

Sistemas Distribuídos (COS470)

<http://www.gta.ufrj.br/cos470>



José Ferreira de Rezende
rezende@gta.ufrj.br

Ementa do Curso



- Conceitos Fundamentais de Sistemas Distribuídos
- Paradigmas de Sistemas Distribuídos
- Definições de Processos e Threads
- Comunicação em Sistemas Distribuídos
- Sincronização em Sistemas Distribuídos
- Conceitos de Middleware
- Redes P2P: conceitos básicos, arquiteturas, aplicações
- Introdução a Grades Computacionais
- Tecnologias de Middleware Tradicionais
- Middlewares de Nova Geração

Bibliografia



- G. Coulouris, J. Dollimore e T. Kindberg, "Sistemas Distribuídos: Conceitos e Projetos", 4a edição.
- A.S. Tanenbaum and M.V. Steen, "Distributed Systems: Principles and Paradigms", 2nd Edition.
- S. Ghosh, "Distributed Systems: An Algorithmic Approach".

Sistemas Distribuídos

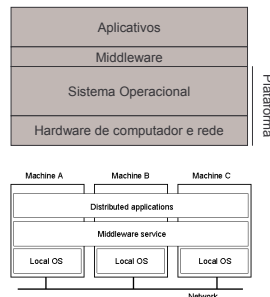


- **definição:** coleção de computadores independentes que se apresenta para os usuários como um único sistema coerente.
- **características desejáveis**
 - transparente
 - aberto
 - escalável
 - seguro
 - tolerante à falhas
- **tipos:** computacionais, pervasivos e de informação

Definição de Sistemas Distribuídos

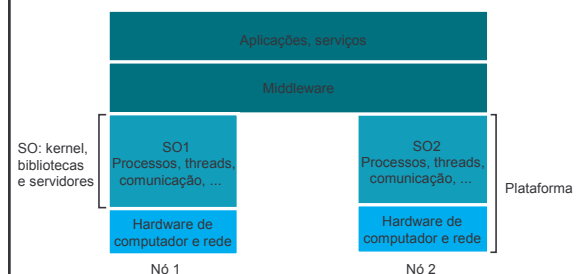


- **middleware**
 - camada de software que fornece uma abstração de programação que mascara a heterogeneidade das redes, do hardware, de SOs e linguagens de programação subjacentes
 - ex. CORBA, Java RMI, serviços web, ODP e DCOM
 - fornece um modelo de computacional uniforme para ser usado pelos programadores de aplicativos distribuídos
 - invocação remota de objetos
 - notificação remota de eventos
 - acesso remoto a banco de dados
 - processamento distribuído de transações




fonte: slides do livro do Tanenbaum

Camadas do Sistema




fonte: slides do livro do Coulouris



Tipos de Sistemas Distribuídos


- Sistemas Computacionais
 - Clusters
 - Grades
- Sistemas de Informação
- Sistemas Pervasivos (Ubíquos)



Transparência em Sistemas Distribuídos


| Transparência | Descrição |
|----------------|-------------------------------------------------------------------------------------------|
| Acesso | Oculta diferenças na representação de dados e no modo de acesso ao recurso |
| Localização | Oculta onde o recurso está localizado |
| Escalabilidade | Oculta que um sistema e os aplicativos se expandam em escala |
| Mobilidade | Oculta que um recurso pode ser movido para uma outra localização enquanto em uso |
| Replicação | Oculta que um recurso pode ser replicado |
| Concorrência | Oculta que um recurso pode ser usado por diversos usuários concorrentes |
| Falha | Oculta a falha e a recuperação de um recurso |
| Desempenho | Oculta reconfigurações do sistema para se adaptar a carga variável, mantendo o desempenho |

Diferentes formas de transparência num sistema distribuído (ISO/Tanenbaum/Coulouris)




Modelos de Arquitetura

- cliente-servidor
- peer-to-peer
- variações
 - serviços fornecidos por vários servidores
 - alta disponibilidade, balanceamento de carga, processamento paralelo e alto desempenho
 - servidores proxies e cache
 - maior disponibilidade e desempenho, menor carga nos servidores e rede remotos
 - código móvel
 - applets
 - agentes móveis
 - código+dados que transitam/migram entre computadores de uma rede

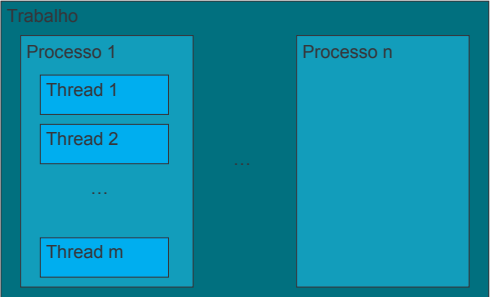



Processos e Threads

- um processo consiste em um ambiente de execução, junto com uma ou mais threads
 - thread é uma abstração do S.O. de uma atividade (ou fio de execução)
 - o ambiente de execução é a unidade de gerenciamento de recursos (conjunto de recursos locais gerenciados pelo núcleo do S.O.) aos quais suas threads têm acesso
 - espaço de endereçamento
 - recursos de comunicação e sincronização entre threads
 - recursos de mais alto nível, tais como arquivos e janelas abertas
- analogia de Chris Lloyd (jarro com ar e alimento + moscas)



Trabalho, Processos e Threads





Programação Multi-threading

- por que?
 - máquinas com múltiplos cores/CPU's se tornaram comuns
- quando
 - tarefas bem definidas e longas suficientes
 - dados usados para completar as tarefas não são (ou são pouco) compartilhados
 - sem interação com o usuário (*background*)

Processos versus Threads



- **processo**
 - programa em execução
 - S.O. fornece total proteção e torna transparente o compartilhamento de recursos por múltiplos processos
 - alto grau de transparência de concorrência -> degradação de desempenho (na comutação de processos)
- **thread**
 - um mínimo de informação é mantido para permitir o compartilhamento de uma CPU por múltiplas threads
 - exige um esforço adicional no desenvolvimento
 - threads não são protegidas uma das outras

Vantagens de Múltiplas Threads (S.N.D.)



- em processos com thread única (*single-threaded process*) sempre que uma chamada de sistema bloqueante é executada, o processo como um todo é bloqueado
 - com múltiplas threads, algumas threads podem ficar bloqueadas enquanto outras continuam realizando suas tarefas
- permite explorar o paralelismo quando o programa é executado em máquinas com múltiplas CPUs/Cores
 - com o barateamento dessas máquinas, isso torna-se cada vez mais importante
 - um bom projeto: transparência de paralelismo

Vantagens de Múltiplas Threads (S.N.D.)



