

PARTE 1 - INTRODUÇÃO

1. Introdução a Sistemas Distribuídos
 - 1.1 O que é um Sistema de Computação Distribuída?
 - 1.2 Porque Sistemas de Computação Distribuída?
 - 1.3 Características de projeto específicas da natureza distribuída
2. Conceitos de Hardware
 - 2.1 Multiprocessadores
 - 2.2 Multicomputadores
3. Conceitos de Software
 - 3.1 Sistemas Operacionais de Rede
 - 3.2 Sistemas Operacionais Distribuídos
 - 3.3 Sistemas Timesharing Multiprocessadores
4. Modelos de Sistemas Distribuídos

1. INTRODUÇÃO A SISTEMAS DISTRIBUÍDOS

1.1 O que é um Sistema de Computação Distribuído?

O termo é usado para definir uma grande gama de sistemas de computação (cuja característica é a interconexão de vários processadores), desde sistemas fracamente acoplados (WAN - wide area networks), sistemas fortemente acoplados (LAN - local area networks) até sistemas completamente acoplados como os sistemas multiprocessadores. Para caracterizarmos os sistemas distribuídos podemos dizer que: *Um sistema distribuído é uma coleção de computadores independentes que são ligados através de uma rede e equipados com software de sistema distribuído.* Neste caso, o software distribuído serve para que os computadores coordenem suas atividades e possam compartilhar os recursos do sistema. Em um sistema distribuído, bem projetado, o usuário pensa no sistema como se fosse um único computador, mesmo que ele seja implementado com várias máquinas em diferentes localizações.

O desenvolvimento de sistemas distribuídos está vinculado ao surgimento de redes de computadores de alta velocidade (redes locais) a partir dos anos 70. Os avanços recentes em termos de hardware (PC's de alto desempenho, estações) tem determinado uma busca maior de SCD. Esta tendência é suportada também pelo desenvolvimento de software para sistemas distribuídos.

Exemplos de sistemas distribuídos:

1. Uma rede de estações de trabalho e um conjunto de processadores que podem ser alocados dinamicamente como necessário. Tal sistema, equipado com software apropriado pode suportar as necessidades computacionais de uma população de usuários considerável, desempenhando um papel semelhante a um sistema multi-usuário centralizado, poderia ser qualificado como sistema distribuído.

Sistemas multi-usuário caracterizam-se por serem sistemas de tempo compartilhado com um único processador onde um sistema operacional trata do gerenciamento dos recursos de hardware de forma a permitir o compartilhamento dos mesmos entre os usuários. O UNIX pode ser considerado o mais conhecido exemplo de um sistema operacional multi-usuário. Por esta razão, muitos desenvolvedores adotaram o modelo UNIX como meta para sistemas distribuídos, produzindo implementações que pudessem explorar os recursos de vários computadores oferecendo facilidades e um desempenho melhor do que o convencional.

O modelo distribuído UNIX tem sido implementado de várias formas. A implementação mais usada é a desenvolvida pela *Sun Microsystems* que tem como base o UNIX BSD (Berkeley) a partir do qual foram feitas modificações que levaram ao desenvolvimento do NFS (Network File System). Este componente somado ao mecanismo de RPC (Remote Procedure Call) e ao componente NIS (Network Information Service) são usados como base para a maioria das implementações correntes do UNIX distribuído. Apesar de ser atraente em termos comerciais, o UNIX não foi projetado a partir de atributos específicos para sistemas distribuídos. Por exemplo, uma característica

importante para SOD é a sua propriedade de sistema aberto, o que permite sistemas serem estendidos para atender novos requisitos e necessidades do usuário.

2. Muitos sistemas comerciais que tem como propósito o processamento de dados e informação envolvem comunicação de dados. Exemplos podem incluir sistemas usados pelas companhias aéreas para reserva e venda de lugares, as redes operadas por bancos e grandes cadeias de lojas e supermercados. O requisito destas aplicações incluem um alto nível de confiabilidade, segurança contra interferência externa e privacidade da informação mantida pelo sistema. Eles devem fornecer acesso concorrente a banco de dados, tempo de resposta garantido, pontos de acesso ao serviço distribuídos geograficamente, potencial para crescimento e capacidade para integração de sistemas usados por diferentes companhias. A maioria das implementações destas aplicações estão baseadas em hardware, software e redes de comunicação dedicadas. Mesmo existindo uma abordagem apropriada para o desenvolvimento deste tipo de aplicação [Bacon 1993] usada com sucesso, ela sofre de limitações no que diz respeito ao crescimento, confiabilidade e segurança. Os requisitos destas aplicações são encontrados em áreas que as soluções distribuídas de propósito geral tem sido desenvolvidas.

1.2 Porque Sistemas de Computação Distribuídos - Características

Apenas o fato de ser possível a construção de sistemas distribuídos não quer dizer que é uma boa idéia. É preciso analisar muito bem a questão dos sistemas distribuídos contra os sistemas centralizados. A partir dos modelos apresentados fica evidente que é mais complexo e difícil a construção de sistemas distribuídos do que os tradicionais centralizados. A complexidade é devido ao fato de que além de ser capaz de efetivamente usar e gerenciar um numero grande de recursos distribuídos, o software de um sistema distribuído deveria ser capaz de tratar os problemas de comunicação e segurança que são completamente diferentes dos centralizados. O que faz com que os sistemas distribuídos tenham uma utilização crescente são as vantagens que eles apresentam que são maiores do que as desvantagens.

