comandos intuitivos e aplicações com códigos fáceis de entender para se comunicar com a interface do *middleware*.

Muitos *middlewares* utilizam protocolos e API's proprietárias para o seu funcionamento, o que os limita para alguns ambientes. Códigos proprietários são necessários para resolver os problemas que não são endereçados pela padronização. Então, mesmo os vendedores que seguem a linha padrão necessitam colocar códigos proprietários para que os problemas que não estejam no domínio padrão possam ser considerados, permitindo também a concorrência entre os diversos vendedores.

Os componentes de um *middleware* estão se tornando mais importantes que os serviços oferecidos pelo sistema operacional (S.O), pois os primeiros estão substituindo as funções não distribuídas do S.O por funções distribuídas que utilizam redes. Por exemplo, as aplicações dependem mais no mecanismo de *Remote Procedure Call* (RPC) do que no nível de transporte de mensagens.

Uma desvantagem dos *middlewares* se concentra justamente em sua capacidade de amenizar a heterogeneidade, pois ele o faz adicionando uma falsa homogeneidade no sistema, o que apenas retarda a colisão entre os sistemas heterogêneos.

Os *middlewares* apresentam alguns desafios, que devem ser levados em consideração no momento da sua construção e utilização. Dentre eles pode-se destacar flexibilidade, performance, integração com a *Web* e entre *middlewares* e a computação.

8. REDES P2P

A definição do que é uma rede do tipo *peer-to-peer* (*P2P*), pode parecer obscura em um primeiro momento. Contudo, é clara a distinção entre o paradigma cliente-servidor e o paradigma P2P. No primeiro existe uma clara distinção entre a entidade que está provendo um serviço e o cliente que o está consumindo. Por outro lado, não existe distinção entre os elementos de uma rede P2P, todos os nós possuem funcionalidades equivalentes.

Uma rede *P2P* é formada por vários nós, e a comunicação e troca de dados na rede é feita diretamente entre os nós, ao invés de arbitrada por um nó intermediário. Cada nó em uma rede *P2P* pode agir simultaneamente como cliente e servidor.

Em sistemas *P2P*, cada participante da rede ("par") torna disponível uma parte de seus recursos ao mesmo tempo em que usa recursos disponibilizados por outros pares. Isto estabelece um contraste com o modelo mais usual de comunicação em redes de comutação de pacotes (tais como a Internet), que é o modelo *cliente-servidor*: nesse modelo, apenas os servidores fornecem recursos, e os clientes apenas consomem recursos.

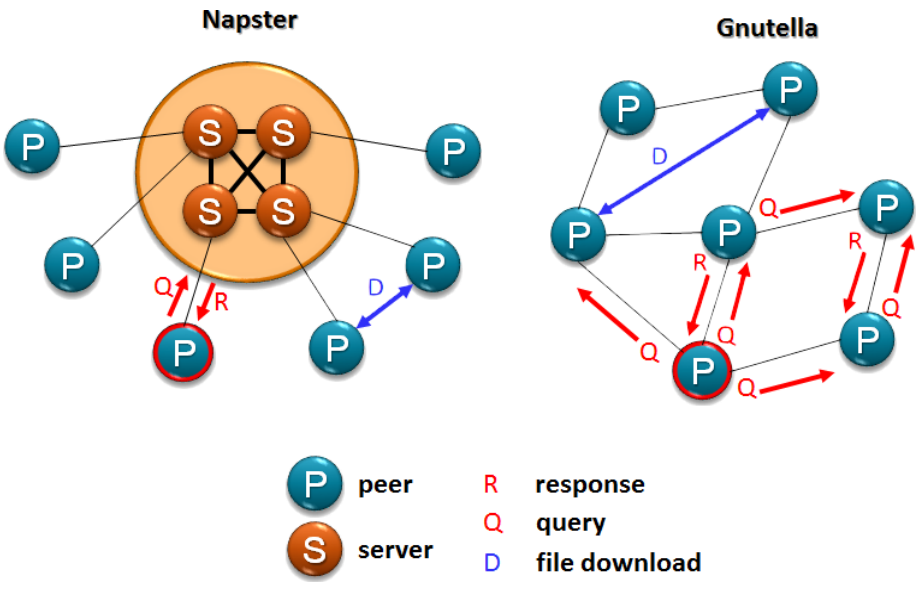
Sistemas e aplicações *Peer-to-peer* são sistemas distribuídos sem qualquer forma de controle centralizado ou hierarquia organizacional, onde o software que está sendo executado em cada nó é equivalente em funcionalidade. Tais sistemas também possuem muitos aspectos técnicos interessantes como, auto-organização, adaptabilidade e escalabilidade.

Embora a definição exata de *P2P* seja discutível, estes sistemas não possuem infraestrutura centralizada, dedicada, mas ao invés dependem da participação voluntária dos pares para contribuir com recursos com os quais a infraestrutura é construída.

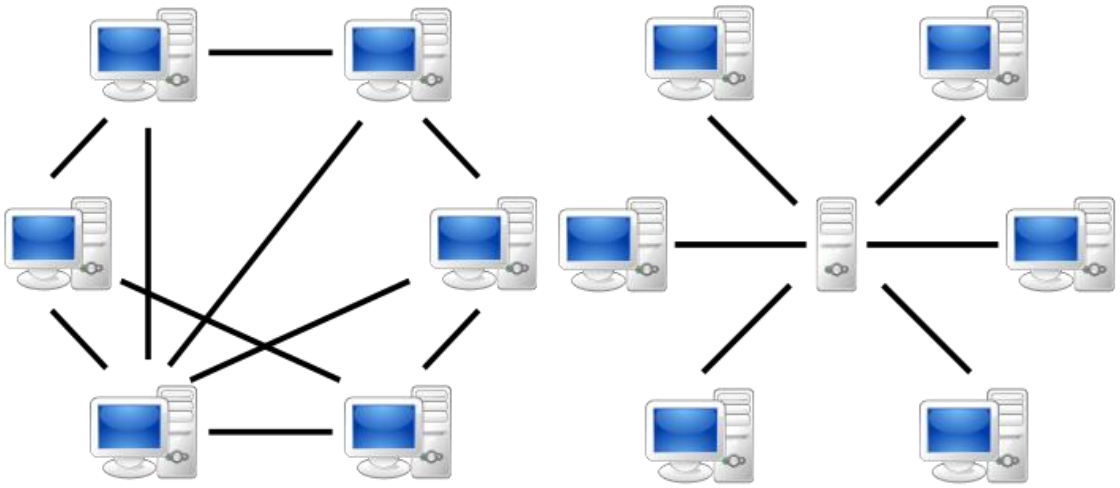
A estrutura de aplicação *P2P* foi popularizada por aplicativos de compartilhamento de arquivos, como o *Napster*. Este sistema era utilizado para a troca de arquivos, geralmente música, entre os seus usuários tornando-se muito utilizado e conhecido em pouco tempo. No ano de 2001 o *Napster* foi considerado a aplicação com maior crescimento na Web tendo sido transferido por cerca de 50 milhões de usuários. O mesmo foi desenvolvido por uma companhia privada de mesmo nome que foi processada pela indústria fonográfica devido ao fato de os usuários do sistema o utilizarem para a troca de músicas protegidas por direitos autorais sem a devida permissão. Para utilizar o Napster, os usuários se registravam no sistema identificando os arquivos disponibilizados pelos mesmos. Após isto, um outro usuário poderia consultar o servidor, ou cluster de servidores, para encontrar e transferir um arquivo anteriormente compartilhado.

Com o encerramento das operações do Napster, seus usuários e defensores começaram a utilizar um outro sistema, o *Gnutella*. Diferentemente do seu predecessor, este sistema não necessita de uma entidade central para a realização das buscas dos arquivos possibilitando a operação direta entre os usuários. Desta forma, a indústria da música viu-se impossibilitada de impedir o funcionamento do *Gnutella*, pois não se sabe quem são os indivíduos que estão trocando arquivos. Mesmo que se identifiquem alguns usuários que estejam infringindo a lei, a punição dos mesmos não irá impedir o funcionamento da rede e o restante dos usuários continuará a permutar músicas.

A figura ao lado ilustra a diferença básica entre o funcionamento do *Napster* e do *Gnutella*, enquanto o primeiro possui um servidor (mais precisamente um *cluster* de servidores) para responder os pedidos de busca, o último realiza as buscas de forma distribuída pela rede.