- em aplicações complexas compostas por uma coleção de programas executados por múltiplos processos, a cooperação entre processos exige o uso de IPC que por sua vez exige comutação de contexto
 - comunicação entre threads é normalmente feita por compartilhamento de dados
- questão de engenharia de software: muitas aplicações são mais facilmente projetadas usando se múltiplas threads
 - aplicações que precisam realizar múltiplas tarefas independentes (ou pouco dependentes)

Implementação de Threads

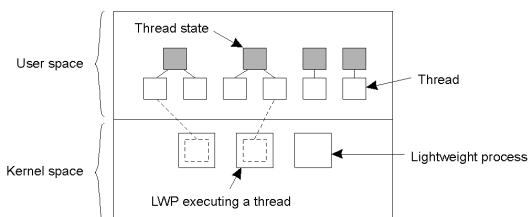


- na forma de um pacote
 - criação/destruição de threads e sincronização entre elas
- espaço usuário (biblioteca)
 - vantagem: desempenho (criação/destruição e comutação de threads)
 - desvantagem: o bloqueio numa chamada leva ao bloqueio de todo o processo
- espaço kernel
 - desvantagens: toda manipulação exige uma chamada ao sistema e a comutação de threads se torna tão lenta quanto a de processos
- híbrido (lightweight processes + user-level library)
 - múltiplos LWPs por processo que executam threads
 - pacote no espaço usuário para manipulação de threads sem intervenção do kernel

Implementação de Threads



- kernel-level lightweight processes + user-level threads.



fonte: slides do livro do Tanenbaum

Construindo um servidor



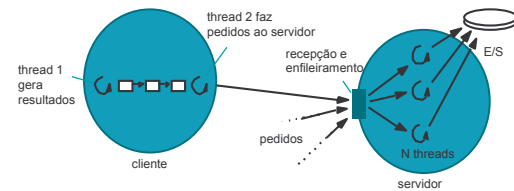
| Modelo | Características |
|-----------------------------------------|-----------------------------------------------------------------------------|
| múltiplas threads | paralelismo (usando processos sequenciais), chamadas ao sistema bloqueantes |
| thread única | sem paralelismo, chamadas ao sistema bloqueantes |
| máquina de estados finita (evento-ação) | paralelismo, chamadas ao sistema não-bloqueantes |

Multithreading em S.D.



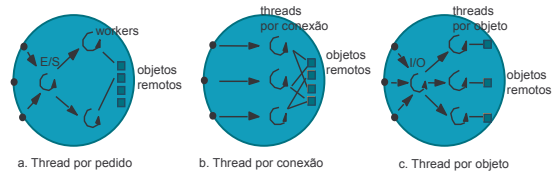
- múltiplas threads permite realizar mais facilmente uma comunicação na forma de múltiplas conexões lógicas ocorrendo simultaneamente
- bastante usado na implementação de servidores
 - simplifica o código
 - explora paralelismo, maximizando taxa de rendimento

Cliente e Servidor com Threads



fonte: slides do livro do Coulouris

Arquiteturas de Servidores Multi-threaded

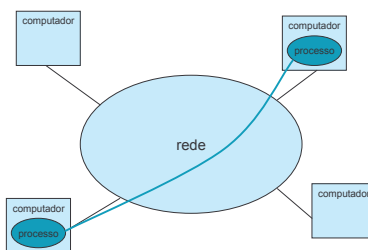


fonte: slides do livro do Coulouris

Comunicação Inter-Processo (IPC)



- procedimento através do qual processos trocam informações e coordenam suas atividades



Modelos de Comunicação



- básico: troca de mensagens (*message-passing*)
 - orientado a mensagens
 - orientado a fluxo (*stream*)
 - comunicação de grupo (*multicasting*)
- de mais alto nível
 - Remote Procedure Call (RPC)
 - comunicação cliente-servidor
 - Remote Method Invocation (RMI)
 - comunicação entre objetos distribuídos

Camadas de Middleware



fonte: slides do livro do Coulouris

Tipos de Comunicação (1)



- **síncrona**
 - processos remetente e destino são sincronizados a cada mensagem
 - operação de *send* causa bloqueio
 - processo remetente desbloqueia apenas quando:
 - a mensagem é passada ao middleware
 - a mensagem é recebida pelo processo destino
 - a mensagem é processada pelo destino que então retorna uma resposta
- **assíncrona**
 - operação *send* é não-bloqueante
 - mensagem é copiada para um buffer local
 - transmissão em paralelo com o processamento no remetente
 - operação *receive* bloqueante/não-bloqueante
 - processo bloqueado enquanto mensagem não chega
 - processo continua execução podendo ser notificado (p.ex. por uma interrupção) da chegada de uma mensagem

Tipos de Comunicação (2)

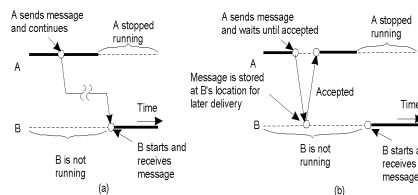


- **persistente**
 - a mensagem transmitida é armazenada pelo sistema de comunicação até que ela seja entregue ao receptor
 - não é necessário que a aplicação transmissora continue executando após ter transmitido a mensagem e a aplicação receptora não precisa estar executando quando da transmissão da mensagem
 - ex. sistema de correio eletrônico
- **transiente**
 - a mensagem é armazenada pelo sistema de comunicação somente enquanto as aplicações transmissora e receptora estão executando
 - ex. serviços de comunicação no nível de transporte

Persistência e Sincronismo na Comunicação



- a. comunicação persistente assíncrona
- b. comunicação persistente síncrona

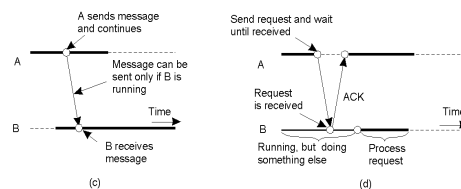


fonte: slides do livro do Tanenbaum

Persistência e Sincronismo na Comunicação



- c. comunicação transiente assíncrona
- d. comunicação transiente síncrona (receipt-based)

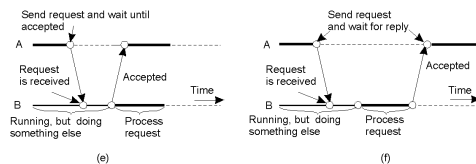


fonte: slides do livro do Tanenbaum

Persistência e Sincronismo na Comunicação



- e. comunicação transiente síncrona (delivery-based)
- f. comunicação transiente síncrona (response-based)