- Vantagens dos sistemas distribuídos sobre os centralizados
 - Melhor relação preço/benefício. Esta é uma das mais importantes razões para a atual tendência à distribuição. O aumento do poder computacional e diminuição do preço dos microprocessadores, combinado com o aumento da velocidade das redes de comunicação, os sistemas distribuídos tem um relação custo/benefício bem melhor que sistemas centralizados. Uma outra razão aqui é que os sistemas distribuídos é que eles facilitam o compartilhamento de recursos entre máquinas.
 - Distribuição inerente. Algumas aplicações envolvem máquinas separadas fisicamente, ou seja, o fato de muitas aplicações serem eminentemente distribuídas representa uma outra razão para a construção de sistemas distribuídos.

- Características responsáveis pela utilização de sistemas distribuídos
 - Compartilhamento de dados. Permite que mais de um usuário acesse uma base de dados comum. Isto é fundamental em muitas aplicações, de forma que estas máquinas devem necessariamente estar interconectadas (Ex. reserva de passagens numa companhia aérea). Aplicações cooperativas são conhecidas como *groupware* e dependem pesadamente do compartilhamento de dados entre programas rodando em diferentes localizações.
 - Compartilhamento de dispositivos. Permite que mais de um usuário possa ter acesso a periféricos muito caros, tais como impressoras laser. Os dados não são as únicas coisas que podem ser compartilhadas, os periféricos mais sofisticados e mais caros também são candidatos.
 - Crescimento incremental. O poder computacional pode crescer dependendo da necessidade. Considerando um *mainframe*, se a carga de trabalho aumenta, chega-se um ponto em que a única solução seria comprar uma máquina maior. Em um sistema distribuído, é possível adicionar mais processadores ao sistema permitindo assim um aumento gradual. Além disto a inclusão de novos recursos pode ser realizada sem afetar o funcionamento normal do sistema (sistemas distribuídos abertos).
 - Concorrência. A existência de vários processos executando em um computador monoprocessador é chamada de execução concorrente. Em um sistema distribuído com M computadores (1 proc.) existirão pelo menos M processos executando simultaneamente. Concorrência e execução paralela acontece naturalmente em um sistema distribuído devido as atividades separadas dos usuários, a independência dos recursos e a localização de servidores em máquinas separadas. A separação destas atividades permite que o processamento aconteça em paralelo em máquinas separadas. Acessos e atualizações concorrentes a recursos devem ser sincronizados.
 - Escalabilidade. O crescimento de um sistema faz com que, normalmente, o mesmo falhe a medida que o número de máquinas aumenta. Um sistema que inicialmente é implementado para umas centenas de usuários irá suportar alguns milhões? O software de um sistema distribuído, sistema e aplicação, não deveria precisar de mudanças quando o número de componentes do sistema cresce. Como o tamanho e complexidade das redes cresce, o desafio maior é o projeto de software de sistema distribuído que seja efetivo frente a este crescimento. O requisito é projetar serviços distribuídos que continuem operando de forma efetiva com mil ou milhões de clientes. O trabalho envolvido no processamento de cada requisição para acesso de recursos compartilhados deve ser independente do tamanho da rede.

- Confiabilidade. Se uma máquina cair, o sistema como um todo pode sobreviver. Se tivermos distribuído a carga de trabalho por muitas máquinas, a falha de um simples processador vai derrubar no máximo uma única máquina, deixando intacto o restante do sistema (5% fora representa um decréscimo de 5% no desempenho). Um aspecto importante da confiabilidade é a disponibilidade, que significa o tempo no qual o sistema está disponível para uso.
- Transparência. Como conseguir a imagem de um sistema único? Esta é, provavelmente, a característica mais importante a ser tratada. Sistemas que conseguem isto são ditos *transparentes*. O conceito de transparência pode ser aplicado a vários aspectos dos sistemas distribuídos:
 - Localização : os usuários não podem dizer onde os recursos de software e hardware estão localizados.
 - Migração : os recursos devem estar livres para serem movimentados de um local para outro sem que seus nomes sejam trocados.
 - Replicação : permite que o sistema faça cópias de recursos (arquivos) sem que os usuários saibam disto.
 - Concorrência : permite que múltiplos usuários possam compartilhar recursos automaticamente. Os usuários não notam a existência de outros usuários.

Apesar dos sistemas distribuídos apresentarem muitas vantagens em relação aos centralizados, existem alguns aspectos que devem ser analisados com cuidado, antes de decidir sua utilização.

- Software. Até o presente momento não há muita disponibilidade de software para os sistemas distribuídos. No estado atual da arte, não há muita experiência de projeto, implementação e de utilização do software distribuído (SO, linguagens, aplicações).
- Ligação em rede. A rede pode saturar. O ponto principal aqui é a possibilidade de que sejam perdidas mensagens na rede, o que nos obriga a utilizar um software especial para fazer a manipulação das mensagens. Outro aspecto é a possibilidade da rede ficar sobrecarregada com o tráfego gerado pelo troca de mensagens.
- Segurança : os dados secretos também são acessíveis facilmente. Se as pessoas podem acessar facilmente os dados disponíveis no sistema, significa que fica fácil acessar dados que em tese não deveriam interessar ao usuário que os consultou.

