CAPÍTULO I – INTRODUÇÃO

Sistema Distribuído: é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente. Ou seja, são componentes autônomos cujos usuários veem feito um único sistema. Em grande parte, a diferença entre vários computadores, suas organizações e suas comunicações estão ocultas dos usuários. Além disso, usuários e aplicações podem interagir com um sistema distribuído de forma consistente e uniforme.

Middleware: camada lógica, acima dos SOs e abaixo de usuários e aplicações, a qual suporta computadores e redes heterogêneas, oferecendo uma visão de sistema único.

Metas do Sistema Distribuído: acesso a recursos; transparência da distribuição; abertura; escalabilidade.

Acesso a Recursos: deve facilitar o acesso e o compartilhamento de recursos remotos, de maneira controlada e eficiente.

Transparência da Distribuição: é o nível com que um SD consegue se apresentar a usuários e aplicações como se fosse um único sistema. Note que nem sempre ocultar aspectos é bom. Há muitas limitações físicas de velocidade de transmissão não-camufláveis, problemas relativos a fuso horário, a localização (quando se deseja buscar serviços próximos, não se deve ocultá-la.) Além disso, pode-se gastar mais recursos que o justificável para promover a transparência, não se fazendo viável, portanto.

- **Trans. de acesso:** oculta diferenças em representação de dados (*big endian* e *little endian*), e o modo como os recursos podem ser acessados por usuários, como nomenclatura do sistema de arquivos.

- **Trans. de localização:** os usuários não devem poder dizer qual é a localização física de um recurso no sistema. Para isto, confere-se nomes lógicos a recursos, em vez de nomes que secretamente indiquem sua localização. A URL é um exemplo de nome lógico.

- **Trans. de migração:** a movimentação de recursos não afeta o modo como estes podem ser acessados, e, para isto, também são necessários nomes lógicos.

- **Trans. de relocação:** sem que se perceba, recursos podem ser relocados enquanto estão sendo acessados. Um exemplo é a movimentação por diferentes redes usando *laptops*.

- **Trans. de replicação:** capacidade de ocultar o fato de que existem diversas cópias de recursos. Para isto, toda réplica deve possuir o mesmo nome. Assim, só funciona se houver nomes lógicos e, consequentemente, transparência de localização.

- **Trans. de concorrência:** capacidade de que um usuário não perceba que um outro está usando o mesmo recurso, especialmente em casos de compartilhamento competitivo, mas útil também ao compartilhamento cooperativo. Para que o estado do recurso se mantenha consistente, deve-se usar mecanismos como travas de edição ou transações.

- **Trans. à falha:** capacidade de mascarar que um recurso deixou de funcionar bem e que o sistema se recuperou da falha. Esta transparência é quase impraticável na vida real, porque é difícil distinguir entre um recurso morto e um que está apenas muito lento.

Abertura: um SD é aberto quando oferece interfaces, serviços de acordo com regras padronizadas que descrevem a sintaxe e a semântica desses serviços. Por isto, tais interfaces devem ter:

- **Interoperabilidade:** até quanto duas implementações devem coexistir/trabalhar em conjunto.

- **Portabilidade:** até que ponto uma aplicação desenvolvida para A pode ser executada sem modificações em B.

- **Extensível:** ser fácil de adicionar partes que são executadas em SOs diferentes.

Escalabilidade em tamanho: fácil de adicionar mais usuários e recursos ao sistema.

Escalabilidade em termos geográficos: usuários e recursos podem estar distantes de si.

Escalabilidade em termos administrativos: fácil de administrar, ainda que abranja muitas organizações administrativas diferentes.

Em geral, perde-se desempenho quanto mais escalável for o SD, mas comunicações assíncronas auxiliam na performance. Auxilia, também, a redução da comunicação, ao deixar o cliente executar parte da computação do servidor, feito quando se usa JS para verificar os campos de um formulário antes de enviá-lo.