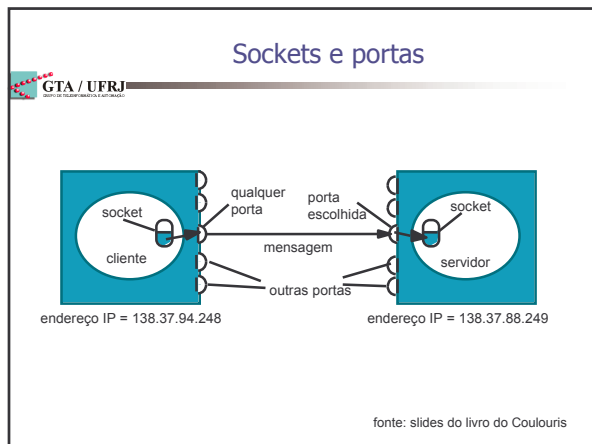


fonte: slides do livro do Tanenbaum

Comunicação via sockets



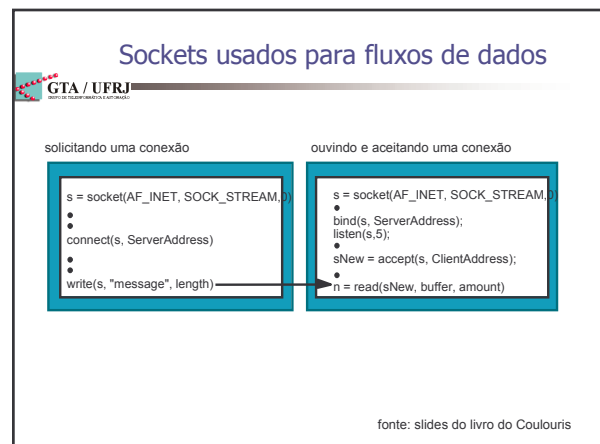
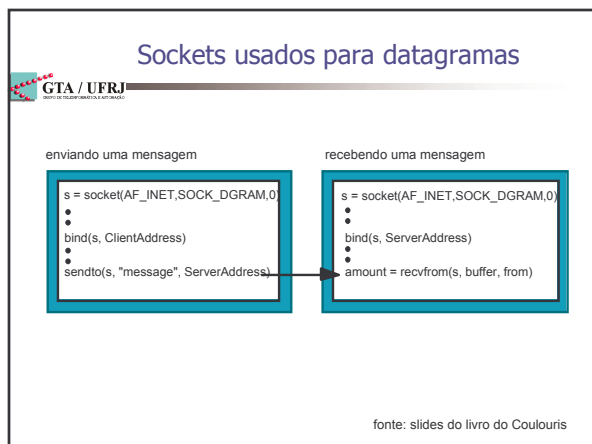
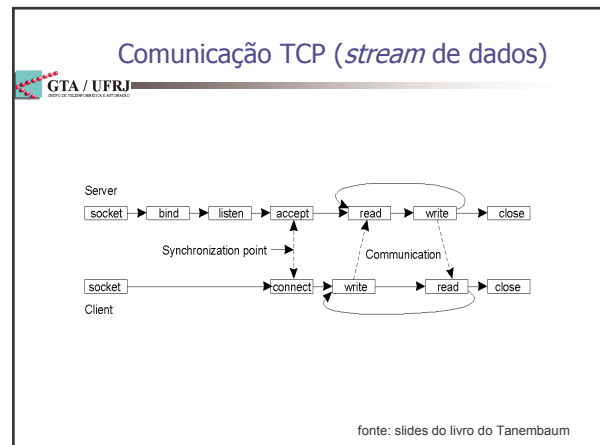
- **transmissão entre o socket de um processo e o socket de outro processo**
 - mensagens enviadas para destinos identificados por <endereço destino, porta destino>
 - cada computador tem 2^{16} portas locais
 - processo não pode compartilhar uma porta com outros processos (exceção multicast)
- **para um processo receber mensagens, o seu socket deve estar associado a uma porta local e a um dos endereços IP do computador em que é executado**



- ## Comunicação UDP
- sem confiabilidade
 - mensagens perdidas: problema na comunicação e transbordo de buffer
 - mensagens fora de ordem: mudanças no roteamento e escalonamento de pacotes nos roteadores
 - um serviço confiável pode ser construído acima do UDP pela própria aplicação

API Sockets: TCP/IP

| primitiva | significado |
|-----------|---------------------------------------------------------|
| socket | cria um ponto final de conexão |
| bind | associa um endereço local ao socket |
| listen | anuncia desejo em aceitar conexões |
| accept | bloqueia o chamador até que chegue um pedido de conexão |
| connect | tenta ativamente estabelecer uma conexão |
| send | envia dados |
| receive | recebe dados |
| close | termina a conexão |



Message-Passing Interface (MPI)



- surgiu da necessidade dos desenvolvedores de aplicações para computadores paralelos de alto desempenho de:
 - usar primitivas com um nível mais alto de abstração para facilitar o desenvolvimento de aplicações
 - contar com diferentes formas de bufferização e sincronização
 - usar eficientemente protocolos de alta velocidade proprietários
- comunicação transiente assíncrona e síncrona
- comunicação de grupo (grupo de processos envolvidos numa determinada computação)

Message-Passing Interface (MPI): Algumas primitivas



| primitivas | significado |
|--------------|-------------------------------------------------------------------------------------|
| MPI_bsend | coloca mensagem num buffer local de emissão |
| MPI_send | envia uma mensagem e espera até que ela seja copiada para um buffer local ou remoto |
| MPI_ssend | envia uma mensagem e espera até que recepção inicie |
| MPI_sendrecv | envia uma mensagem e espera por resposta |
| MPI_issend | passa referência para mensagem de saída e continua |
| MPI_issend | passa referência para mensagem de saída e espera até que inicie recepção |
| MPI_rcv | recebe a mensagem (bloqueante) |
| MPI_irecv | verifica se chegou alguma mensagem (não-bloqueante) |

Sistema de Enfileiramento de Mensagens (Message-Queuing System)

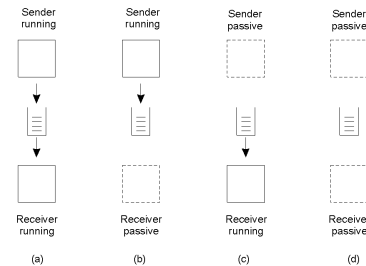


- comunicação persistente assíncrona
- aplicações se comunicam inserindo mensagens em filas, as quais são encaminhadas através de uma série de servidores de comunicação, sendo eventualmente entregues ao destino