1.3 Características de projeto específicas da natureza distribuída

Mesmo que o projeto de um sistema distribuído ou aplicação deva considerar características não relacionadas com distribuição (técnicas de engenharia de software, interfaces computador-usuário), na construção de um sistema distribuído algumas características são específicas da natureza distribuída do sistema.

- Nomeação. Sistemas distribuídos são baseados no compartilhamento de recursos e na transparência da sua distribuição. Os nomes atribuídos aos recursos ou objetos deve apresentar significado global que seja independente de localização do objeto e deve ser suportado por um sistema de interpretação que traduza nomes de forma a permitir o acesso de programas a recursos com nome. É importante usar esquemas que tenham escalabilidade e onde os nomes sejam traduzidos eficientemente conforme as metas de desempenho.
- Comunicação. O desempenho e confiabilidade das técnicas de comunicação usadas na implementação de sistemas distribuídos são críticas em se tratando do desempenho do sistema distribuído. A otimização da comunicação em sistemas distribuídos com a manutenção de um modelo de programação de alto nível é uma característica importante.
- Estrutura de Software. A idéia de sistemas abertos é conseguida através do projeto e construção de componentes de software com interfaces bem definidas. Abstração de dados é uma técnica de projeto importante para sistemas distribuídos. Serviços podem ser vistos como gerentes de objetos de um dado tipo de dados, onde a interface do serviço pode ser vista como um conjunto de operações. Estruturar um sistema de forma que novos serviços possam ser introduzidos e integrados completamente com serviços existentes sem existir duplicação destes é uma característica importante. É importante que o sistema seja flexível para que ele possa ser modificado, em função de novas tendências e erros cometidos. Existem duas tendências com respeito a estrutura dos SOD'. Numa delas, cada máquina deveria rodar um kernel tradicional que fornece a maioria dos serviços (kernel monolítico). Na outra, o kernel deveria fornecer o mínimo possível de serviços, sendo que a maior parte dos serviços do SO seriam fornecidos por servidores no nível do usuário (microkernel) (figura 4).
- Alocação de carga de trabalho. Bom desempenho é um requisito de grande interesse para programadores e projetistas. Sistemas distribuídos apresentam como característica importante a maneira como recursos de processamento e comunicação são alocados na rede para conseguir um efeito ótimo no processamento de uma mudança da carga de trabalho.
- Manutenção da consistência. A idéia de sistemas distribuídos traz consigo problemas de consistência. O significado destes problemas para o projeto é o seu impacto no desempenho dos sistemas distribuídos. A manutenção da consistência

a custos razoáveis é talvez o problema mais difícil encontrado no projeto de sistemas distribuídos. Alguns tipos importantes de consistência são: 1) Consistência nas modificações, quando vários processos acessam e modificam dados concorrentemente a atividade de modificar um conjunto de dados relacionados não pode ser realizada instantaneamente, mas é desejável que as modificações aparentem ser atômicas. 2) Consistência de replicação, quando dados derivados de uma única fonte são copiados para vários computadores e modificados em um ou mais deles, a possibilidade de inconsistências existe entre os valores dos dados nas diferentes máquinas. 3) Consistência de cache, quando dados são mantidos em cache em um cliente e modificados por outro. 4) Consistência em falhas, se outros nodos dependem de um faltoso eles também podem falhar mais tarde. 5) Consistência de clock, a noção de tempo físico ou absoluto em sistemas distribuídos é um problema. 6) Consistência de IU, modificações a nível de interface de usuário, mudança de uma tela, podem se tornar inconsistentes para o usuário.

- Requisitos de usuário. Alguns requisitos de usuário que devem ser considerados quando do projeto de sistema distribuídos: 1) Funcionalidade, que serviços os usuários podem esperar de um sistema de computação distribuído? Pelo menos aqueles que o usuário consegue em um PC. 2) Reconfigurabilidade, diz respeito a habilidade que o sistema tem para acomodar mudanças que podem acontecer durante a execução do sistema, sem que o mesmo seja interrompido. 3) Qualidade de serviço (QoS), um SD deve fornecer uma performance adequada em termos de tempo de resposta aos usuários. Confiabilidade e disponibilidade do sistema também caracterizam qualidade de serviço.

Todas as características anteriores de nada adiantam se o SOD for uma carroça. Ele não deve comportar-se de maneira pior do que um sistema centralizado. Algumas métricas podem ser usadas : tempo de resposta, throughput (número de jobs por hora), utilização do sistema e quantidade de recursos da rede consumidos. O problema do desempenho leva em consideração o fato de que a comunicação, essencial nos SOD's, é lenta. Uma grande parte do tempo de comunicação é gasto em protocolos. Na busca da otimização do desempenho poderíamos minimizar o número de mensagens. Entretanto, uma das formas de ganhar desempenho é ter muitas atividades em paralelo nos vários processadores, e isto requer mais mensagens. Uma das formas é prestar atenção na granularidade das computações. Paralelismo de grão fino (grande número de pequenas computações com grande interação) deve ser evitado em favor do paralelismo de grão grosso (grandes computações com pequeno número de interações). Aspectos de tolerância a falhas também influenciam de forma pesada no desempenho.

2. CONCEITOS DE HARDWARE

Mesmo considerando que todos os sistemas distribuídos são, em última análise, compostos de vários processadores, existem várias formas diferentes de se organizar o hardware de tais sistemas, especialmente no que diz respeito a implementação da conexão de seus componentes, e de como eles se comunicam.

