**Sistema Distribuido**

É composto por instruções concorrentes ou paralelas, que são interpretadas e executadas por um ou mais processadores.

**Por que distribuir?**

-Manutenção

-Acomodação as capacidades das maquinas disponiveis

-Melhor desempenho e escalabilidade

-Tolerância a falhas

**Infra-Estrutura de Software**

Software de sistema responsável pelo controle direto e gerência do hardware e operações básicas de sistema.

**Requisitos de sistemas distribuidos**

Comunicação em redes – controle de sessões

Coordenação – sincronização

Confiabilidade – transmissões, replicação

Escalabilidade – transparência de localização, replicação

Heterogeneidade – hardware, plataformas de SO, linguagens de programação

**Sistemas distribuidos: conceitos e arquiteturas**

Sistemas centralizados: grande porte fisico, grande consumo de energia e SO único (dependência do fabricante)

Sistemas em rede: Computadores diversos todos com capacidade de processamento, portes diversos, SO’s diversos e redes diversas.

Sistema distribuido: componentes interligados, processamento distribuido ou paralelo

**Em resumo sistemas distribuidos são:**

-varios componentes

-conectados via uma rede

-compartilhando recursos

-transparência

**Por que construir SDs?**

-Modelar o mundo real: pessoas são distribuidas, informação é distribuida

-Relação desempenho/custo

-Modularidade

-Expansibilidade: são capazes de crescimento incremental

-Disponibilidade: têm capacidade de replicação e redundância

-Escalabilidade: idealmente, SDs não tem qualquer componente centralizado

-Confiabilidade: o sistema deve ser capaz de se recuperar de falhas e “Disponibilidade”

**Conceito-chave**

Distribuição, Comunicação, Complexidade, Heterogeneidade...

Demandam >>>> TRANSPARÊNCIA

**Tipos de Transparência**

-Localização: esconde onde o recurso está localizado

-Acesso: operações idênticas para acesso local e remoto

-Migração: esconde que um recurso pode mudar de lugar

-Relocação: esconde que um recurso pode mudar de lugar enquanto está em uso

-Concorrência: compartilhamento de recursos sem interferência de processos concorrentes

-Falha: esconde a falha e recuperação de um recurso

-Replicação: esconde a existência de replicas de recursos

**Sistemas operacionais para computadores distribuidos**

-Fortemente acoplados: tentam manter visão unica e global dos recursos gerenciados

-Fracamente acoplados: coleção de computadores, cada um executando seu próprio SO. No entanto trabalham juntos para tornar os serviços e recursos de uns disponiveis aos outros.

-SOD: fortemente acoplado, tenta esconder e gerenciar recursos de hardware.

-SOR: fracamente acoplado, oferece serviços locais para clientes remotos.

-Middleware: Uma camada adicional que provê transparência de distribuição.

**Tipos de SOD’s**

-SO multi-processador

-alto desempenho através de multiplos processadores(numero de cpu’s deve ser tranparente para a aplicação)

-memória compartilhada

-proteção contra acesso concorrente através de sincronização

-SO multi-computador

-não há compartilhamento de memória

-comunicação confiável através do middleware

**SOR**

-serviços estão disponiveis em maquinas distintas

-login remoto

-cópia remota (é preciso saber onde maquinas se localizam)

- pouca transparência

**Middleware**

Escalabilidade e abertura – SORs

Transparência e uso mais facil – SODs

Serviços agregados:

-facilidade de comunicação de alto nivel: transparência de acesso

-nomeação: transparência de localização

-persistência: através de um sistema de arquivo distribuido

-transações distribuidas: propriedades “ACID”

-segurança

**Definição formal de middleware**

Um conjunto reusável, expansível de serviços e funções, que são comumente necessários por parte de várias aplicações distribuidas, para funcionarem bem em um ambiente de rede.

**Elementos do middleware**

-Portabilidade

-Interoperabilidade: é a capacidade de um sistema se comunicar de forma transparente com outro sistema.

**Middleware orientado a objeto**

-Metodos e dados remotos

-Fornece referências para objetos

-Serviços

**Benefícios do middleware**

-Simplificação da programação distribuida: serviços

-Integração de sistemas heterogêneos

-Abstração do sistema de transporte: interoperabilidade, portabilidade

**Transmissão de dados**

Dados em programas são estruturados, porém na comunicação as mensagens devem ter um formato externo comum (XDR) para que não haja inconsistência.

Marshalling: faz a linearização dos dados estruturados (coloca em XDR)

Unmarshalling: traduz do XDR para o formato local

**Protocolo Request-Reply**

Típico na interação cliente servidor

-DoOperation: clientes invocam operações remotas

-GetRequest: servidor adquire as requisições

-SendReply: servidor envia o resultado

**Remote Procedure Call (RPC)**

Objetiva permitir chamada de procedimento remoto como se fosse local.

-evitar passagem de endereços e variaveis globais: pois o endereço de memória do seu computador pode representar algo diferente no servidor

-tratamento de exceções: timeout

-Stubs(cliente-servidor): transparência de acesso, provê a forma de chamada remota semelhante a de local. Trata algumas exceções, utiliza marshalling e unmarshalling.

-Utiliza o protocolo Request-Reply para troca de mensagens entre cliente e servidor

-Utiliza um Binder para resolução de nomes (ligação dinâmica, transparência de localização)

**RPC tradicional x RPC assíncrono**

-no tradicional quando o cliente faz uma chamada, ele tem que esperar a resposta do servidor para poder continuar a fazer outras coisas.

-no assíncrono o cliente não precisa esperar, enquanto o resultado não chega ele pode trabalhar em outras coisas.