Modelo de Enfileiramento de Mensagens



- 4 tipos de comunicação de acordo com acoplamento no tempo entre emissor e receptor



fonte: slides do livro do Tanenbaum

Modelo de Enfileiramento de Mensagens: API básica

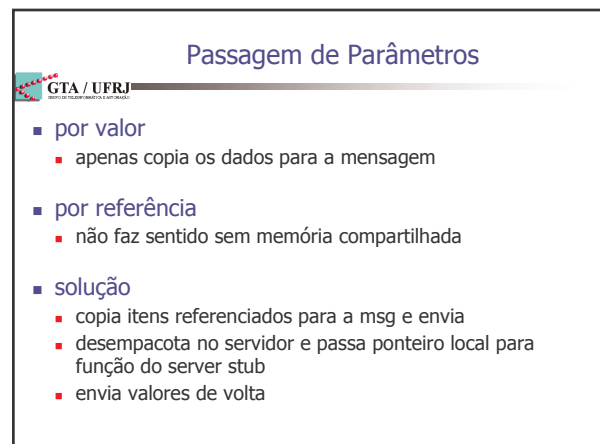
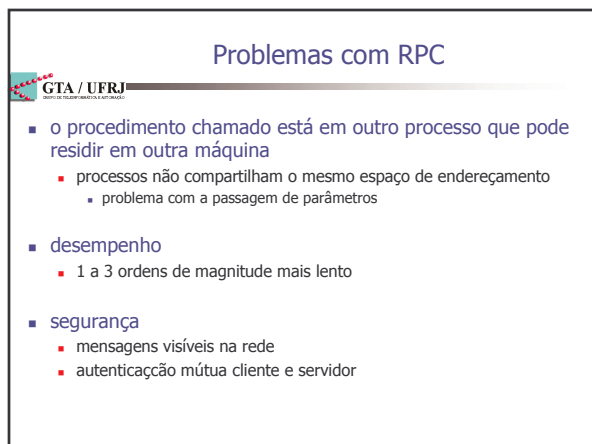
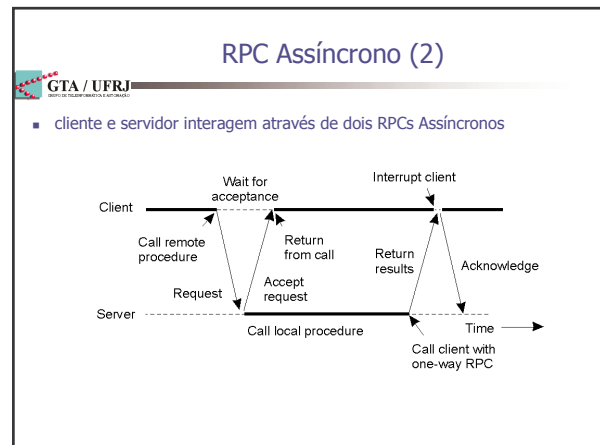
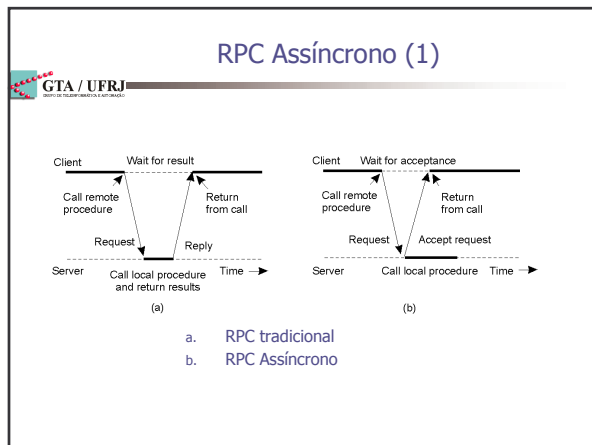
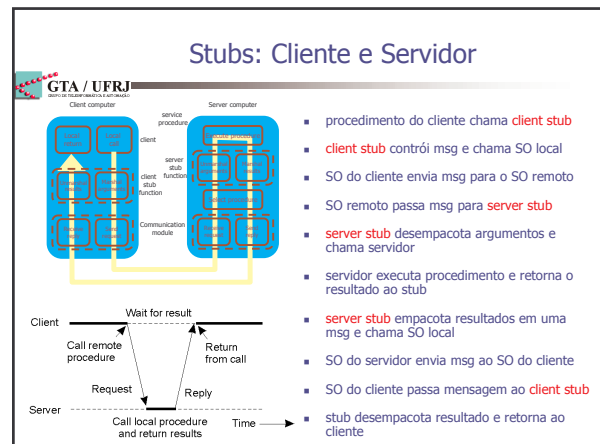
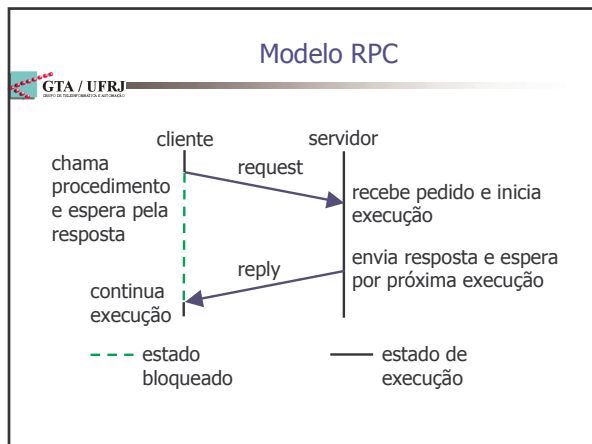


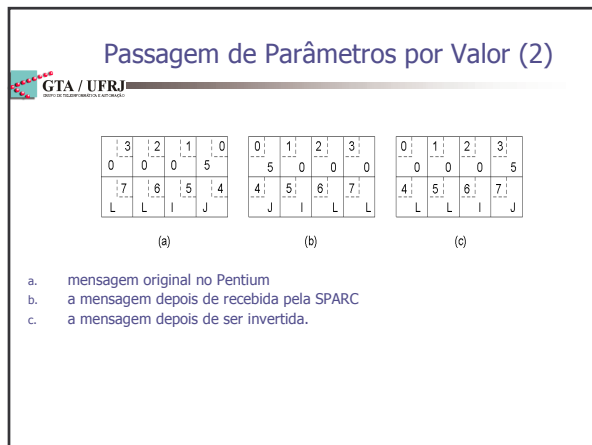
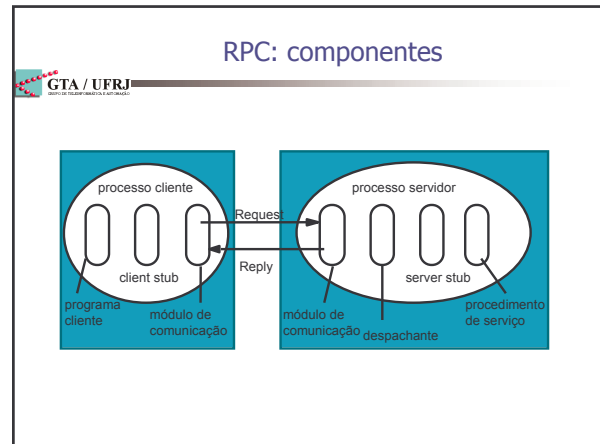
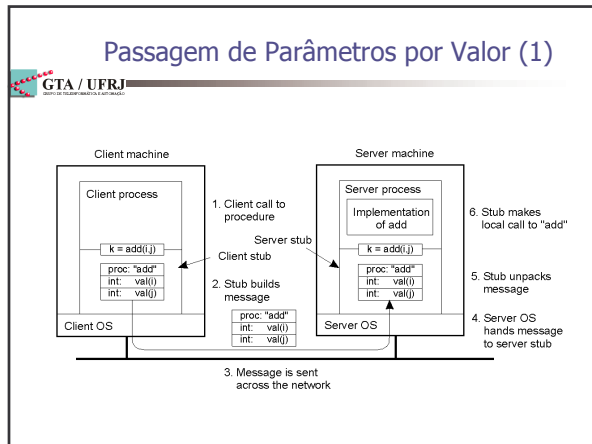
| primitiva | significado |
|-----------|-------------------------------------------------------------------------------------------------|
| put | insere uma mensagem numa determinada fila |
| get | bloqueia até que uma determinada fila está não-vazia, removendo a primeira mensagem |
| poll | verifica se uma fila específica contém mensagens, removendo a primeira (não-bloqueante) |
| notify | instala um <i>handler</i> para ser chamado quando uma mensagem for colocada na fila determinada |