Diversos esquemas de classificação para os sistemas de computação com múltiplas CPU's tem sido propostos. A taxonomia mais citada é a de Flynn (1972), onde ele evidencia duas características consideradas por ele essenciais: o número de fluxos de instruções e o número de fluxos de dados.

SISD: computador com um fluxo de instruções e um fluxo de dados (todos os computadores com um único processador).

SIMD: computadores com um fluxo de instruções e múltiplos fluxos de dados. Esta categoria refere-se a um conjunto (array) de processadores com uma unidade de controle que busca uma instrução que comanda várias unidades aritméticas que executam em paralelo, cada uma delas com seus próprios dados (supercomputadores - repetem o mesmo cálculo em vários conjuntos de dados).

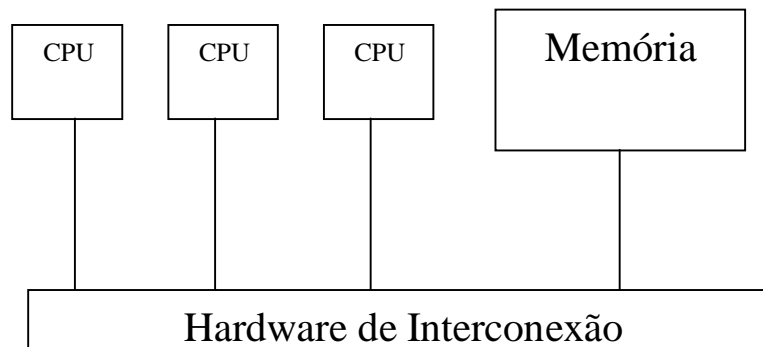
MISD: computadores com múltiplos fluxos de instrução e um fluxo de dados.

MIMD: grupo de computadores independentes, cada um com o seu próprio PC, programa e dados. Sistemas distribuídos são considerados MIMD. Neste caso, estas máquinas podem ser divididas em dois grupos: aquelas com memória compartilhada, chamadas *multiprocessadores* e aquelas com memória privativa chamadas *multicomputadores*. Cada uma destas categorias pode ser dividida com base na arquitetura da rede de conexão: barramento (*bus* - uma única rede: barramento, cabo ou outro meio que conecte todas as máquinas) e chaveada (*switched* - fios individuais conectam uma máquina a outra com decisões de chaveamento).

Outra dimensão da taxonomia é que os sistemas de computação distribuída, consistindo de múltiplos processadores interconectados, são classificados em 2 tipos básicos:

1. Os sistemas *fortemente acopladas*. Nestes sistemas, existe uma única memória para o sistema todo (vista pelo sistema todo) que é compartilhada por todos os processadores. Qualquer comunicação, neste tipo de sistema, é feita através da memória compartilhada.
2. Os sistemas *fracamente acopladas*. Estes sistemas não compartilham memória, cada processador tem a sua memória local. Aqui todas as comunicações entre processadores são realizadas a partir da passagem de mensagens através da rede de interconexão

2.1 Multiprocessadores (Sistemas fortemente acoplados)



- Multiprocessadores baseados em barramento

Estes sistemas são compostos por um número de CPU's conectados por um barramento comum, acessando uma memória comum. Um exemplo simples para um sistema deste tipo poderia ser uma placa-mãe de alta velocidade (32 linhas de endereço, 32 de dados e 20 a 30 de controle), ligados nela as placas com os processadores e as memórias. Se a memória é *coerente*, então A pode escrever na memória e B logo em seguida obterá o novo valor. Este esquema apresenta como problema a diminuição drástica do desempenho em função do aumento do número de processadores (com 4 ou 5 processadores o barramento fica saturado). A solução utilizada é normalmente a incorporação de uma memória *cache* em cada processador. Desta forma é possível em até 90% dos casos utilizar os dados na memória *cache* diminuindo com isto o tráfego no barramento e desta forma aumentando o número de processadores (é possível de 32 a 64 processadores). Entretanto, a utilização das *caches* estabelece um problema de consistência. Se dois processadores A e B estão lendo a mesma palavra de suas respectivas *caches* e A escreve nesta palavra, quando B for ler novamente obterá o valor antigo e não o que A acabou de escrever. Para eliminar este problema são utilizadas as chamadas *caches write-through* (assim que um dado é escrito na *cache* é imediatamente atualizado na memória).

- Multiprocessadores com chaveamento

Quando se deseja um número de processadores maior do que o limite estabelecido na abordagem anterior, emprega-se um método diferente para efetuar a conexão do processador e a memória. Uma delas é a divisão da memória em módulos que são conectados aos processadores através de um chaveamento do tipo *crossbar* (barramento cruzado), assim cada processador e cada memória tem uma conexão ao sistema e em cada interseção existe uma *chave eletrônica* que pode ser aberta ou fechada por hardware. O fechamento de uma chave significa a possibilidade de um processador acessar uma memória. Várias chaves podem estar fechadas em um determinado instante, significando que vários processadores podem estar acessando a memória ao mesmo tempo (não o mesmo módulo). O problema aqui é o número de chaves necessárias para conectar n processadores à n memórias (n^2 chaves). Uma solução encontrada, que é denominada de *rede ômega*, diminui

o número de chaves necessárias ($n \log_2 n$). Um outro problema é o retardo que se tem para acessar as memórias (Ex. do livro). Uma solução encontrada, conhecida por máquina *NUMA* (NonUniform Memory Access) atribui uma memória a cada processador sendo que o acesso a esta memória é mais rápido e o acesso a qualquer outra memória é bem mais demorado. Os sistemas multiprocessadores baseados em barramento são limitados pelo número de processadores enquanto que as redes do tipo *crossbar* e suas variações são muito caras e envolvem algoritmos muito complexos. Isto nos leva a concluir que a construção de um sistema multiprocessador com memória compartilhada, fortemente acoplado, é uma tarefa difícil e muito cara.