Outra técnica é a distribuição, em que se toma um componente e subdivide-se ele em fragmentos menores, espalhando-os depois, como o DNS. O uso de cache, uma forma de replicação, também auxilia na melhora de performance da escalabilidade. Isto porque ele deixa, próximo a usuários distantes, uma cópia dos recursos utilizados.

Sistemas distribuídos de computação

Utilizados para tarefas de computação de alto desempenho. Pode ser:

- **Clusters:** o hardware consiste em um conjunto de estações de trabalho (PCs) semelhantes, conectados por rede local de alta velocidade. Usado, principalmente, para computação paralela em que um único programa intensivo é executado paralelamente em várias máquinas. São homogêneos.

- **Em grade:** têm alto grau de heterogeneidade, não havendo premissas de como deve ser o hardware, SO, rede. Fornece grupos de recursos a certas organizações virtuais. O foco nestes sistemas é a arquitetura, a fim de prover acessos corretamente a recursos de diferentes domínios administrativos, por organizações virtuais específicas. Se popularizaram devido à prevalência de arquiteturas orientadas a serviço. É dividido nas seguintes camadas:

- Base: provê interfaces para recursos locais em sítios específicos.

- **Conectividade:** provê protocolos de comunicação e de segurança para autenticar usuários e recursos.

- Recursos: gerencia um só recurso, sendo também responsável pelo controle de acesso.

- Coletiva: manipula o acesso a múltiplos recursos, descobrindo-os, alocando-os e escalonando-os. Pode consistir em muitos protocolos.

- Aplicação: contém as aplicações que funcionam dentro de uma organização virtual.

Sistemas distribuídos de informação

Sistemas de processamento de transações: usa conceitos de transações ACID, incluso suas operações (READ, WRITE, BEGIN_TRANSACTION), para fornecer serviços a clientes. Com frequência, os Sds valem-se de subtransações, que podem ser executadas cada uma em um lugar diferente para maior desempenho.

Integração de aplicações empresariais: as aplicações cliente e servidor usam um *middleware* para comunicação, ou o RPC (aplicação), ou o RMI (objetos.) Nestes, um componente da aplicação ou do objeto envia uma requisição a um outro componente executando uma chamada de procedimento local, o que resulta no empacotamento da requisição como uma mensagem e em seu envio ao chamador. Da mesma forma, o resultado é enviado de volta, e devolvido à aplicação como resultado da chamada de procedimento.

As desvantagens destes é que ambos chamador e chamado precisam estar em funcionamento quando da comunicação, e precisam saber como se referir um ao outro.

Sistemas distribuídos pervasivos

Não necessariamente estas características se aplicam a todos os dispositivos, mas são: componentes pequenos, alimentados por bateria, móveis, e usando só comunicação sem fio. Por conta disto, precisam ter ciência de que seu ambiente pode mudar o tempo todo; deve, portanto, **adotar mudanças contextuais**. A configuração do conjunto de aplicações que executa em um dispositivo deve ser fácil (**incentivar composições ad hoc**.) No mais, precisam **reconhecer compartilhamento como padrão**, já que se juntam a outros sistemas para acessar ou fornecer informações.

Redes de sensores: são utilizadas para processar informações. Eficácia vem em primeiro lugar, por conta das limitações técnicas. Em SDs, funcionam como BDs distribuídos.

CAPÍTULO II – ARQUITETURA

Estilos Arquitetônicos

Em camadas: componentes mais acima na hierarquia podem fazer requisições àqueles mais abaixo, mas não o contrário. O fluxo de requisições vai de cima para baixo, e o de respostas, de baixo para cima. A camada OSI ou a arquitetura TCP/IP são exemplos.

Baseada em objetos: cada componente é um objeto, que é conectado aos outros por meio de chamadas de procedimento remotas. Usada em sistemas de software de grande porte; se ajusta bem ao modelo cliente-servidor.