Remote Procedure Call (RPC)



- modelo de alto nível para a comunicação cliente-servidor
- 1984: Birrell & Nelson
 - mecanismos para chamadas de procedimento em outras máquinas
 - processo numa máq. A pode chamar procedimentos na máq. B
 - A é suspensa
 - execução continua em B
 - quando B retorna, o controle é passado de volta para A
- é exatamente como uma chamada normal de procedimento
- vantagem: simplifica o desenvolvimento de aplicações
 - RPC esconde todo o código relacionado à rede
 - programadores não precisam se preocupar com detalhes de sockets, portas, ordenação dos bytes, etc.





- ### Representação dos dados
- assume que sistemas são heterogêneos:
 - diferentes ordenações dos bytes (little endian, big endian)
 - diferentes tamanhos para inteiros e outros tipos
 - diferentes representações de ponto flutuante
 - diferentes conjuntos de caracteres (EBCDIC, ASCII, etc.)
 - requer uma codificação padrão dos dados da aplicação (parâmetros) para permitir a comunicação entre processos remotos
 - tipagem implícita
 - apenas valores são transmitidos
 - SUN RPC: XDR (eXternal Data Representation)
 - tipagem explícita
 - tipos são transmitidos juntamente com os valores
 - ex. ISO ANS.I, XML

- ### Associação entre cliente e servidor (Binding)
- como localizar a máquina e o processo do servidor?
 - solução 1: uso de uma base de dados central (diretório)
 - DCE
 - solução 2: cliente precisa do nome do servidor e o servidor mantém a base de dados
 - Sun RPC

- ### Tratamento de Falhas
- RPC é mais vulnerável a falhas do que chamadas de procedimento locais
 - rede, crash do cliente ou servidor, erros gerados pelo servidor, etc.
 - aplicações devem estar preparadas para falhas do RPC
 - quebra de transparência
 - reenvio de mensagens de pedido
 - cliente retransmite até que chegue uma resposta ou assuma que o servidor falhou
 - servidor deve filtrar mensagens duplicadas
 - retransmissão de respostas:
 - servidor mantém um histórico das mensagens de resposta para evitar que seja necessária a re-execução dos procedimentos do servidor em caso de retransmissão de respostas perdidas

Semântica da Chamada RPC

GTA / UFRJ

- chamadas locais de procedimentos: **exactly-once**
- at-least-once** - aceitável somente se as operações do servidor são idempotentes: $f(x) = f(f(x))$

| Semântica da chamada RPC | Medidas de Tolerância à Falhas | | |
|--------------------------|--------------------------------|-------------------------|------------------------------|
| | Reenvia mensagens de pedido | Filtragem de duplicatas | Resposta |
| MayBe | Não | Não aplicável | Não aplicável |
| At-least-once | Sim | Não | Reexecuta procedimento |
| At-most-once | Sim | Sim | Retransmite resposta (cache) |

Programação com RPC

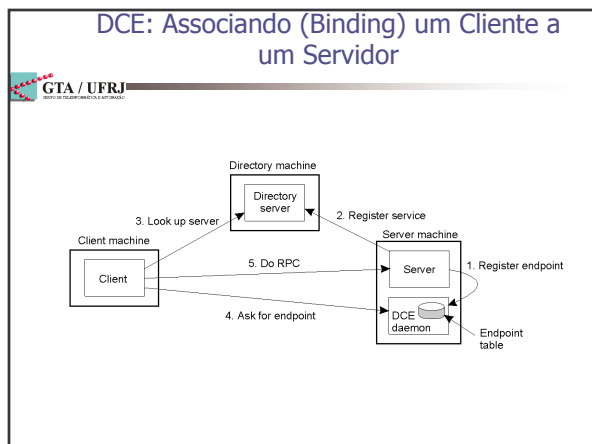
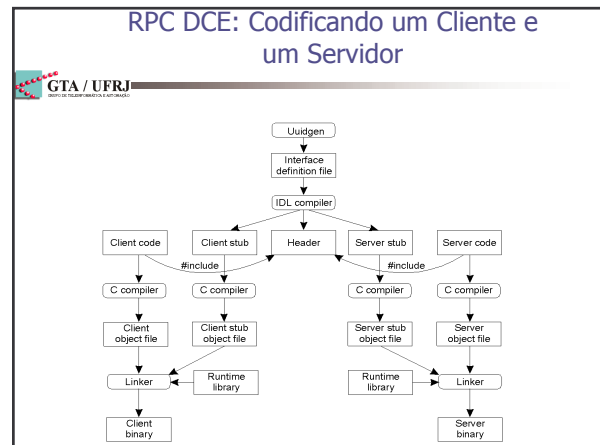
GTA / UFRJ

- suportado por poucas linguagens
 - compiladores não geram client e server stubs
 - solução: compilador separado (pre-compilador) para gerar stubs
- Interface Definition Language (IDL)
 - permite aos programadores especificar interfaces de procedimentos remotos (nomes, parâmetros, valores retornados)
 - usado pelos pre-compiladores para gerar stubs
 - código de (un)marshalling e protocolos de transporte
- exige modificações no código do cliente
 - inicialização de opções relacionadas ao RPC
 - tipo de transporte, localização do serviço e tratamento de falhas
- poucas (ou nenhuma) modificações no servidor