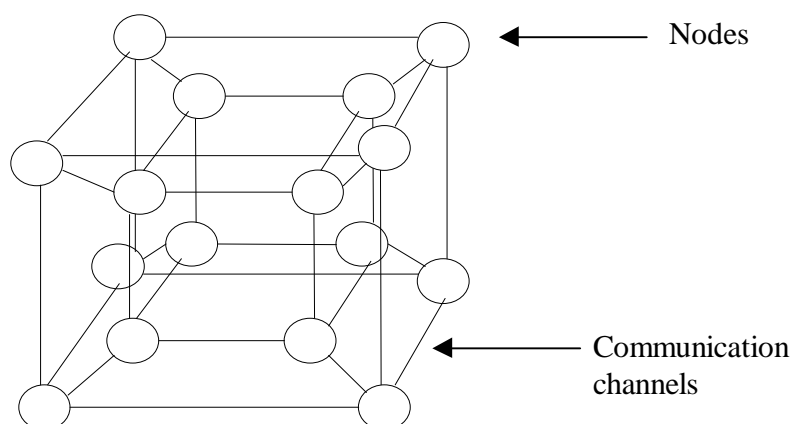
2.2 Multicomputadores (Sistemas fracamente acoplados)

- Multicomputadores baseados em barramento

No caso de um sistema multicomputador (sem memória compartilhada) as dificuldades de construção são menores porque, neste caso, cada processador tem uma conexão direta com a sua própria memória local. Aqui o problema diz respeito a como os processadores se comunicam uns com os outros. É necessário algum esquema de interconexão que, pelo fato de envolver apenas comunicação processador-processador (velocidade pode ser menor), tem um volume de tráfego bem menor que no caso de tráfego entre processadores e memórias.

- Multicomputadores com chaveamento

Neste caso a interconexão dos processadores é feita por chaveamento. Várias redes para conexão deste tipo de sistema foram propostas e construídas, sendo que em todas elas cada processador tem acesso direto e exclusivo a sua própria memória privada. Os exemplos mais conhecidos são: a *grelha* (grade) e o *hipercubo*. As *grelhas* são mais apropriadas para problemas de natureza bidimensional, tais como aqueles que envolvem a teoria dos grafos e visão robótica. O *hipercubo* é um cubo n -dimensional. O de dimensão 4 é visto como 2 cubos comuns, sendo que cada um tem 8 vértices e 12 lados. Cada vértice é um processador e cada lado é uma conexão entre 2 processadores. Os vértices correspondentes em cada um dos 2 cubos são também conectados. No caso do *hipercubo* de dimensão n cada processador tem n conexões a outros processadores, com isto a complexidade das interligações cresce de forma logarítmica com o tamanho da estrutura. Considerando que só os vizinhos próximos devem ser conectados, a maioria das mensagens deve andar bastante pela estrutura antes de chegar a seu destino. Porém, o caminho mais longo possível de ser percorrido também cresce em escala logarítmica com o tamanho da estrutura (*grelha* n^2). Se tem sistemas baseados em *hipercubo* com 1024 processadores e até 16.384.



3. CONCEITOS DE SOFTWARE

O desenvolvimento de software para sistemas distribuídos envolve tanto aplicações quanto sistemas. De uma forma geral, o desenvolvimento de sistemas distribuídos é baseado em modelos como cliente/servidor e baseado em objetos com o objetivo de caracterizar o compartilhamento de recursos. O modelo cliente/servidor é caracterizado por um conjunto de processos servidores, cada um funcionando como gerente de recurso para uma coleção de recursos de um determinado tipo, e uma coleção de processos clientes, cada um realizando uma tarefa que necessita acesso a algum recurso, de hardware ou software, compartilhado. No modelo baseado em objetos, cada recurso compartilhado é visto como um objeto. Os objetos são identificados unicamente e podem ser movimentados para qualquer lugar na rede sem mudar suas identidades. Um programa que utiliza um recurso requisita acesso ao mesmo enviando uma mensagem de requisição para o objeto correspondente. A mensagem é entregue para o procedimento ou processo que então realiza a operação requisitada e envia uma mensagem de resposta para o processo requisitante. Atualmente, grande parte dos sistemas de suporte, especificamente sistemas operacionais, utilizam o modelo cliente/servidor, onde os clientes são processos de usuário e servidores são processos do sistema. Entretanto, existem sistemas de suporte no nível de middleware orientados a objetos distribuídos. Neste caso, o suporte assume um nível funcional entre o cliente e o servidor sendo que este nível fornece serviços para os objetos comunicantes tais como localização e autenticação.