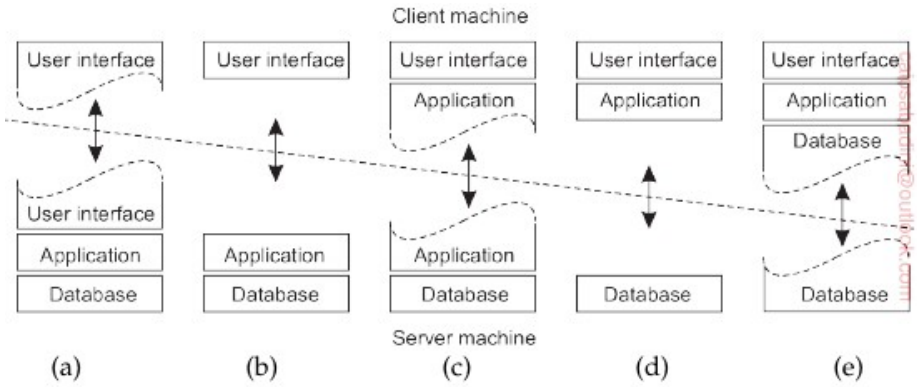
Centrada em dados: processos comunicam-se por meio de um repositório comum passivo ou ativo. Aplicações em rede que dependem de arquivos compartilhados são um exemplo. SDs baseados na Web também usam essa arquitetura, porque processos se comunicam por meio da utilização de serviços baseados na Web.

Baseada em eventos: processos se comunicam por meio de propagação de eventos, que podem também transportar dados. Esta propagação de eventos está associada geralmente a sistemas publicar/escrever, em que processos publicam eventos, e só os processos que subscreveram para estes eventos os receberão. Nestes sistemas, processos são referencialmente desacoplados, ou seja, eles não precisam se referir explicitamente uns aos outros.

Espaços compartilhados de dados: combinação das arquiteturas baseadas em eventos com aquelas centradas em dados. Nestes, os processos são desacoplados também no tempo, ou seja, não precisam estar ambos ativos quando ocorrer a comunicação.

Arquitetura de Sistemas

Centralizadas: usa o modelo cliente-servidor. O servidor é um processo que implementa um serviço específico. O cliente é um processo que requisita serviços ao servidor, e espera por sua resposta. Para evitar falhas de conexão a longa distância, é recomendável o uso de protocolos orientados à conexão (TCP), embora isto seja desnecessário em redes locais devido a sua lentidão. Há casos em que um servidor pode agir de forma intermediária, funcionando também como o cliente para fazer requisições a outro servidor. Exemplo: um cliente solicita os dados de um BD a um servidor http, que por sua vez os solicita ao servidor do BD propriamente dito.



Multidivididas: há três níveis lógicos distintos: interface, aplicação e banco de dados. Há cinco formas de separar estes níveis entre um cliente e um servidor:

a. Parte da interface (dependente do terminal) no cliente; resto da interface (controle sobre apresentação de dados), toda aplicação e BD no servidor. Exemplo: terminais burros.

b. Toda interface no cliente; toda aplicação e BD no servidor.

c. Toda interface e parte da aplicação no cliente; resto da interface e todo BD no servidor. Exemplo: quando se verifica um formulário com JS antes de enviá-lo, ou de um editor de textos que opera seus dicionários/ortografias em um servidor remoto.

d. Toda interface e aplicação no cliente; BD no servidor. É o mais comum nos PCs que se conectam a um BD ou sistema de arquivos por meio da rede.

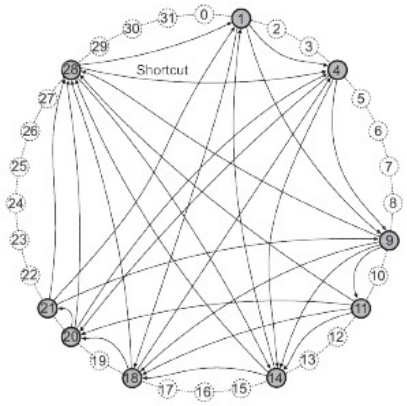
e. Toda interface, aplicação, e parte do BD no cliente; resto do BD no servidor. É o caso de caches de páginas web. Este último caso, o de ter clientes cada mais vez gordos, é a tendência atual.