Com o aumento da capacidade dos computadores e dos links de Internet domésticos, outros usos de redes *P2P* existem, por exemplo para projetos de computação distribuída, para distribuição de fluxos de mídia em tempo real (*streaming*), telecomunicações (telefonia, videoconferência...) e outros.



Modelos de rede P2P e cliente-servidor

As redes P2P são geralmente construídas **sobre a camada de aplicação**, formando uma **rede sobreposta**. Essas redes sobrepostas são utilizadas para a descoberta de pares e indexação de conteúdo. Esse tipo de arquitetura confere uma vantagem às redes *P2P* em relação a outros modelos de distribuição de conteúdo: a rede *P2P* é abstrata (nível de aplicação), e por isso fica isolada da estrutura física da rede sobre a qual os protocolos *P2P* rodam.

Redes *P2P* podem ser classificadas como **puras** ou **híbridas**. As redes *P2P* puras não carregam a noção de cliente ou servidor, ou seja, não existem nós de infraestrutura, e cada par tem um status igual na estrutura, funcionando ao mesmo tempo como cliente e servidor. Redes híbridas permitem a existência de nós especiais, frequentemente chamados de *supernós* (*supernodes*). Esses nós tipicamente têm mais recursos (especialmente banda) disponíveis.

O controle de conexões e fluxo de informações numa rede *P2P* pode ser **distribuído** ou **centralizado**. No modelo distribuído, as informações de controle (indexação e descoberta de pares) circulam entre os pares, assim como os dados; enquanto no modelo centralizado, existe um nó central responsável pela indexação do conteúdo e pelo gerenciamento do processo de troca de dados, porém as trocas de dados são feitas diretamente entre os pares e não são regidas por nenhum algoritmo em particular.

Em um sistema *P2P*, cada participante é um nó da rede sobreposta, ou seja, existe um "enlace" entre cada dois nós que conhecem um ao outro: se um participante A conhece um participante B da rede, então um "enlace" de rede sobreposta é formado com direção de A para B. Baseado em como os enlaces são formados, pode-se classificar os sistemas *P2P* em **estruturados** ou **não-estruturados**.

Sistemas *P2P* **estruturados** são aqueles em que existe um protocolo que garante que cada nó possa rotear de forma eficiente uma busca por qualquer dado, mesmo que ele seja muito raro. Para tal, é necessário que os nós e, em alguns casos, também os recursos, sejam organizados segundo critérios e algoritmos específicos, o que leva a redes sobrepostas com propriedades especiais. A maioria das redes deste tipo utiliza tabelas *hash* distribuídas.

Em sistemas *P2P* **não-estruturados**, a distribuição dos nós na rede é feita de forma arbitrária. Redes não-estruturadas são construídas de forma mais simples, pois um nó que queira participar da rede pode começar simplesmente copiando alguns links de outros participantes, e então ir estabelecendo suas próprias conexões ao longo do tempo. Uma desvantagem das redes não-estruturadas em relação às redes estruturadas é que, as mensagens de busca de conteúdo devem inundar a rede, a fim de encontrar pares que estejam compartilhando os dados requisitados, o que ocasiona picos de tráfego de controle; além disso, as buscas podem às vezes não ser resolvidas, mesmo que existam na rede nós compartilhando os dados requisitados, com a probabilidade disso acontecer sendo maior à medida que esses dados sejam mais raros.

Uma tabela *hash* distribuída (*DHT, Distributed Hash Table*) funciona como uma tabela *hash* tradicional, no sentido de que associa a cada *valor* (ex. nó da rede) uma *chave* única. A diferença é que a tabela é, como o nome diz, distribuída entre os nós da rede sobreposta, ficando cada nó responsável por uma porção da tabela. Um algoritmo de *DHT* genérico implementa apenas uma função, busca (chave). Esta função retorna o identificador do nó responsável pela chave. DHTs são usadas para tornar as buscas na rede *P2P* mais eficientes do que nos sistemas não-estruturados.

Hoje, muitos dos programas *P2P* mais populares (*eMule*, *BitTorrent*, *Gnutella*) usam métodos baseados em *DHT* paralelamente a suas redes não-estruturadas originais para descoberta de pares, a fim de aumentar a eficiência da rede.

8.1. Classificando redes P2P

Conforme demonstrado por OLIVEIRA e ROCHA (2003), agora apresentaremos dois tipos de classificação para redes *P2P*.

8.1.1. Primeira Classificação

- Modelo Centralizado

No modelo centralizado, uma unidade central (Servidor ou conjunto de servidores) contém os identificadores de todos os participantes da rede *P2P*. Desta forma, estabelece-se um modelo cliente servidor entre cada *peer* (nó) da rede e o servidor *P2P*. Exemplo: *Napster*.

- Modelo Descentralizado

No modelo descentralizado não existe uma coordenação central (servidor). Neste modelo, rede *P2P* é dita completamente distribuída. Exemplos: *Gnutella*, *Freenet*.

- Modelo Hierárquico

Este modelo envolve, basicamente, a mistura dos modelos centralizado e descentralizado. Para tanto, são introduzidos na rede *P2P* os chamados *supernós* (*supernodes* ou *superpeers*). Exemplos: *KaZaA*, *Morpheus*.

8.1.2. Segunda Classificação

- *CIA* (*Centralized Indexing Architecture*)

Uma rede *peer-to-peer* com uma arquitetura *CIA* é aquela que contêm um servidor central ou um cluster de servidores que é responsável por responder os pedidos de busca e realizar todas as tarefas de manutenção da infraestrutura.

O exemplo mais comum de uma rede deste tipo é o sistema *Napster*. Em redes do tipo *CIA*, existe um ponto central que se estiver inoperante, a rede não funciona. Redes deste tipo não são verdadeiramente *peer-to-peer*, porque além de existir um ponto único de falha, apenas a transferência de arquivos é feita de forma distribuída. O mecanismo de buscas e a manutenção da infraestrutura são realizados de forma centralizada, conforme o paradigma cliente/servidor.

- *DIFA* (*Distributed Indexing with Flooding Architecture*)

Esta arquitetura, assim como também a *DIHA*, é caracterizada pela completa descentralização de seu funcionamento. Os mecanismos de busca e manutenção da infraestrutura estão distribuídos pela rede. Neste tipo de sistema, cada nó é responsável por manter a listagem dos seus próprios arquivos, e responder quando receber uma busca para um arquivo que seja uma resposta válida para a busca em questão. Para isto, é utilizado o mecanismo de "*flooding*" ou inundação. A rede deste tipo mais conhecida e estudada é a rede *Gnutella*.

As redes deste tipo possuem uma limitação muito grande que está ligado ao fato de seu mecanismo de busca utilizar o mecanismo de inundação. Para que uma rede não fique saturada com a repetição infinita do *flooding* de mensagens, estas mensagens possuem um número máximo de nós que a mesma pode atravessar. Isto possui uma séria implicação, mesmo que um arquivo exista no sistema e que o nó que o contenha esteja on-line, uma busca para este arquivo pode falhar. Contudo, embora completamente descentralizadas, estas redes possuem deficiências sérias na performance.

- *DIHA* (*Distributed Indexing with Hashing Architecture*)

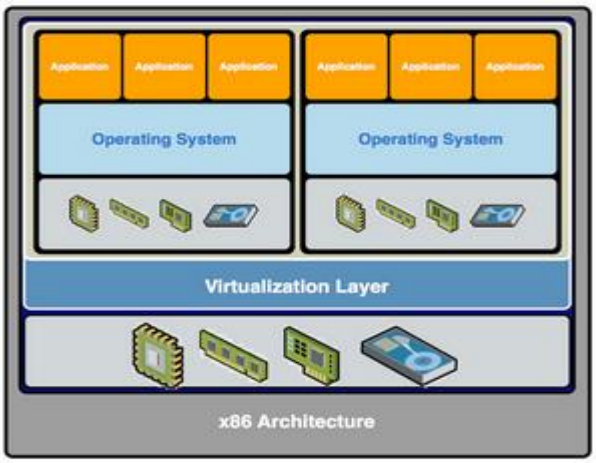
Esta arquitetura também possui uma característica totalmente descentralizada. A principal diferença entre as redes *DIFA* e *DIHA* está no mecanismo de busca. Nos sistemas com inundação, cada *peer* é responsável pelo espaço de índices relativo aos arquivos que ele próprio contêm. A arquitetura *DIHA* muda este conceito, cada nó é responsável por um subconjunto do espaço total de índices. Quando o nó entra na rede, este recebe um espaço do conjunto dos índices dos arquivos, ao sair da rede a mesma deverá designar estes índices para outro nó. As buscas não são difundidas na rede sem direção como no *flooding*, ao invés disto são direcionadas para o nó correto que é o responsável pelo respectivo índice dentro do espaço de índices. Os protocolos que implementam este tipo de arquitetura são bem mais complexos, existem poucos estudos sobre a utilização dos mesmos.