Sun RPC

GTA / UFRJ

- um servidor RPC oferece um ou mais conjuntos de procedimentos
 - cada conjunto é chamado de um programa, sendo unicamente identificado por um número de programa
- portmap
- marshalling/unmarshalling
 - formato XDR (*eXternal Data Representation*)
- pre-compilador: rpcgen
 - http://www.cisco.com/en/US/docs/ios/sw_upgrades/interlink/r2_0/rpc_pr/rprpcgen.html
- comunicação: TCP ou UDP
- serviços implementados usando Sun RPC: NIS e NFS



Invocação a Método Remoto Remote Method Invocation (RMI)

GTA / UFRJ

- fornece uma maneira eficiente e transparente de comunicação entre objetos distribuídos, permitindo a computação distribuída orientada a objetos
 - aplicações/sistemas de objetos distribuídos
- similar a RPC, mas com instanciação dinâmica de novos objetos e suas respectivas interfaces
- suportado por CORBA (independente de linguagem) e JAVA
- componentes: referências para objetos, interfaces, ações (métodos), exceções e coleta de lixo (*garbage collection*)

Componentes de um modelo de objetos



- referências a objetos
 - referencia objetos de uma classe
 - podendo ser atribuído e solicitado
- interfaces
- ações
 - provocadas pelas invocações aos métodos
 - se a fronteira de um processo ou computador é ultrapassada, então a RMI será usada para gerar a ação
 - iniciado por um objeto invocando um método em outro objeto
 - efeitos possíveis
 - mudança de estado do invocado
 - podendo desencadear outras invocações a métodos

Componentes de um modelo de objetos



- exceções
 - reações definidas pelo programador e do próprio sistema a situações inesperadas e de erro
 - permite a recuperação de situações de erro
 - problemas específicos devido ao ambiente distribuído
 - servidor pode ter falhado (crash) ou estar muito ocupado para responder dentro de limites de tempo razoáveis
 - exceções específicas
 - temporizadores (*timeouts*)
 - devido a infra-estrutura distribuída subjacente
- coleta de lixo
 - determina se existem referências restantes para um objeto no sistema
 - liberação de recursos (memória) ocupados por objetos que não são mais referenciados e não serão mais necessários no futuro

RMI: Princípio de Base



- a definição de comportamento e a implementação desse comportamento são conceitos distintos
- RMI permite que o código que define o comportamento e o código que o implementa executem em máquinas distintas
 - num sistema distribuído, clientes se preocupam com a definição do serviço e o servidores focam em como prover o serviço
- p.ex. a definição de um serviço remoto é codificado usando uma interface Java enquanto a implementação do serviço é codificado em uma classe Java
 - interfaces definem comportamento e classes definem implementação

Referência para o Objeto



- antes de invocar um método em um objeto remoto é necessário uma referência para o objeto
- existem algumas formas de achar essa informação
 - protocolos de descoberta
 - serviços de nomeação, p.ex. RMI Registry (*rmiregistry*)
 - servidores registram objetos remotos com nomes
 - clientes procuram por referências de objetos que casem com um nome
 - nomes usam o formato URL
 - `rmi://<maq:<port>/<NomeServico>`
 - `rmi://146.164.69.2:3180/Chat`

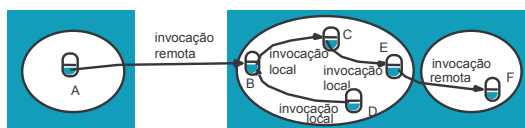
`void rebind(String name, Remote obj)`
This method is used by a server to register the identifier of a remote object by name.

`void bind(String name, Remote obj)`
This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.

`void unbind(String name, Remote obj)`
This method removes a binding.

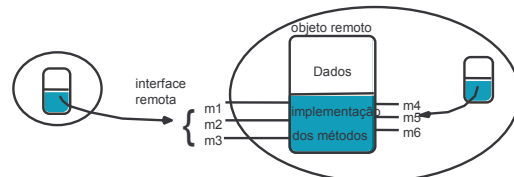
`Remote lookup(String name)`
This method is used by clients to look up a remote object by name. A remote object reference is returned.

Invocações Remotas e Locais

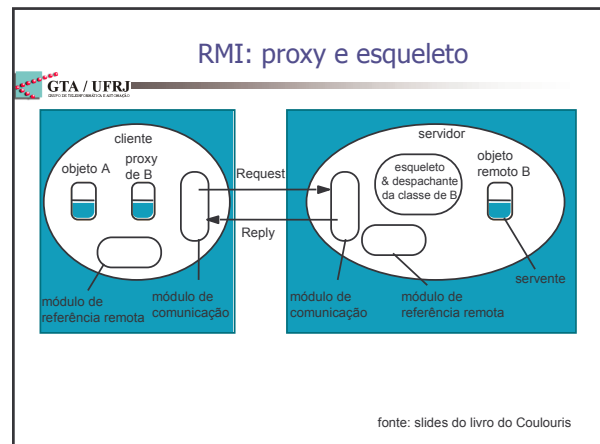
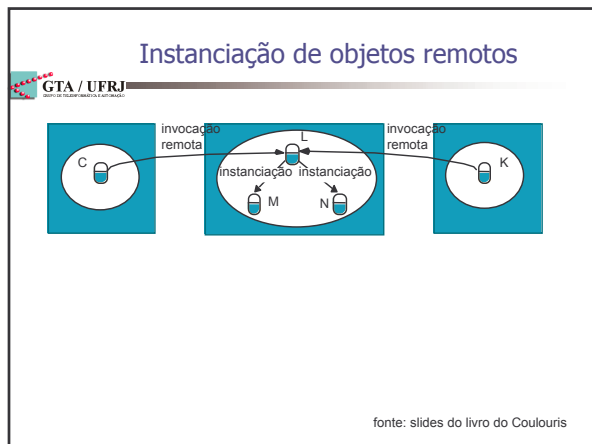


fonte: slides do livro do Coulouris

Objeto remoto e sua interface remota



fonte: slides do livro do Coulouris



- ### Implementação de RMI
- **módulo de comunicação**
 - mensagens request/reply
 - no servidor
 - seleciona despachante para a classe do objeto a ser invocado
 - obtém uma referência local a partir do identificador do objeto remoto usando o módulo de referência remota
 - passa a referência local ao escalonador
 - **módulo de referência remota**
 - translação entre referências locais e remotas usando uma tabela de objetos remotos
 - contém entradas para todos os objetos remotos mantidos pelo servidor
 - e para todos os proxies
 - usado quando (de)codificando referências a objetos remotos