Sistemas operacionais distribuídos apresentam muitos aspectos comuns aos sistemas operacionais centralizados, mas também diferem em outros tantos. Um SOD apresenta os mesmos elementos de gerenciamento que um centralizado: gerenciamento de processos, de memória, de I/O, de comunicações e de arquivos. Entretanto, deve ser colocado que a distribuição dos recursos, a ausência de um estado global e o retardo nas transmissões, implica que os métodos e técnicas desenvolvidas para os sistemas centralizados não podem ser usadas nos sistemas operacionais distribuídos. Mesmo que as técnicas centralizadas atendam os requisitos daqueles sistemas, sua implementação é muito cara. Isto colocado, um sistema operacional distribuído deveria:

- controlar a alocação dos recursos da rede para permitir o seu uso da maneira mais efetiva.

- fornecer para o usuário um computador virtual conveniente para servir como um ambiente de programação de alto nível.
- esconder a distribuição dos recursos.
- fornecer mecanismos para proteger os recursos do sistema contra acesso de usuários não autorizados.
- fornecer segurança na comunicação.

Um sistema operacional distribuído é aquele que visualiza o usuário como um sistema operacional centralizado, mas roda em múltiplas CPU's independentes onde:

- o uso de múltiplos processadores deveria ser invisível (transparente) para o usuário, ou
- os usuários veem o sistema como um sistema monoprocessador virtual, não como uma coleção de máquinas distintas conectadas por um sistema de comunicação.

Podemos considerar que apesar da importância que o hardware tem sobre o sistema computacional, o software é mais importante, pois é ele, mais especificamente o sistema operacional, que fornece a imagem do sistema como um todo para os usuários. De uma forma geral, sistemas de suporte em sistemas de computação distribuídos localizados a nível do sistema operacional são classificados em: sistemas operacionais de rede, sistemas operacionais distribuídos e sistemas time-sharing multiprocessadore. Uma forma de caracterizar cada sistema é identificando objetivos e características. Algumas metas são particularmente importantes, interoperabilidade, transparência e autonomia, e definem objetivo principal de cada um dos sistemas. Outras características usadas para diferenciar estes sistemas são a imagem do sistema e a capacidade de tolerância a falhas.

3.1 Sistemas Operacionais de Rede

Como visto anteriormente uma rede de computador pode ser definida como um sistema fracamente acoplado de múltiplos computadores. O objetivo primário em uma rede é o compartilhamento de recursos (incluindo programas e dados). A única interação no sistema é a troca de informação entre as estações de trabalho através de um canal de comunicação externo. O termo interoperabilidade caracteriza de forma única as características desejadas em um sistema de rede. Protocolos de comunicação padronizados e interfaces comuns para banco de dados e sistemas de arquivo são exemplos de mecanismos para suportar interoperabilidade. Interoperabilidade também fornece flexibilidade permitindo a troca de informação entre estações em uma rede heterogênea. Considere o exemplo de um conjunto de estações de trabalho interligadas através de uma rede local. Neste caso, cada usuário tem uma estação para seu uso exclusivo, com ou sem disco e com um sistema operacional.

A função de troca de informação pode ser dividida em níveis de forma que no nível de hardware a responsabilidade da troca de informação é da subrede de comunicação enquanto que nos níveis mais altos o sistema operacional fornece serviços de transporte de dados. Um exemplo de organização destes níveis é a arquitetura OSI-ISSO. Numa situação como esta, onde cada estação tem um alto grau de autonomia e existem poucas exigências a nível de

sistema, o software de controle do ambiente é conhecido como *Sistema Operacional de Rede* (SOR). Sistemas operacionais de rede são considerados como uma coleção de sistemas operacionais conectados a uma rede e incorporando módulos que permitem acesso a recursos remotos. Neste caso, os usuários estão conscientes do fato de que vários computadores estão sendo usados. Um sistema operacional de rede tem as seguintes características:

1. Cada computador tem o seu próprio sistema operacional.
2. Cada usuário trabalha em seu próprio computador e a utilização de um computador diferente deste necessita um *login* remoto.
3. Os usuários tem conhecimento de onde cada um dos seus arquivos está armazenado e movem estes arquivos através das máquinas com comandos explícitos de transferência de arquivos.
4. O sistema quase não tem tolerância a falhas.

Num ambiente deste tipo podemos citar os seguintes módulos (figura 2): comunicação, o sistema operacional de rede, o sistema operacional local e os usuários. O módulo de comunicação fornece entrega confiável de mensagens entre as máquinas. O sistema operacional de rede liga e coordena ações remotas e comunicação entre os sistemas operacionais locais. O sistema operacional local é um sistema centralizado tradicional, apenas ele invoca o sistema operacional de rede quando o serviço requisitado não pode ser realizado localmente. O sistema de arquivos NFS (Network File System) da Sun segue este modelo.

A maioria dos sistemas operacionais de rede usam uma API (Application Programming Interface) com sockets e chamada de procedimento remoto para serviço de transporte com o objetivo de suportar comunicação entre sistemas operacionais em domínios de rede diferentes. Um SOR é caracterizado pela inclusão de um nível de transporte e o suporte para aplicações de rede que executam no serviço de transporte. Exemplo de aplicações de rede incluem: login remoto, transferência de arquivos, e execução remota. Login remoto torna a estação de um usuário em um terminal de uma estação remota da rede e permite compartilhamento do processador remoto e seus recursos. Basicamente as entradas são transformadas em pacotes que são enviados através da rede conforme protocolos estabelecidos. Uma aplicação de rede com estas características é *telnet*. Transferência de arquivos é a capacidade de transferir arquivos ou partes entre diferentes estações da rede. Um protocolo de transferência de arquivos, tipo *ftp*, deve fornecer uma interface para o sistema de arquivos local e suportar comandos interativos do usuário. Execução remota é a capacidade de enviar uma mensagem requisitando a execução de um programa em uma estação remota. Uma vez que programas executáveis são dependentes de máquina execução remota implica frequentemente em interpretação de arquivos do tipo script ou códigos intermediários. Um exemplo de abordagem para execução remota é a linguagem Java e seu ambiente de programação.