9. VIRTUALIZAÇÃO

As máquinas virtuais (*Virtual Machines – VM*) foram idealizadas e introduzidas nas décadas de 50 e 60, com a finalidade de permitir o *time-sharing*[2] de equipamentos que eram muito caros e ficavam, por vezes, ociosos por muito tempo. A principal proposta do compartilhamento de hardware era prover a máxima utilização dos equipamentos Mainframe da IBM de forma segura, conseguindo assim aperfeiçoar o uso do hardware entre vários usuários.

Virtualização é uma tecnologia que faz um computador físico funcionar como se fosse mais computadores onde cada computador "virtualizado" possui a mesma arquitetura básica como se fosse um computador físico. Existem várias maneiras de se fazer isso, e cada um tem os seus prós e contras (MARSHALL, 2006).

É a simulação de uma plataforma de hardware, sistema operacional, dispositivo de armazenamento ou recursos de rede. Consiste em uma técnica de separar aplicação e sistema operacional dos componentes físicos.



Por exemplo, uma máquina virtual possui aplicação e sistema operacional como um servidor físico, mas estes não estão vinculados ao software e pode ser disponibilizado onde for mais conveniente.

Uma aplicação deve ser executada em um sistema operacional em um determinado software. Com virtualização de aplicação ou apresentação, estas aplicações podem rodar em um servidor ou ambiente centralizado e ser deportada para outros sistemas operacionais e hardwares.

Cada vez mais empresas estão buscando formas de **reduzir os custos e complexidade com o ambiente de TI**. A virtualização se tornou componente chave para o desenvolvimento de uma estratégia eficiente na busca destes objetivos.

Dentre os **desafios enfrentados nos datacenters** podemos destacar:

- Datacenters atingiram a capacidade máxima;
- Servidores subutilizados;
- Gerenciamento e Segurança complexa dos servidores;
- Problemas de compatibilidade de aplicações.

De acordo com MOREIRA (2006) o problema de se utilizar um servidor para rodar cada serviço é que ele aproveita mal os recursos das máquinas, em média, os servidores utilizam somente de 5% a 10% da sua capacidade. Com o objetivo de reduzir os custos de administração e manutenção e centralizar o trabalho dos gerentes de tecnologia, as empresas apostaram em um novo conceito: utilizar equipamentos mais robustos, com mais recursos de processamento e espaço em disco, para hospedar as diversas aplicações da companhia, prática batizada de consolidação de servidores.

[2] Capacidade de múltiplos usuários compartilharem recursos computacionais de uma única origem.

Para MARSHALL (2006), virtualização é um conceito que permite dividir ou partilhar os recursos de um computador por vários ambientes simultaneamente. Esses ambientes podem interoperar mesmo sem conhecer uns aos outros. Um único ambiente pode, ou não, saber o que está sendo executado em um ambiente virtual. Esses ambientes são mais comumente conhecidos como máquinas virtuais (VMs). VM será quase sempre um local onde está instalado um sistema operacional (por exemplo, Linux, Windows etc.) e são conhecidos como sistemas operacionais do cliente. Instruções para uma VM é geralmente transmitida diretamente para o hardware físico permitindo que o ambiente funcione mais rápido e eficiente do que uma emulação[3].

A criação de máquinas virtuais é o que possibilita que diversos Sistemas Operacionais rodem simultaneamente em um mesmo equipamento físico. Uma máquina virtual é "virtualmente" igual a uma máquina física, ou seja, ela tem BIOS, processo de boot, dispositivos, tais como, discos, placas de rede, memória, placas de vídeo e assim por diante.

Para que seja possível criar uma máquina virtual, precisamos de um software que proporcione uma camada de virtualização, também conhecido como ***hypervisor***. Ele é um componente crítico para a implantação de um projeto de virtualização, independente da tecnologia utilizada, são também conhecidos como ferramentas de gerenciamento. O *hypervisor* gerencia o ambiente virtual devem gerenciar tanto o ambiente físico como o virtual, assim como sistema operacional e aplicações. São distintas para cada tipo de virtualização que se deseja aplicar.

Existem basicamente dois tipos hoje no mercado. Aqueles que você instala sobre um sistema operacional (Windows ou Linux), e aqueles que possuem um *kernel* próprio e são instalados diretamente no hardware (*bare-metal*).

Os principais aplicativos de virtualização que são instalados sobre um SO são o *VMWare Server*, O *Microsoft Virtual Server* e o *Citrix XenServer Express*. Todos são *free*. Como sistemas de virtualização *bare-metal* existem o *VMWare ESX Server* (*VM-Ware Infrastructure*), *Citrix XenServer Enteprise Edition* e o *Oracle VM*.

9.1. Tipos de Virtualização

Quando se pensa ou se ouve falar em virtualização, logo pode nos vir a cabeça um computador com vários "computadores" com sistemas operacionais rodando neles, porém se pensamos desta forma, nos remetemos a apenas um dos tipos existentes de virtualização. A seguir veremos diversas variações deste conceito.

- **Virtualização de servidor:** Técnica de execução de um ou mais servidores virtuais sobre um servidor físico; permite maior densidade de utilização de recursos (hardware, espaço etc.), enquanto permite que isolamento e segurança sejam mantidos.
- **Virtualização de aplicação:** Permite executar aplicações em um ambiente virtualizado no desktop do usuário, isolando a aplicação do Sistema Operacional; isso é possível através do encapsulamento da aplicação no ambiente virtual - quando a solução completa de virtualização de aplicações é implantada, é possível distribuir aplicações de um servidor central.
- **Virtualização de desktop:** Consiste na execução de múltiplos sistemas operacionais em uma única workstation e permitindo que uma aplicação de linha de negócio seja executada em um sistema operacional não compatível.
- **Virtualização de apresentação:** Permite executar e manter o armazenamento das aplicações em servidores centralizados, enquanto provê uma interface familiar para o usuário em sua estação.
- **Virtualização de perfil:** Os usuários podem ter os seus documentos e perfil separados de uma máquina específica, o que permite a fácil movimentação do usuário para novas estações em caso de roubo ou quebra de equipamento.

[3] Conceito que permite um ambiente agir como se fosse outro ambiente. As instruções são interpretadas a partir do ambiente onde estão sendo executadas as instruções reais. A emulação tem baixo desempenho quando comparado à virtualização devido à sobrecarga do interpretador.

9.1.1. Virtualização de Servidores

Com o aumento da quantidade de informações das empresas também há um aumento na quantidade de servidores e por consequência aumento de custos com gerenciamento, energia elétrica e aumento de complexidade do ambiente.

Para resolver esses problemas a virtualização de servidores oferece uma otimização da infraestrutura de TI. Esta otimização leva à consolidação e contenção de servidores. Com a criação de uma infraestrutura virtual, podemos colocar "N" servidores virtuais em um mesmo servidor físico, aumentando a eficiência energética destes equipamentos e diminuindo a complexidade do ambiente. Este "N", em ambientes de produção, chega facilmente a 15 ou 20 (FARIAS apud MODA, 2009).

Algumas vantagens na redução de 15 ou mais vezes o número de servidores da sua empresa acarreta em: Redução do espaço físico necessário para armazená-los; Redução do consumo de energia dos equipamentos; Redução da dissipação de calor e consequentemente da necessidade de refrigeração; Redução das conexões de cabos de rede. Menos cabos, significa menor número de portas de switch necessárias; Redução de tomadas e cabos de energia; Redução de *HBAs* e *Switches Fiber Channer* para acesso ao *storage*.