Arquiteturas Descentralizadas

Peer-to-peer: suporta distribuição horizontal, em que os clientes e servidores podem ser subdivididos em partes logicamente equivalentes. Cada parte opera sua própria porção do conjunto de dados completo, o que equilibra a carga. Por isto, grande parte da interação entre processos é simétrica: um processo age como cliente e servidor simultaneamente. Por isto, processos se organizam em uma **rede de sobreposição**, cujos enlaces representam canais de comunicação possíveis entre os processos.

Tabela de Hash Distribuída: procedimento mais utilizado para construção estruturada da rede de sobreposição. Nela, os itens recebem, de um grande espaço de identificadores, uma chave aleatória de 128 ou 160 bits.

Ao consultar um item dado, o endereço de rede do nó responsável por aquele item é retornado. O **chord** é um sistema que permite esta possibilidade. Ele organiza os nós em forma de anel. Se há um anel real 1, e os nós virtuais 2 e 3, e o outro nó real 4, é o nó real 4 quem atenderá requisições aos nós 2 ou 3 que cheguem para o nó 1. Imagine, por exemplo, que a chave aleatória seja de 4 bits, e que o anel seja como abaixo:



Nesta imagem, requisições feitas ao nó 1 que devem ser atendidas pelo 2 e 3 são, na verdade, atendidas pelo 4, pois:

$$1+2^0=2 \Rightarrow \textit{será atendida pelo } 4\textit{, pois } 4 \textit{ é o sucessor do nó } 1 \textit{ que é} \geq \textit{ao resultado}$$
$$1+2^1=3 \Rightarrow \textit{será atendida pelo } 4\textit{, pois } 4 \textit{ é o sucessor do nó } 1 \textit{ que é} \geq \textit{ao resultado}$$
$$1+2^2=5 \Rightarrow \textit{será atendida pelo } 9\textit{, pois } 9 \textit{ é o sucessor do nó } 1 \textit{ que é} \geq \textit{ao resultado}$$
$$1+2^3=9 \Rightarrow \textit{será atendida pelo } 9\textit{, pois } 9 \textit{ é o sucessor do nó } 1 \textit{ que é} \geq \textit{ao resultado}$$
$$1+2^4=17 \Rightarrow \textit{será atendida pelo } 18\textit{, pois } 18 \textit{ é o sucessor do nó } 1 \textit{ que é} \geq \textit{ao resultado}$$

Para nós maiores, é necessário fazer o resto da divisão para achar o nó que atenda a requisição:

$$18+2^0=19 \Rightarrow \text{será atendida pelo } 20$$
$$18+2^1=20 \Rightarrow \text{será atendida pelo } 20$$
$$18+2^2=22 \Rightarrow \text{será atendida pelo } 28$$
$$18+2^3=26 \Rightarrow \text{será atendida pelo } 28$$
$$18+2^4=34 \Rightarrow \text{será atendida pelo } 4, \text{ pois } 34\,\%\,32=2, \text{ e o nó } 4 \text{ atende ao } 2$$

CAPÍTULO III – PROCESSOS

Threads

Por processo, múltiplas threads de controle proporcionam uma granularidade fina, o que facilita a construção de aplicações distribuídas, além de ter melhor desempenho.

Atrasos de viagem de ida e volta podem ser compensados com threads, como é o caso de requisições web, em que um documento e seus elementos são buscados paralelamente e exibidos conforme suas partes vão sendo recebidas.

Virtualização

Definição: trata de estender ou substituir uma interface existente, de modo a imitar o comportamento de outro sistema. Surgiu para permitir executar software herdado em mainframes caros.

Virtualização é útil em SDs por permitir que se diminua a diversidade de plataformas e máquinas, ao permitir que cada aplicação em sua própria máquina virtual. Com isso, ganha-se portabilidade e flexibilidade, conceitos importantes em SD.