3.2 Sistemas Operacionais Distribuídos

Nos sistemas operacionais de rede cada ponto pode rodar o seu próprio SO e não existe, necessariamente, coordenação a não ser as regras que aplicações do tipo cliente-servidor devem seguir. A evolução deste tipo de sistema são os sistemas com software fortemente acoplado em hardware fracamente acoplados (multicomputadores). Neste caso, a meta é fazer com que o usuário tenha a sensação de que todos os computadores do sistema compõem um único sistema timesharing e não uma coleção de máquinas distintas (single-system image, virtual uniprocessor). Ou seja, incluir mais formas gerais de atividades cooperativas entre uma coleção de sistemas de computação conectados. Como os recursos e atividades distribuídas são gerenciados e controlados é a responsabilidade primária de um SOD. Esta necessidade de implementação distribuída das funções de gerenciamento e controle do sistema é suprida por algoritmos distribuídos.

Atualmente, nenhum sistema preenche os requisitos completamente. Algumas características dos sistemas distribuídos são apresentadas a seguir:

- deve existir um mecanismo global de comunicação entre processos de forma que qualquer processo possa se comunicar com qualquer outro;
- deve existir um esquema global de proteção;
- o gerenciamento de processos deve ser o mesmo em qualquer lugar;
- deve existir um conjunto de chamadas de sistema disponíveis em todas as máquinas de forma a fazer sentido em um ambiente distribuído;
- o sistema de arquivos deve ser visto da mesma forma em qualquer lugar sendo os arquivos vistos por todos;
- o kernel do SO é igual para todas as CPU's facilitando a coordenação das atividades que devem ser globais, mesmo assim cada kernel pode ter um controle maior sobre os seus recursos (gerência de memória, escalonamento).

Uma necessidade básica de um SOD é a de esconder detalhes de implementação dos usuários do sistema. Desta forma, a distinção chave entre SOR e SOD reside no conceito de transparência.

3.3 Sistemas Time-sharing Multiprocessador

Esta combinação é chamada de software fortemente acoplado com hardware fortemente acoplado. Nesta categoria existem várias máquinas de propósito específico (máquinas dedicadas). Considerando propósitos gerais, o exemplo mais comum são os multiprocessadores que rodam o UNIX. Neste caso, a característica principal é a existência de uma única fila de execução (todos os processos em uma lista de execução mantida em memória compartilhada). Os processos ficam na memória compartilhada e cada CPU executa um processo. Quando um processo bloqueia a espera de I/O ou outro evento, a CPU responsável por aquele processo deve encontrar outro para rodar (executa o escalonador de forma mutuamente exclusiva). Não existe preferência de CPU por parte de um processo, a não ser quando este mesmo processo já rodou em uma CPU que não foi usada desde sua parada (figura 3).

Este tipo de sistema é diferente dos sistemas de rede e distribuídos no que diz respeito ao sistema de arquivos. Eles tem um sistema de arquivos tradicional, onde cada processo que executa uma chamada de sistema causa um trap para o SO que de alguma forma bloqueia a entrada de outras CPU's enquanto estruturas de dados críticas estão sendo acessadas. Em alguns sistemas multiprocessadores uma CPU é dedicada a execução do SO enquanto que as outras executam programas de usuário.

O modelo de objetos distribuídos é normalmente construído a partir do conceito de middleware. A idéia básica é o encapsulamento de dados em interfaces funcionais para os objetos escondendo detalhes de implementação e limitando o acesso para a invocação de métodos definidos para os objetos. Os métodos são invocados nos objetos de forma indireta, via referências, eliminando a necessidade de instâncias locais para os objetos. Desta forma, arquiteturas de objetos distribuídos suportam transparência a nível de localização, plataforma e linguagem. O custo da transparência é normalmente baixa performance. Como exemplo de software de suporte a objetos distribuídos podemos citar CORBA, DCOM e Java RMI.

4. MODELOS DE SISTEMAS DISTRIBUÍDOS

Vários modelos são usados para construir sistemas de computação distribuídos. Estes modelos podem ser classificados em cinco categorias: minicomputador, estação de trabalho, estação de trabalho servidora, conjunto de processadores e híbrida.