Com isso tudo, reduzimos a complexidade do ambiente, e indiretamente centralizamos o gerenciamento. (SCHÄFFER apud MODA, 2009)

Segundo o mesmo autor, com a contenção, significa que se precisar de um novo servidor, tudo que precisa fazer é criar uma nova máquina virtual, o que reduz significativamente o tempo de provisionamento de novos servidores. As máquinas virtuais são arquivos, e é muito simples a criação de máquinas modelos que se tornam novos servidores em minutos. Esses novos servidores não significarão gastos com aquisição de equipamentos físicos e nem com o aumento da complexidade. Quando um servidor não aguentar mais a adição de novas máquinas virtuais basta adicionar mais um novo servidor físico à sua rede.

Ainda, conforme o mesmo autor indica, a virtualização de servidores torna as máquinas virtuais independentes do hardware físico, promovendo a facilidade em promover a continuidade de negócios e o baixo tempo de *disaster recovery*.

9.1.2. Virtualização de Desktops

O conceito de virtualização de desktops é o mesmo empregado na virtualização de servidores, ou seja, executar diversos sistemas operacionais em um único equipamento físico, onde cada usuário possui um sistema operacional próprio, como se estivesse usando um desktop normal. Este conceito elimina qualquer trauma de migração e possui uma série de benefícios: Gerenciamento centralizado; Instalações simplificadas; Facilidade para a execução de backups; Suporte e manutenção simplificados; Acesso controlado a dados sensíveis e à propriedade intelectual mantendo-os seguros dentro do Data Center da empresa; Independência do hardware; Disponibilização de novos desktops reduzida para alguns minutos; Migração de desktops para novo hardware de forma transparente; Maior disponibilidade e mais fácil recuperação de desktops; Compatibilidade total com as aplicações.

Dessa forma as empresas podem centralizar todos os desktops, mesmo os de unidades remotas, dentro do seu Data Center.

9.1.3. Virtualização de Aplicações

Segundo VIRTUE IT apud MODA (2009) a virtualização de aplicação é a habilidade de poder instalar e usar qualquer aplicação, enquanto protege o sistema operacional e outras aplicações de modificações que poderiam afetar a estabilidade e segurança do sistema, ou seja, é tornar a aplicação independente dos componentes do sistema operacional. Em outras palavras, a virtualização de aplicações transforma um programa em um arquivo executável sem necessidade de utilizar centenas de arquivos, chaves de registro e *DLLs*.

9.2. Vantagens e desafios da Virtualização

Os benefícios e economia propiciados pela utilização de virtualização de servidores é evidente, tornando o Data Center mais dinâmico, eficiente e flexível. A perspectiva de aumento

da utilização desta tecnologia tem feito as empresas olharem superficialmente sobre a tecnologia, que, como toda nova tecnologia, tem seus percalços.

9.2.1. Vantagens da adoção de tecnologia de virtualização

Segundo MARSHALL (2006) os benefícios são inúmeros e destaca as seguintes vantagens:

- **Portabilidade**: Capacidade de ter uma plataforma de hardware consistente, mesmo que o hardware seja de diferentes fabricantes;
- **Gerenciamento**: Ambientes virtuais podem ser gerenciados facilmente e oferecem acesso ao hardware virtual.
- **Eficiência**: Quando implementado corretamente, permite que o servidor de virtualização de hardware físico seja usado mais eficientemente, permitindo maior utilização dos recursos de hardware.

9.2.2. Desafios da virtualização em data centers

A Network World apud MODA (2009) aponta os oito principais desafios da virtualização em data centers. Através de pesquisas com profissionais, analistas e fornecedores de TI, foi elaborada uma lista com obstáculos que o usuário poderá enfrentar para implementar ambientes virtuais.

- **Abdicando do físico:** Necessidade de uma dedicação maior e análise atenta de quais recursos físicos serão necessários à carga de trabalho.
- **Performance de aplicativos abaixo da média:** Muitos aplicativos ainda não foram ajustados para ambientes virtuais.
- **Segurança falha:** Quando você implementa um ambiente virtual, elimina o vínculo entre hardware e software, o que pode gerar confusão na hora de proteger sua infraestrutura. Isso torna o processo de segurança mais complexo.
- **Aprisionamento:** É necessário a adoção de uma maneira-padrão de criar e gerenciar máquinas virtuais afim de não ficarem presas à um único fornecedor.
- **Acúmulo de máquinas virtuais:** Devido à facilidade de implementação de máquinas virtuais isso pode gerar um acúmulo de VM's ociosas.
- **Custos de licenciamento:** Do mesmo modo que empresas discutem o preço de licenças de software com múltiplos processadores, elas também têm surpresas com licenças em ambientes virtuais.
- **Paixão por armazenamento:** É fácil esquecer do impacto que a arquitetura mais centrada de recursos virtuais pode ter. O *storage*, por exemplo, deve ser examinado com atenção, já que, em muitos casos, os recursos virtuais vão acessar uma *storage área network* (SAN) compartilhada.
- **Barreiras virtuais:** É possível que migre um aplicativo que está sendo executado de uma máquina física para outra, desde que os processadores nestas máquinas sejam iguais, devido à diferença de arquiteturas de processador.

9.2.3. Segurança em Virtualização

Devido aos benefícios e facilidades da virtualização, esta tecnologia vem sendo utilizada como proposta para suprir outras necessidades e resolver outros tipos de problemas, como a segurança, por exemplo. A segurança é um dos campos onde a virtualização pode ser aplicada, seja para isolar aplicações instáveis ou comprometidas, seja como solução de rápida recuperação de desastres, seja como ferramenta no auxílio de análises forenses ou até mesmo como uma solução de baixo custo para detecção de intrusão.

FARIAS apud MODA (2009) afirma que devido esse novo campo que a virtualização vem sendo empregada, tornou-se cada vez mais necessário que softwares que fazem o compartilhamento do hardware físico fossem avaliados por uma entidade idônea. Esta avaliação ocorre para verificação do nível de segurança da estrutura do software testado. Basicamente isso ocorreu por dois motivos: o primeiro é pela concentração do risco, ou seja, várias máquinas virtuais sendo executadas sobre um mesmo hardware; já o segundo, é que algumas empresas

multinacionais e o governo de alguns países somente realizam a aquisição de produtos que sejam avaliados por essas entidades, como por exemplo, a *National Security Agency* (NSA), a qual faz parte do *U.S. Department of Defense* (Departamento de Defesa dos Estados Unidos).

Segurança da Informação é a preservação da confidencialidade, integridade e disponibilidade das informações.

Independentemente do ambiente computacional ser convencional ou virtualizado é preciso atender à esses três princípios básicos de segurança:

- **Confidencialidade:** Garantir que as informações sejam acessíveis apenas para aqueles que estão autorizados a acessá-las.
- **Integridade:** Salvaguardar a exatidão e a inteireza das informações e métodos de processamento.
- **Disponibilidade:** Assegurar que os usuários autorizados tenham acesso às informações e aos ativos associados quando necessário.

Ao fazer-se uso de toda a infraestrutura virtual que a *VMware* disponibiliza, torna-se possível a utilização de quatro ferramentas que ajudam a garantir, por exemplo, alta disponibilidade, rápida recuperação, balanceamento de carga e integridade ao ambiente virtual (FARIAS apud MODA, 2009).

- ***VMotion*:** A tecnologia *VMotion* possibilita a migração online das máquinas virtuais em execução entre servidores *VMware ESX*.
- ***High Availability*:** O *VMWare High Availability* provê uma solução simples a baixo custo de alta-disponibilidade para as aplicações que são executadas pelas máquinas virtuais.
- **Balanceamento de Carga (DRS):** O *VMware Distribuited Resource Scheduler* dinamicamente realoca e faz o balanceamento de carga entre os servidores *VMware ESX* que formam o conjunto de recursos lógicos.
- ***Backup* do Ambiente Virtual:** Funciona como um centralizador de backup das máquinas virtuais. São drives e scripts que fazem realizam o backup das máquinas virtuais que são executados nos servidores *VMWare ESX* (FARIAS, 2007).

9.3. Virtualização e TI Verde: Uma visão para o futuro

Recente relatório divulgado pela *Global Action Plan*, entidade britânica para a conservação do ambiente, os servidores computacionais são uma grande ameaça para o aquecimento global. Segundo os estudos, a grande maioria dos responsáveis pelas áreas de TI das empresas não detém este tipo de informação. Metade dos profissionais pesquisados acreditam que provocam um "significante" impacto ambiental, entretanto, 86% desconhecem qual o grau de emissão de carbono gerado pelas suas atividades. Este impacto ambiental provocado pelos servidores é real e requer muita atenção. Segundo os especialistas, o péssimo dimensionamento das necessidades computacionais é um dos principais fatores que contribuem para o crescimento descontrolado de servidores. Adquirir novos equipamentos significa aumentar o consumo de energia, tanto para o seu funcionamento quanto para a sua refrigeração.