- ### Componentes do RMI
- **proxies (no cliente):** cópias locais de objetos remotos
 - **despachante (no servidor):** recebe pedido e usa methodID para selecionar msgs apropriadas no esqueleto (skeleton)
 - **esqueleto (no servidor):** implementa métodos da interface remota
 - decodifica (unmarshaling) os argumentos
 - invoca método correspondente no objeto remoto
 - espera terminar a execução da invocação
 - codifica (marshaling) os resultados, possíveis exceções, e retorna-os em uma mensagem de reply ao método invocado pelo proxy
 - **implementação**
 - em alguns sistemas, eles podem ser automaticamente compilados:
 - CORBA: usa descrições IDL de interfaces remotas para compilar a camada RMI, gerando código C++
 - Java RMI: classes do proxy, esqueleto e despachante são geradas a partir da classe do objeto remoto

- ### Alguns componentes adicionais da RMI
- **métodos de fábrica (factory methods)**
 - interfaces de objetos remotos não incluem construtores
 - métodos de fábrica (factory methods) substitui construtores na criação de objetos remotos
 - **binders**
 - serviço que mantém o mapeamento de nomes textuais de objetos em referência de objetos remotos
 - **uso de threads no servidor**
 - toda invocação a método remoto é normalmente passada para uma thread para execução
 - vantagens: se a RMI invoca outros métodos locais ou remotos que causam tempos de espera

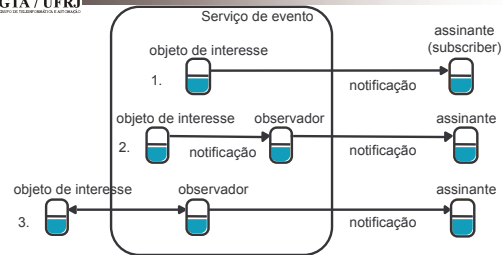
- ### Notificação de Eventos Distribuída
- **idéia**
 - um objeto reage à mudanças que ocorrem em um outro objeto
 - SD baseados em eventos permitem que objetos em diferentes localizações sejam notificados de eventos que acontecem em um determinado objeto
 - **exemplos de eventos**
 - modificações de um documento
 - livro com etiqueta eletrônica entrando numa nova sala
 - modificação do preço de uma ação
 - **paradigma publish/subscribe**
 - o gerador de eventos publica (**publish**) tipos de eventos
 - o receptor de eventos assina (**subscribe**) aos tipos de eventos que ele tem interesse
 - quando um evento ocorre, o receptor é notificado

Sistemas Distribuídos baseados em Eventos



- duas características principais
 - heterogeneidade dos seus componentes
 - assincronismo entre publish e subscriber
- notificações são objetos que contém informações sobre os eventos
 - eventos podem ser de diferentes **tipos**
 - cada evento pode ter diferentes **atributos**
 - tipos e atributos** são usados tanto nas assinaturas quanto nas notificações
- serviço de eventos CORBA e Jini

Arquitetura de Notificação de Eventos Distribuída



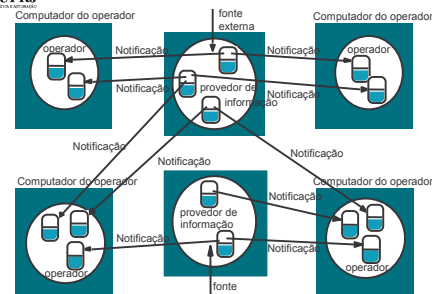
- observadores**
 - permitem desvincular um objeto de interesse de seus assinantes
 - assume o papel do publisher
 - caso 3: observador consulta o objeto de interesse e gera as notificações

Arquitetura



- os observadores podem desempenhar diversos papéis:
 - apenas encaminhamento
 - realiza a função de notificação dos assinantes
 - filtragem
 - redução do número de notificações
 - padrões de eventos
 - relacionamento entre vários eventos
 - caixa de correio
 - armazena as notificações
- a especificação do sistema de evento distribuído Jini permite um subscriber numa JVM assinar e receber notificações de eventos em um objeto de interesse em outra JVM

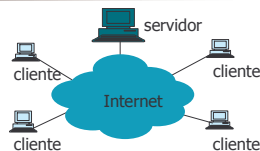
Sistema de corretora de ações



Arquitetura Cliente/Servidor



- modelo de grande sucesso na Internet
 - WWW, FTP, WebServices, etc.
- fonte dos dados: servidor conhecido, de alto poder computacional e confiável
- clientes solicitam dados ao servidor
- não explora o potencial de computação distribuída
- dependente de servidores bem configurados e com informação acessível
 - administração centralizada: problema de escalabilidade
 - ponto único de falha



- recursos nas bordas não utilizados
 - nós (clientes) com capacidade razoável ficam "escondidos"
 - a existência de um ou milhares de computadores é indiferente

Arquitetura P2P



- nós são equivalentes (pares)**
 - software executado em cada nó é equivalente em funcionalidade
 - clientes e servidores: tanto fornecem quanto consomem dados
- sistema descentralizado e distribuído**
 - sem fonte centralizada de dados
 - maior robustez
 - dados podem estar em qualquer nó e qualquer nó pode iniciar uma interação
 - maior democracia e ameaça aos direitos autorais
- maior dinamicidade: nós entram e saem da rede**
- escalabilidade**
 - não há gargalo para crescimento
- robustez**
 - não há ponto de falha único
- flexibilidade**
 - auto-configuração / configuração dinâmica



Definição P2P (1)



O compartilhamento de recursos e serviços computacionais diretamente entre sistemas

A distributed network architecture may be called a **Peer-to-Peer network**, if the **participants share a part of their own hardware** resources (processing power, storage capacity, network link capacity, printers). These shared resources are necessary to provide the **Service and content offered by the network** (e.g. file sharing or shared workspaces for collaboration). They are **accessible by other peers directly, without passing intermediary entities**. The participants of such a network are thus resource (Service and content) **providers** as well as resource (Service and content) **requestors** (Server-concept).

R.Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of P2P Architectures and Applications", Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)

Definição (2)



- classe de aplicações que traz como vantagem a utilização de recursos disponíveis nas **bordas** da Internet
- quais recursos?
 - armazenamento
 - processamento
 - conteúdo
 - presença humana

Objetivos



- compartilhamento/redução de custos
- aumento da escalabilidade/confiabilidade
- agregação de recursos
 - SETI@Home, distributed.net agregam capacidade de processamento.
 - Napster, Gnutella e FreeNet agregam espaço de armazenamento e largura de banda para transferir dados
- aumento de autonomia
 - o serviço não fica dependente de servidores
- privacidade e anonimato
 - esquema que permite esconder a identidade dos nós requisitantes e fornecedores dos dados.
- dinamismo
 - sistema flexível em que os recursos entram e saem continuamente

Tipos de aplicação (1)



- Comunicação e colaboração
 - comunicações e colaboração em tempo real entre pares
 - exemplo: chat e instant messaging (Yahoo, msn, Jabber,...).
- Computação distribuída
 - utilizar tempo de cpu disponível nos pares
 - exemplo: seti@home (<http://setiathome.ssl.berkeley.edu>)
 - The Search for Extraterrestrial Intelligence
 - usuários executam partes da "busca"
- Internet Service Support
 - serviços de internet que se baseiam em infraestruturas p2p
 - exemplo: multicast e aplicações de segurança (DoS e virus)