⇒ Modelo minicomputador

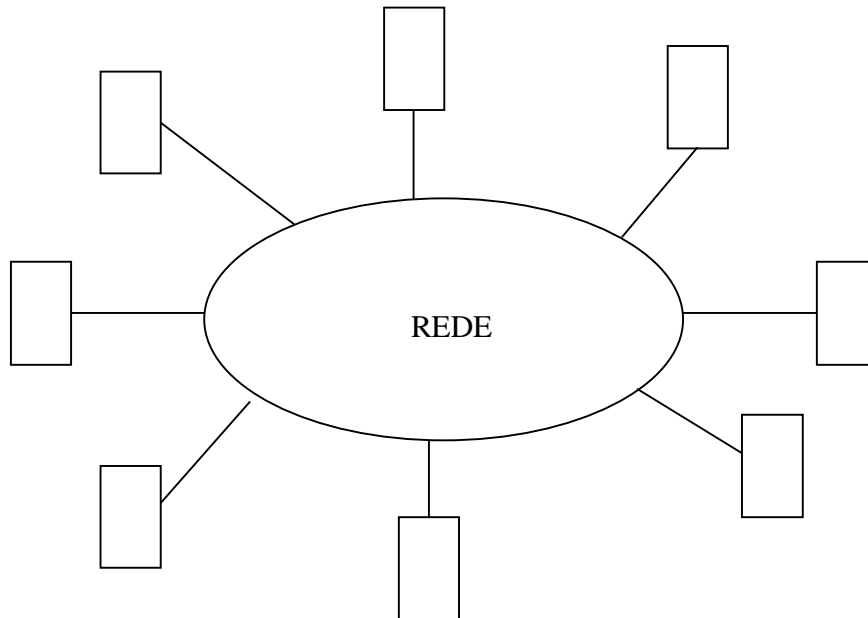
É uma extensão do modelo tempo compartilhado centralizado. Consiste de um conjunto de minicomputadores interconectados por uma rede de comunicação. Cada minicomputador tem múltiplos usuários. Cada usuário do SD está logado em um minicomputador com acesso remoto aos outros minicomputadores. A rede permite um usuário acessar recursos remotos que estão disponíveis em alguma outra máquina.

⇒ Modelo estação de trabalho (Workstation)

Este modelo consiste de um conjunto de estações conectadas por uma rede de comunicação, sendo que cada estação esta equipada com o seu próprio disco e sendo usada como um computador pessoal. Visto que neste caso é bem provável que em algum tempo muitas estações vão estar ociosas, a idéia deste modelo é a utilização de uma LAN com velocidade bem alta de forma que as estações ociosas possam ser usadas por processos de usuários logados em outras estações. Quando o sistema entende que o usuário não tem poder computacional suficiente para uma execução eficiente ele transfere processos para outras estações que estão ociosas e executa os processos lá. A implementação deste modelo não é tão simples e algumas considerações devem ser abordadas:

1. Como encontrar uma estação ociosa?

2. Como é feita a transferência de uma processo de uma estação para outra e como ele é executado?
3. O que acontece com o processo remoto se o dono da estação se logar nela?



⇒ Modelo estação de trabalho servidora (Workstation-server)

Este modelo consiste de um pequeno conjunto de minicomputadores e várias estações (maioria sem disco) interconectados por uma rede de comunicação. Quando estações sem disco são usadas o sistema de arquivos usado por elas deve estar implementado em uma estação com disco ou em um minicomputador. Neste caso os minicomputadores são usados como servidores (de arquivo, de impressão, etc), assim sendo neste modelo além das estações existem máquinas especializadas chamadas *servidores*. Por razões de confiabilidade e escalabilidade é possível a existência de vários servidores do mesmo tipo cooperando entre si através da rede. Neste modelo, o usuário se loga em uma estação e realiza ali computações normais. Entretanto, requisições para serviços oferecidos por servidores especiais (sistema de arquivos) são enviados para o servidor que fornece o tipo de serviço que o usuário necessita e devolve a resposta para o mesmo. Em comparação com o modelo anterior este modelo apresenta uma série de vantagens:

1. Em geral, é mais barato utilizar poucos minicomputadores com discos de grande capacidade que são acessados através da rede do que muitas estações com disco.
2. Estações sem disco tem preferência do ponto de vista de manutenção. É mais fácil realizar operações de manutenção (backup, hardware) em poucos discos do que em muitos discos. A instalação de novas versões de software é facilitada (servidor de arquivos).
3. Visto que o gerenciamento de arquivos é feito pelos servidores de arquivo, os usuários tem flexibilidade de usar qualquer estação.

4. O acesso aos serviços é feito a partir de um protocolo do tipo cliente/servidor não necessitando de migração de processos.
5. O tempo de resposta é garantido visto que as estações não são usadas para execução remota de processos. Neste caso, o modelo não utiliza a capacidade de processamento das estações ociosas.

⇒ Modelo *Pool* de processadores (Processor-Pool)

Este modelo é baseado na observação de que na maioria do tempo o usuário não precisa de grande poder computacional mas algumas vezes ele pode precisar (recompilar um programa composto por vários arquivos). Assim, no modelo os processadores são mantidos em um *pool* para serem compartilhados pelos usuários. Cada processador no *pool* tem a sua própria memória e não tem terminal associado diretamente a ele. O sistema é acessado através de terminais que estão conectados a rede, estes podem ser estações, terminais gráficos ou terminais X. Existe um servidor especial, servidor de execução (run server), que é responsável pelo gerenciamento dos processadores. O usuário não executa login em uma máquina em particular mas no sistema como um todo. Este modelo permite uma melhor utilização do poder de processamento disponível (estação servidor) e fornece uma flexibilidade maior que o modelo estação servidor no que diz respeito a expansão dos serviços (os processadores no pool podem ser alocados para serviços). O modelo não é adequado para aplicações interativas de alto desempenho.