9.4. Benefícios da Virtualização de Servidores

Existem benefícios reais e mensuráveis para a implantação em Datacenters, mesmo em ambientes de pequenas e médias empresas.

O melhor de tudo é que os benefícios são palpáveis para o negócio, tanto em redução de custos quanto em agilidade da equipe de TI.

Seguem os principais benefícios que a virtualização proporciona:

- **Menor Aquecimento / Economia de Energia**

Servidores consomem Energia Elétrica, e o poder de processamento de servidores modernos estão além da necessidade de muitas aplicações. O uso da virtualização permite múltiplas aplicações rodarem no mesmo servidor, isoladas entre si, o que traz redução do espaço utilizado e da energia elétrica para manter o servidor ligado e refrigerar o ambiente.

- **Redução de Custos de Aquisição e Redução do Espaço Físico**

Em empresas que precisam de mais de 1 servidor para executar seus sistemas, a virtualização traz redução de custos de aquisição da solução, pois um mesmo servidor pode ser utilizado para mais de um sistema, e mesmo que precise comprar um servidor com mais capacidade de processamento, memória e disco para dar conta de diversos sistemas, o custo extra desses recursos é bem menor comparado a adquirir um novo servidor para cada sistema.

Para empresas pequenas, não é necessário investir em licenças de software extras para virtualização, os principais fornecedores (VMware e Microsoft) disponibilizam licenças gratuitas de seus produtos (restrições se aplicam).

- **Green IT / TI Sustentável**

Além de reduzir os custos de aquisição e consumo de energia, a virtualização impacta diretamente no meio ambiente. No final da vida útil, existirão menos servidores para serem descartados, resultando em menos materiais tóxicos no meio ambiente.

Em um ambiente virtual é possível configurar para que alguns *Hosts* sejam desligados durante o período de menor demanda (fins de semana e a noite por exemplo). Um exemplo dessa tecnologia é o *DPM* (*Distributed Power Management*) da *VMware*. Além da economia de energia direta dos servidores, diminui também a necessidade de refrigeração do ambiente.

- **Menor tempo de parada em manutenções programadas**

Em um ambiente virtualizado é possível migrar as aplicações entre servidores físicos on-line, sem desligar ou interromper o serviço, chamado de *vMotion* (*VMware*), *XenMotion* (*Citrix*) ou *Live Migration* (*Microsoft*). Isso permite a execução de tarefas rotineiras, como *upgrade* de *firmware* de servidores por exemplo, em horário comercial, sem parar os sistemas e sem pagar horas extras.

Outras manutenções mais complexas podem se beneficiar também, é possível "suspender" uma máquina virtual pouco antes de uma parada do *hardware*, e quando o mesmo voltar, basta retomar a máquina virtual, ela carregará do ponto onde estava, com os sistemas executando.

Mesmo em um upgrade de ambiente, é possível migrar as máquinas virtuais entre servidores velhos e novos, diminuindo o tempo de reinstalação e configuração da solução.

Fazendo o upgrade do firmware de servidores, também não é necessário realizar o upgrade de drivers nas máquinas virtuais, já que estas não enxergam a camada de hardware diretamente.

- **Rápida recuperação de falhas, backup otimizado e Recuperação de Desastres**

Quando um servidor físico apresenta problemas, o tempo para voltar em produção depende muito de estratégias de *backup* e ferramentas, é necessária uma ferramenta para imagem do sistema operacional, outra para *backup*, outra para aplicar atualização de patches e drivers, para só depois restaurar o *backup*.

Um *cluster* de servidores virtuais traz por padrão a proteção contra a falha física de servidores, as máquinas virtuais são protegidas por um recurso chamado *High Avaliability* (*HA*), onde, na falha do servidor que está executando, a mesma é ligada em um outro servidor automaticamente, e em poucos minutos o sistema está recuperado automaticamente.

Também com uma máquina virtual é possível utilizar ferramentas com o *Veeam Backup* ou o *VMware VDP* para gravar a imagem da máquina virtual, efetuar o backup e até replicar a máquina virtual para outro *Datacenter*. Em caso de falhas, é possível restaurar a máquina virtual rapidamente em outro servidor físico, sem se preocupar com drivers e outros detalhes, provendo uma recuperação muito rápida do ambiente.

- **Ambientes de Testes**

Com um ambiente virtual é muito simples clonar um conjunto de máquinas virtuais, colocar em um ambiente separado (uma *VLAN* diferente por exemplo) e executar testes, tanto de atualizações do sistema operacional, quanto atualizações e modificações dos sistemas de produção, como ERP e CRM. Também é possível utilizar o recurso de *Snapshots* para upgrades controlados, por exemplo, em uma atualização de versão de um sistema ERP, pode-se criar um *Snapshot* pouco antes do serviço, executar a atualização, caso detecte algum problema logo após a atualização, basta voltar o *Snapshot* de antes da atualização para voltar a versão estável em produção, uma operação de poucos segundos, e não de horas como seria voltar um *backup*.

- **Provisionamento rápido de novos aplicativos e servidores**

A virtualização é ideal para atender o crescimento rápido dos negócios, pois possibilita o provisionamento de um servidor em poucos minutos, ao invés de vários dias como seria o processo com servidores físicos.

Antes da virtualização, o processo para provisionar um novo servidor consistia em cotações, compra, entrega, instalação física, alocação de porta de rede, energizar, instalar o sistema operacional e drivers, para só então começar a trabalhar de verdade.

Depois da virtualização, basta escolher um *template* com o sistema operacional correto e fazer o *deploy* do mesmo, em poucos cliques o servidor estará instalado e configurado na rede, pronto para instalar as aplicações. Esse tempo é mais reduzido ainda para instalar as aplicações se o fabricante entregar um *Appliance Virtual*.

- **Independência de fornecedor**

Outra das vantagens da virtualização é a abstração total do hardware, assim, a máquina virtual não fica presa a um tipo de processador, controladora de disco, placa de rede ou qualquer outro dispositivo, permitindo migra-la de hardware muito facilmente.

Com isso, quando for executar a próxima compra de servidores e *storage*, é possível confrontar os preços muito facilmente, sem se preocupar com os custos de migração de plataforma e reinstalação do ambiente, basta movimentar as máquinas virtuais.

- **Isolamento de Serviços e Aplicações**

Muitas vezes para economizar servidores, vários serviços são consolidados no mesmo servidor. É comum por exemplo, em pequenas empresas, o servidor de *AD* (*Active Directory*) ser o mesmo servidor de Arquivos, ou o servidor Linux que fornece Internet também hospedar a homepage da empresa e outros sistemas *Web*.

O problema disso é que algumas aplicações que executam bem em conjunto quando instaladas, podem conflitar depois de um tempo com as atualizações, com requisitos de bibliotecas, versões de bancos de dados e drivers específicos, causando problemas. Também não tem como priorizar recursos para aplicativos no mesmo servidor.

Com a virtualização, é possível ter a mesma economia de servidores, mas cada aplicação fica isolada, com seu sistema operacional, bibliotecas e drivers independentes, e é possível dar maior ou menor prioridade para cada uma.

- **Manter sistema legado**

É comum encontrar algum sistema legado, conectado em um computador velho em um canto do *Datacenter*/CPD. Essa aplicação normalmente não tem mais suporte do fabricante (por falta de contrato ou por ter sido descontinuada mesmo), mas é importante estar ali para fins de conformidade e auditoria.

Virtualizando essa aplicação é possível mitigar os principais riscos: quebra do *hardware* físico (e falta de peças de reposição), corrompimento do sistema de arquivos, *backup*, disponibilidade, ou seja, a aplicação recebe todo o benefício do ambiente virtual.

- **Automatização de processos e contabilização de recursos**

Em implementações mais avançadas, é possível automatizar tarefas da TI para agilidade nos negócios.

O recurso mais simples disponível é o *Template* de máquinas virtuais, onde algumas máquinas virtuais modelos são criadas, com sistema operacional instalado e boas práticas já aplicadas, para rápido provisionamento pela equipe de TI.