Arquiteturas de Máquinas Virtuais

A virtualização pode ser realizada de quatro formas diferentes, por conta das diferentes interfaces oferecidas pelos sistemas; são estas:

a. Instruções de máquina evocadas por programas (interface entre hardware e software.)

b. Instruções de máquina evocadas *apenas* por programas *privilegiados*, como os SOs (interface entre hardware e software.)

c. Chamadas de sistema.

d. Interface de chamadas de biblioteca (API.)

Máquina Virtual de Processo: um dos modos de realizar a virtualização é construir um sistema que forneça instruções abstratas, utilizadas para execução de aplicações. Tais instruções podem ser interpretadas (Java) ou emuladas (Wine.)

Monitor de Máquina Virtual (VMM): há uma camada que protege completamente o hardware. Para acessá-lo, esta camada oferece um conjunto de instruções completas do hardware ou de outro. Com isto, pode-se ter vários SOs diferentes executando independente e concorrentemente na mesma plataforma. É o caso do VMWare ou da VirtualBox. Por conta destas características, o isolamento faz com que falhas ou ataques à segurança não afetem mais toda máquina. Além disso, pode-se mover ambientes completos de uma máquina a outra.

Servidores

Iterativo: são os próprios servidores que manipulam as requisições, e retornam uma resposta ao cliente requisitante.

Concorrente: passa as requisições para um thread separado, ou para outro processo. Imediatamente depois, espera pela próxima requisição. É o caso de servidores multithread. Pode-se, também, bifurcar um novo processo para cada requisição.

Os servidores possuem *daemons*, aplicações específicas para atender a serviços específicos, que ouvem uma porta a fim de atender requisições a esta.

O envio de dados fora da banda ajuda na interrupção do servidor. Basta enviar uma solicitação de interrupção na porta de controle, que é ouvida com mais prioridade do que a porta de dados.

Servidor sem estado: não mantém informações sobre o estado de seus clientes, e muda seu estado sem ter de informar a nenhum cliente. É o caso de servidores web.

Servidor com estado: mantém informações persistentes sobre seus clientes. Um problema é que precisam recuperar todo seu estado após uma falha. É o caso de servidores FTP.

Cookie: pequena porção de dados que contêm informações específicas dos clientes que interessam ao servidor. É uma maneira de lidar com a ausência de estado. Por ficarem na máquina do cliente, podem representar uma grande falha de segurança.

Clusters de Servidores

É dividido logicamente em três camadas: o comutador, os servidores de aplicação e os de arquivos ou bancos de dados.

Comutador: deve ser capaz de distinguir serviços, a fim de encaminhar solicitações ao processador de aplicação correto. Oferece transparência de acesso. É implementado em hardware. Devido a maneira como funciona, é também um balanceador de cargas. É bom ter mais de um ponto de acesso afinal, se só há um comutador e ele falha, todo *cluster* fica indisponível.

Ele funciona por fazer um TCP *hand-off*: quando lhe chega uma requisição TCP, ele a transfere à máquina correta, que processará tal requisição.

Ao responder, esta máquina deverá fazer *spoofing*: como a requisição do cliente espera uma resposta do IP do comutador, a máquina deve substituir, nos pacotes de resposta, seu IP pelo IP do comutador, a fim de manter a conexão TCP.

Com IPV6, tornou-se possível conseguir um ponto de acesso estável em servidores distribuídos. Isto aconteceu porque um nó móvel passou a ter uma rede nativa, na qual normalmente reside. Este nó possui um endereço estável (**home address**) associado a ele. A rede estável possui um repassador, chamado **agente nativo**, que cuida do tráfego enquanto nós móveis estão indisponíveis.

Assim, quando um nó móvel se ligar a uma rede externa, ele receberá um endereço externo (**care-of address**), que será repassado ao agente nativo da rede original do nó. Este agente, quando receber requisições ao nó, as repassará para ele por meio do endereço externo. A fim de evitar gargalos no agente nativo, a ferramenta de otimização de rotas do IPV6 é utilizada.