Outro recurso bem conhecido é o *DRS* da *VMware* (*Distributed Resource Scheduler*), que migra as máquinas virtuais de acordo com a necessidade de processamento e memória das mesmas. Esse recurso em conjunto com o *DPM* pode ser usado para economia de energia, como tratado no tópico acima sobre *Green IT*.

Mas existem níveis de automatização bem maiores que podem ser conseguidos, é possível, por exemplo, determinar que uma máquina de teste seja excluída automaticamente se ficar 30 dias sem ser utilizada.

Também é possível criar um portal *self-service* para os usuários avançados, por exemplo: para que desenvolvedores criem os ambientes de testes que precisam sem intervenção da equipe de operações.

Outra funcionalidade forte é a contabilização dos recursos. Com a tendência de isolamento das aplicações, é possível indicar quais VMs pertencem a qual departamento, e contabilizar quantos recursos estão sendo usados por cada departamento, permitindo a TI indicar os custos por área de negócio.

- **Migração para a Computação em Nuvem facilitada**

Muitos dos provedores de nuvem suportam *IaaS* (*Infrastructure as a Service*), assim, uma máquina virtual pode ser facilmente migrada da estrutura interna para a nuvem sem maiores dificuldades.

Virtualizar seu ambiente atual é o primeiro passo rumo à computação em nuvem. É uma das formas da TI estar mais ligada ao negócio, e entregar mais funcionalidades com menos recursos.

10. CLOUD COMPUTING

Cloud Computing ou Computação em Nuvens é a virtualização de produtos e serviços computacionais, ou seja, é uma maneira de armazenar todas as informações em servidores virtuais chamados de "nuvem", onde há uma tendência mundial para este modelo não necessitando de máquinas velozes com um grande potencial de hardware e sim de um simples computador conectado à internet para rodar todos os aplicativos.

Hoje, há grandes empresas investindo nesta tecnologia para oferecerem esses serviços a seus clientes, como a gigante *Google* e a *Microsoft*. Sem percebermos, já estamos fazendo parte dessa história, basta ter um *Gmail* ou uma conta no *Outlook*, onde "guardam-se" dados nesses servidores virtuais, uma mensagem, e-mail, fotos, etc., você já está usufruindo da grande nuvem. São parques computacionais armazenando todas as informações particulares de usuários no mundo, sendo esses produtos e serviços disponibilizados gratuitamente em sua maioria. Na realidade, uma simples troca está sendo feita, na qual dispõem-se de serviços e tem-se espaço para guardar informações. Em contrapartida, são coletados os dados do usuário para um futuro uso comercial.

Destacam-se as seguintes características da Computação nas Nuvens:

- Acesso às aplicações independente de sistema operacional ou hardware;
- O usuário não precisará se preocupar com a estrutura para execução da aplicação: hardware, backup, controle de segurança, manutenção, entre outros, ficam a cargo do fornecedor de serviço;
- Compartilhamento de dados e trabalho colaborativo se tornam mais fáceis, uma vez que todos os usuários acessam as aplicações e os dados do mesmo lugar;
- Dependendo do fornecedor, o usuário pode contar com alta disponibilidade, já que, se por exemplo, um servidor parar de funcionar, os demais que fazem parte da estrutura continuam a oferecer o serviço.
- *Self-service* sob demanda. O usuário pode adquirir unilateralmente recurso computacional, como tempo de processamento no servidor ou armazenamento na rede na medida em que necessite e sem precisar de interação humana com os provedores de cada serviço.
- Amplo acesso. Os recursos são disponibilizados por meio da rede e acessados através de mecanismos padronizados que possibilitam o uso por plataformas *thin* ou *thin client*, tais como celulares, *laptops* e outros;
- Serviço medido. Sistemas em nuvem automaticamente controlam e otimizam o uso de recursos por meio de uma capacidade de medição. A automação é realizada em algum nível de abstração apropriado para o tipo de serviço, tais como armazenamento, processamento, largura de banda e contas de usuário ativas.

10.1. Modelos de Serviços

Em ambientes de Computação nas Nuvens, podem-se ter três modelos de serviços. Estes modelos são muito importantes, pois definem um padrão arquitetural para as soluções de *Cloud*.

10.1.1. Software como Serviço (SaaS)

O *SaaS*, pode ser definido, de acordo com a *MSDN* (*Microsoft Developer Network*), como um software implantado como serviço hospedado, acessado através da Internet. Um mesmo software pode

ser utilizado por múltiplos usuários, sejam pessoas ou empresas. Esse tipo de serviço é executado e disponibilizado por provedores em servidores de responsabilidade de uma empresa desenvolvedora, ou seja, o software é desenvolvido por uma empresa que ao invés de vendê-lo ou usá-lo para benefício exclusivo, disponibiliza-o a um custo baixo a uma grande quantidade de usuários.

10.1.2. Plataforma como Serviço (PaaS)

Esse tipo de serviço disponibiliza servidores virtualizados nos quais os utilizadores podem executar aplicações existentes ou desenvolver novas aplicações, sem ter que se preocupar com a manutenção dos sistemas operacionais, servidores, balanceamento de cargas ou capacidade de computação. O objetivo do *PaaS* é facilitar o desenvolvimento de aplicações destinadas aos usuários de uma nuvem, criando uma plataforma que agiliza esse processo.

10.1.3. Infraestrutura como Serviço (IaaS)

Disponibiliza *grids* ou *clusters* ou servidores virtualizados, redes, armazenamento e software de sistemas desenhados para aumentar ou substituir as funções de um centro de dados.

Os serviços de infraestrutura abordam o problema de equipar de forma apropriada os datacenters, assegurando o poder de computação quando necessário. Além disso, devido ao fato das técnicas de virtualização serem comumente empregados nessa camada, economias de custos decorrentes da utilização mais eficiente de recursos podem ser percebidas.

11. CLUSTER E GRID

Algumas áreas de conhecimento (por exemplo, astronomia, meteorologia e genética) requerem, para os problemas estudados, de muitos recursos computacionais com alto desempenho para suprir cálculos complexos e repetitivos. Os avanços da tecnologia de computadores geralmente não acompanham a demanda solicitada e, às vezes, a utilização de supercomputadores é inviável financeiramente. Uma alternativa a ser adotada pode ser a soma dos recursos computacionais já existentes utilizando-os de forma mais apropriada e equilibrada, resultando em um ganho substancial de desempenho (*speedup*). Neste contexto, podem ser aplicados os paradigmas de cluster e grid computacional que melhor usufruem, respectivamente, dos recursos e serviços de maneira local e geograficamente distribuída.

Os ambientes de alto desempenho, vistos tanto nos clusters como em grids, são alcançados através de uma melhor taxa na execução de aplicativos e no número de dados na aplicação se comparados à execução em uma máquina local. Para obter este cenário, três abordagens são consideradas: aumento na velocidade do processador, algoritmos mais otimizados e ambientes de computação paralela e distribuída. Em ambos os paradigmas, estes esforços são somados garantindo a baixa latência de comunicação (na ordem de centenas de μs) e a elevada taxa de transmissão (a partir de dezenas de Mbps até, atualmente, dezenas de Gbps).

Um cluster computacional é um ambiente de computação paralela formado por um conjunto de computadores, chamados nós, interligados, muitas vezes, por dispositivos do tipo *switches* em uma rede LAN de alto desempenho (exemplos são a *Myrinet* e *ATM*). Os nós cooperam entre si para atingir um determinado objetivo comum. A arquitetura de um cluster é classificada, segundo Tanenbaum, como MIMD do tipo fracamente acoplado, ou seja, tem memória distribuída (multicomputadores), por isto os nós devem se comunicar a fim de coordenar e organizar todas as ações a serem tomadas. Deste modo, externamente o cluster é visto como sendo um único sistema.

Por outro lado, uma configuração de grid computacional é um ambiente computacional geograficamente distribuído que permite a interoperabilidade entre organizações a ele associadas. Seu principal objetivo é compartilhar e agregar recursos de forma a disponibilizá-los como serviços. Dentre outros, os serviços oferecidos podem ser a alocação de recursos, o gerenciamento de processos, a comunicação, a autenticação e a segurança. Os recursos que compõem um grid sejam eles físicos, virtuais ou humanos, são interligados por redes *WAN* de alta velocidade (via cabo ou *wireless*). Por conseguinte, um grid pode ser considerado como um ambiente de alto desempenho.

Configuração	*Cluster*	*Grid*
Domínio	Único	Múltiplos
Nós	Milhares	Milhões
Segurança do processamento e do recurso	Desnecessária	Necessária
Custo	Alto, pertencente a um único domínio	Alto, todavia, dividido entre domínios
Granularidade do problema	Grande	Muito grande
Sistema Operacional	Homogêneo	Heterogêneo

Diferenças entre as configurações de *cluster* e *grid* (COLVERO e CUNHA, 2012)

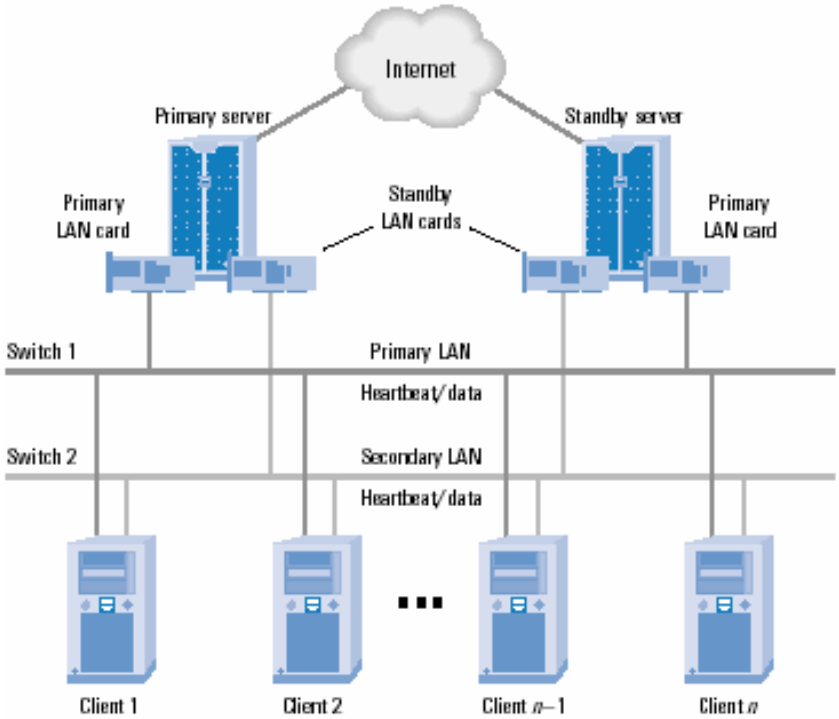
11.1. Clusters de Computadores

Para DANTAS apud PRADO (2005, p. 32) um cluster pode ser entendido como uma agregação de computadores de uma forma dedicada (ou não) para a execução de aplicações específicas de uma organização ou de uma instituição.

A utilização de dois ou mais computadores em conjunto para resolver certo problema denomina-se cluster, que em português significa agrupamento.

O termo cluster, ou aglomerado de computadores, pode ser formado por um conjunto de computadores convencionais agrupados fisicamente em um ambiente, ou simplesmente por computadores dedicados.

O mesmo autor aponta que frente aos altos custos de máquinas do tipo MPP e SMP e de uma maior oferta de computadores pessoais existentes nas redes das instituições, tem se adotado o aglomerado de computadores para a utilização em ambientes que requerem processamento de alto desempenho.



Exemplo de uma arquitetura *cluster*

Basicamente existem três tipos de clusters: Balanceamento de carga (*Load Balancing*), Alta disponibilidade (*High Availability*) e Alto desempenho (*High Performance Computing*).

11.1.1. Cluster Balanceamento de Carga ou Horizontal Scaling (HS)

O *cluster* de balanceamento de carga (*Load Balancing* ou *Horizontal Scaling – HS*) tem por objetivo distribuir as requisições que chegam ao *cluster*. Mesmo no caso de falha em um dos nós, estas requisições são redistribuídas entre os outros membros do *cluster*.

Este tipo de *cluster* tem como propósito a distribuição igualitária de processos ao longo dos nós do agrupamento de computadores, com a ajuda de algoritmos de escalonamento.

PITANGA apud PRADO (2010) destaca a importância de algoritmos para balanceamento e enumera três exemplos destes algoritmos:

- **Least Connections**: Esta técnica redireciona as requisições para o servidor baseado no menor número de requisições/conexões.

- **Round Robin**: Este método usa a técnica de sempre direcionar as requisições para o próximo servidor disponível de uma forma circular.

- **Weighted Fair**: Esta técnica dirige os pedidos para os servidores baseados na carga de requisições de cada um e na capacidade de resposta dos mesmos (performance).

Este tipo de cluster é muito utilizado para serviços web e de comércio eletrônico.

Também é comum a associação do *cluster* de balanceamento de carga com o de alta disponibilidade criando-se um *cluster* hibrido.

11.1.2. Cluster de Alta Disponibilidade (HA)

Um *cluster* de alta disponibilidade (*High Availability – HA*) é usado para manter o acesso à informação sempre disponível.

É um modelo construído para prover uma disponibilidade de serviços e recursos de forma ininterrupta através do uso da redundância.

São implementados para fornecer uma disponibilidade de serviços de forma sempre ativa, através de operações redundantes nos nós, ou seja, se um deles falhar, então outro nó iria executar sua tarefa, de forma que o *cluster* não se desliga por inteiro. A figura a seguir faz a representação deste funcionamento.

Por essas características o *cluster* de alta disponibilidade é utilizado especialmente em missões críticas.

11.1.3. Cluster de Alto Desempenho (HPC)

Para PITANGA apud PRADO (2010) é um tipo de sistema para processamento paralelo que consiste de uma coleção de computadores interconectados, trabalhando juntos como um recurso de computação simples e interligado de forma a conseguir um maior processamento.

Um ambiente de alto desempenho necessita de um *cluster* com inúmeros processadores, grande quantidade de memória e espaço em disco.

Esse tipo de *cluster* foi desenvolvido para oferecer maior poder de computação para a solução de um determinado problema. E é o *cluster* mais comum no meio acadêmico e científico.

O funcionamento básico de um *cluster* de alto desempenho pode ser exemplificado da seguinte forma: dado um problema, que possa ser divido em partes menores (“ser paralelizado”), um servidor fica responsável por dividir o problema em várias partes e enviar para os respectivos nós. Esses, por sua vez, acham a solução e respondem ao servidor, que junta às respostas e disponibiliza para o usuário.

11.1.4. Benefícios dos Clusters

São inúmeros os benefícios que um aglomerado de computadores, ou seja, *cluster* pode trazer para uma organização ou instituição. Porém, antes do desenvolvimento do *cluster*, a finalidade do mesmo já deve estar previamente definida, pois devem ser desenvolvidos com uma finalidade especifica e não desenvolver sem saber os processos que devem ser agilizados. (DANTAS, 2005).

Os benefícios mais importantes que os *cluster* podem nos proporcionar segundo PITANGA apud PRADO (2010) e DANTAS (2005) são:

- Baixo custo: a redução de custo para se obter processamento de alto desempenho utilizando-se simples PCs.
- Disponibilidade: devido ao cluster possuir no mínimo dois nós, em caso de ocorrer qualquer problema, a disponibilidade dos serviços não deve ser prejudicada.

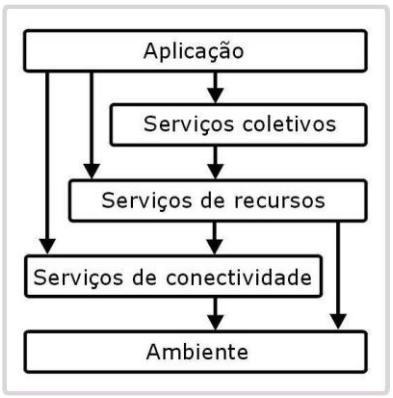
- Escalabilidade: a configuração pode crescer à medida que mais recursos estiverem disponíveis.
- Tolerância às falhas: o aumento de confiabilidade do sistema como um todo, caso alguma parte falhe.
- Alto desempenho: possibilidade de se resolver problemas muito complexos através do processamento paralelo, o que diminui o tempo de resolução dos problemas.

11.2. Grids de Computadores

Em um ambiente grid a alocação de recursos é realizada por administração descentralizada, onde cada organização controla seus próprios recursos aplicando políticas conforme sua demanda de utilização.