Migração de Código

Razões: processos costumam ser migrados por conta de performance, uma vez que rodar em uma máquina menos carregada pode ser mais eficiente. Às vezes, a migração também é útil para

processar os dados próximo do lugar onde estes dados estão, fisicamente. Flexibilidade é ainda outra razão, porque, com a possibilidade de mover códigos entre máquinas diferentes, torna-se possível configura SDs dinamicamente.

Modelos: migrar código trata da movimentação de programas entre máquinas. Destarte, mesmo o estado de execução de um programa, sinais pendentes e outras partes do ambiente podem ter de ser transferidos. Por conta disto, pode-se dividir um processo em três partes lógicas:

a. **Segmento de código**: contém o conjunto de instruções do programa em execução.

b. **Segmento de recursos:** contém referências a recursos externos, de que o processo necessita (arquivos, impressoras, placas de vídeo.)

c. **Segmento de execução:** armazena o estado de execução de um processo. Trata-se de dados privados, da pilha do programam e de seu program-counter.

Além disso, a depender de quanto do processo se quer mover, pode-se ter três tipos de mobilidade:

a. **Fraca:** transfere apenas o segmento de código. É a mais simples, por requerer apenas portabilidade de código, e nada mais.

b. **Forte:** o segmento de execução também pode ser transferido. Um processo em execução pode ser parado e movido para outro lugar, retomando a execução no ponto em que ele a deixou.

c. **Iniciada pelo remetente:** não-relacionada com as acima. A migração, aqui, é iniciada na máquina cujo código está em execução. Normalmente, ocorre para transferir programas a um servidor dedicado à computação pesada.

d. **Iniciada pelo destinatário:** é a máquina que deseja receber o processo que inicia a migração.

Toda forma, em vez de transferir processos, pode-se, simplesmente, clonar estes processos e executá-los na máquina desejada.

Migração e Recursos Locais

Muitas vezes, o segmento de recursos não pode ser simplesmente transferido sem ser trocado. Considera-se três tipos de vinculações entre processos e recursos:

a. **Por identificador:** caso o processo se referencie a URLs ou a FTPs por seus endereços de internet. É a mais forte vinculação.

b. **Por valor:** caso o processo dependa de bibliotecas padronizadas, as quais devem constar na máquina-alvo.

c. **Por tipo:** a mais fraca, ocorre quando o processo especifica apenas que precisa de um tipo de recurso (teclado, monitor, impressora), sem especificar exatamente qual é.

Por conta disto, quando um processo for migrado, é necessário ver se seus recursos poderão ser migrados também. Sob esta ótica, há três tipos de recursos:

a. **Não-ligado:** normalmente, são arquivos de dados, e podem ser movidos facilmente, desde que estejam somente ligados ao programa transferido, e a nenhum outro.

b. **Amarrados:** é o caso de BDs ou *sites* web completos. É possível, mas difícil, movê-los; isto se deve ao alto custo desta operação migratória.

c. **Fixos:** são recursos intimamente vinculados às máquinas em que estão. Quase nunca podem ser movidos, por serem literalmente *fixos*. São, geralmente, dispositivos locais.

Assim, relacionando a vinculação processo-recurso com os tipos de recursos possíveis, a tabela abaixo demonstra quais operações são possíveis em cada caso:

	Não-ligado	**Amarrado**	**Fixo**
Por identificador	MV ou RG	RG ou MV	RG
Por valor	CP ou MV ou RG	RG ou CP	RG
Por tipo	VR ou MV ou CP	VR ou RG ou CP	VR ou RG

MV: Mover o recurso junto ao processo.

RG: Fazer uma referência global ao recurso, a fim de acessá-lo pela máquina-alvo. Às vezes, o preço de criação ou uso (pelo tráfego da rede) desta referência global é proibitivo.

CP: Copiar o valor do recurso, como no caso de arquivos acessados por mais de um processo.

VR: Vincular o processo a um recurso disponível no local, quando, por exemplo, um processo exige uma impressora, basta vinculá-lo à impressora da máquina alvo.