A possibilidade de heterogeneidade das arquiteturas e sistemas operacionais empregados é outra característica típica de um *grid*. Um grande desafio é obter uma forma de encapsular estas diferenças sem comprometer a boa performance. A infraestrutura deste ambiente deve permitir o acesso consistente aos recursos através de serviços padronizados, com interfaces e parâmetros muito bem definidos. Sem estas normas, os serviços prestados podem ser ineficazes. Entretanto, apesar de não existir uma formalização mundial, existem dois modelos apresentados na figura a seguir, que têm boa aceitação na comunidade científica e que podem se tornar padrões (de fato ou de direito) de uma arquitetura de *grid*.

(a)



(b)

Modelos de arquitetura grid

Na figura (a) ilustrada por COLVERO e CUNHA (2012) é apresentado um modelo de *grid* que deve ser compreendido da seguinte maneira:

- Aplicação e serviços: Camada formada por pacotes de software de aplicação, incluindo os conjuntos de ferramentas de desenvolvimento e serviços. A porção de serviço deve prover diversas funções de gerenciamento, tais como, faturamento, contabilidade e medidas de métricas utilizáveis – todos parâmetros importantes para o uso virtual dos recursos compartilhados entre diferentes usuários, departamentos e empresas.
- Middleware: Esta camada deve fornecer protocolos que permitam múltiplos elementos (servidores, ambientes de armazenamento, redes, dentre outros) participarem de um ambiente de grid unificado. Diversos protocolos e funções devem existir nesta camada no intuito de fornecer suporte aos elementos heterogêneos de uma configuração de grid, adaptando-se aos diferentes sistemas operacionais, sistemas de arquivos e protocolos de comunicação.
- Recursos: Esta camada é constituída pelo conjunto de recursos que fazem parte de um grid, incluindo servidores primários e dispositivos de armazenamento. Podemos citar como exemplos de configurações os clusters (por exemplo, as Nows), os serviços de armazenamento e os computadores especiais (por exemplo, os supercomputadores).

- Rede: Compõe a base da conectividade para os recursos de um grid. Assim, podemos imaginar os switches, roteadores e a infraestrutura das redes de comunicação, tais como Sonet/SDH/DWDM.

Uma outra arquitetura do modelo *grid*, mais genérica, representada na figura (b) e ilustrada por COLVERO e CUNHA (2012), é composta por cinco níveis entendidos como:

- Aplicação: Compreende as aplicações dos usuários que operam no ambiente da organização virtual, incluindo bibliotecas de funções, que usufruem os serviços prestados pelas demais camadas.
- Serviços coletivos: Enquanto a camada de recursos destina-se as interações feitas em recursos individuais, esta camada define protocolos globais que se ocupam das interações entre coleções de recursos. Os componentes dessa camada baseiam-se nos níveis de recursos e de aplicação, que implementam uma grande variedade de serviços, tais como: serviços de diretório, monitoração e diagnóstico, replicação de dados, gerenciamento de carga de trabalho e descoberta de recursos, autorização, verificação e colaboração.
- Serviços de recursos: Esta categoria corresponde à definição dos protocolos e APIs que fornecem segurança na negociação, iniciação, monitoramento, controle e na contagem dos recursos compartilhados. As implementações destes protocolos chamam as funções do nível de ambiente para acessar e controlar os recursos locais. Duas classes de protocolos podem ser distinguidas neste ponto: os protocolos de informação e os protocolos de gerenciamento.
- Serviços de conectividade: Nível que define os protocolos básicos de comunicação e autenticação necessários para as transações de rede específicas do grid. Os protocolos de comunicação permitem a troca de dados entre os níveis de ambiente e recursos, sendo atribuídos serviços como transporte e roteamento. Os protocolos de autenticação possibilitam verificar a identidade de usuários e recursos, com a aplicação de métodos de autenticação e criptografia.
- Ambiente: Esta camada tem a finalidade de traduzir as primitivas de compartilhamento de alto nível nas operações específicas de cada recurso como resultado das operações de compartilhamento nos níveis superiores. Deve incluir a implementação de mecanismos de negociação que obtenham informações sobre a estrutura, o estado e as possibilidades dos recursos, e ainda, mecanismos de gerenciamento de recursos que forneçam meios para monitoramento da qualidade de serviço.

Os recursos disponíveis em um grid, tipicamente, são da ordem de milhares, dedicados ou não, distribuídos de forma global e ligados por uma rede de interconexão que apresenta características bem diversas das normalmente empregadas em clusters, por exemplo. Por estas razões, é necessário estabelecer interconexões para acesso aos recursos e monitoramento e controle dos mesmos, através de condições adequadas de hardware e software, respectivamente.

REFERÊNCIAS

ANDREWS, G. R., *Fundations of Multithreaded, Parallel, and Distributed Programming*. London: Addison Wesley International, 2009.

COLVERO, Taís A., CUNHA, Daniel P. *Ambientes de Cluster e Grids Computacionais: Características, Facilidades e Desafios*. UNESC e UFSC, 2012. Disponível em: http://periodicos.unesc.net/sulcomp/article/download/798/750 . Acesso em Jan/2016

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. BLAIR, G. *Distributed Systems: Concepts and Design*. Fifth Edition, USA. Pearson Education, 2012

DANTAS, MARIO. *Computação Distribuída de Alto Desempenho*. 1ª edição. Rio de Janeiro: Axcel Books do Brasil Editora: 2005

DEITEL, H. M., DEITEL, P. J., CHOFINES, D. R. *Sistemas Operacionais*. 3.ed. São Paulo: Pearson Prentice Hall, 2005. Disponível em: http://feuc.bv3.digitalpages.com.br/users/publications/9788576050117/pages/_1 (Biblioteca Digital da FEUC)

MACHADO, F.B, MAIA, L.P. *Arquitetura de sistemas operacionais*. 5.ed. Rio de Janeiro: LTC, 2013.

MACIEL, Rita S. P., ASSIS, Semírames R. *Middleware: Uma solução para o desenvolvimento de aplicações distribuídas*. Salvador. CienteFico, 2004. Disponível em: http://www.cin.ufpe.br/~dmrac/infras%20de%20software/I.8.Semiramis.Middleware.pdf , Acesso em: dez/2015.

MARSHALL, D. et al. *Advanced Server Virtualization*. EUA: Auerbach Publications, 2006.

MODA, Cássio; CREMONIM, Fabiano L.; CREMONIM, Rodrigo M. *Virtualização e Alta Disponibilidade em Ambiente Corporativo*. 2009. Disponível em: http://rodrigomarassi.com/wp-content/themes/minicard/images/artigo-cientifico-virtualizacao-alta-disponibiliade.pdf. Acesso em dez/2015.

MOREIRA, Daniela. *Virtualização: rode vários sistemas operacionais na mesma máquina*. 2006. Disponível em: http://idgnow.com.br/ti-corporativa/2006/08/01/idgnoticia.2006-07-31.7918579158/

PRADO, Claudio L. e SILVA, João M.A. Aplicação de Cluster Beowulf em Instituições de Ensino. São Paulo: Faculdade de Tecnologia de Guaratinguetá. 2010.

OLIVEIRA, André L. e ROCHA, Gabriel A. M. *Compartilhamento de Arquivos através de Redes Peer-to-Peer*. Rio de Janeiro, UFF, 2003. Disponível em: http://www.midiacom.uff.br/~debora/fsmm/trab-2003-2/P2P.pdf . Acesso em: jan/2016

SILBERSCHATZ, Galvin Gaine. *Fundamentos de sistemas operacionais*. 8. Ed. Rio de Janeiro: LTC, 2010.

STEEN, Maarten Van; TANENBAUM, Andrew S. *Sistemas distribuídos*. 2.ed. São Paulo: Pearson Prentice Hall, 2012. Disponível em: http://feuc.bv3.digitalpages.com.br/users/publications/9788576051428 (Biblioteca Virtual da FEUC)

STAIR, R. M. *Princípios de sistemas de informação*. 2. Ed. Rio de Janeiro: LTC, 1998

TANENBAUM, Andrew S. *Sistemas operacionais modernos*. 3.ed., São Paulo: Pearson Prentice Hall, 2009. Disponível em: http://feuc.bv3.digitalpages.com.br/users/publications/9788576052371/pages/_1 (Biblioteca Virtual da FEUC)

TANENBAUM, Andrew S.; *Redes de Computadores*. Quinta Edição, São Paulo: Pearson Prentice Hall, 2011. Disponível em: http://feuc.bv3.digitalpages.com.br/users/publications/9788576059240 (Biblioteca Virtual da FEUC)