Tipos de aplicação (2)

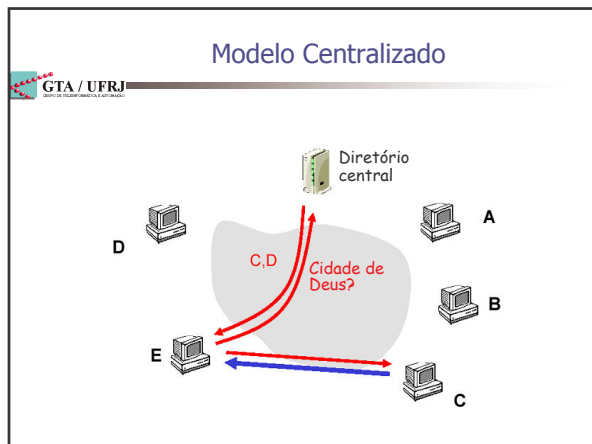


- Sistemas de Bases de Dados
 - BDs distribuídas baseadas em infra-estruturas p2p
 - exemplo: Edutella.
- Distribuição de conteúdos
 - a maior parte dos sistemas enquadra-se neste tipo de aplicação
 - compartilhamento/distribuição de conteúdos digitais (documentos, vídeos, música, etc).
 - aumenta a área de armazenamento e a disponibilidade de informações
 - pode prover o anonimato e apresentar problemas de violação de direitos autorais
 - exemplos: Napster, Gnutella, Freenet, Kazaa, etc.

Modelos de Sistemas P2P: Classificação 1



- modelo centralizado
 - índice global mantido por uma autoridade central
 - contato direto entre clientes e provedores
 - exemplo: Napster
- modelo descentralizado (rede p2p pura)
 - sem índice global (sem coordenação global)
 - exemplos: Gnutella, Freenet
- modelo hierárquico (rede p2p híbrida)
 - introdução dos super-nós (super-nodes ou super-peers)
 - mistura dos modelos centralizado e descentralizado
 - exemplos: KaZaA, Morpheus

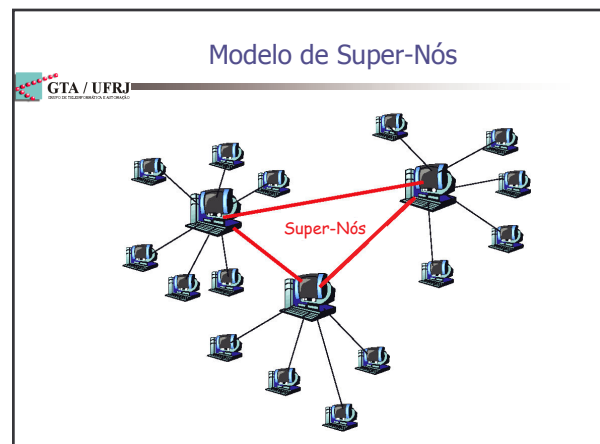


- ### Modelo Centralizado
- os pares se comunicam com um diretório central para publicarem o conteúdo que oferecem para compartilhamento
 - quando solicitado por um par, o índice central identifica o par que melhor corresponde ao pedido
 - por ser o mais barato, o mais disponível ou o mais rápido, em função dos requisitos do usuário
 - transferência de dados é efetuada diretamente entre os pares (requisitante e provedor)
 - desvantagem: escalabilidade e ponto único de falha
 - exemplo: Napster

- ### Modelo Descentralizado (Rede P2P Pura)
- todos os nós são iguais (sem hierarquia): par
 - sem administração central
 - rede auto-mantida e com elevada resiliência
 - a falha de um nó não implica a falha da rede
 - problema: como é que um nó pode se juntar a uma rede?
 - bootstrap*
 - através de uma mensagem de inundação para efetuar a busca de um nó da rede p2p
 - se ligar a um nó pré-definido (p.ex. arquivo de configuração atualizado a cada nova participação na rede p2p)
 - exemplos: Gnutella e Freenet

- ### Rede P2P Híbrida
- definição (1) + uma entidade central para prover parte dos serviços oferecidos pela rede
 - primeiro é contactado um super-nó para obter uma metainformação para facilitar o *bootstrap*
 - identidade de outros pares
 - verificar credenciais de segurança
 - no passo seguinte é estabelecida a comunicação p2p
 - exemplos: Napster, Groove, Aimster, Magi, Softwax e iMesh

- ### Modelo de Super-Nós (Super-Peer)
- alguns nós funcionam como super-nós por possuírem maior capacidade, conectividade ou confiabilidade.
 - um super-nó pode acelerar o processo de adição de novos pares, mantendo uma lista de nós de ligação
 - exemplo: KaZaa



Modelos de Busca P2P: Classificação 2



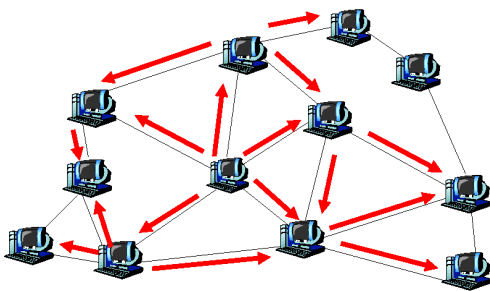
- Centralized Service Location (CSL)
 - Busca centralizada
 - Exemplo: Napster
- Flooding-based Service Location (FSL)
 - Busca baseada em inundação
 - Exemplo: Gnutella
- Distributed Hash Table-based Service Location (DHT)
 - Busca baseada em tabela de hash distribuída
 - Exemplos: CAN, Pastry, Tapestry, Chord

Modelo por Inundação de Pedidos

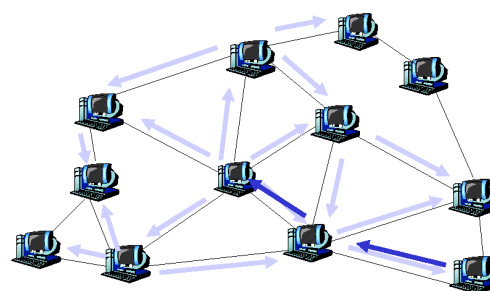


- sistema p2p puro
- não existe publicação dos recursos compartilhados
- o pedido é enviado (flooded) aos pares conectados ao nó que por sua vez enviam aos seus pares
- o processo pára quando o pedido recebe uma resposta ou um certo limite no número de inundações é atingido
- desvantagem: escalabilidade (exige maior largura de banda)
- exemplo: Gnutella
- aplicações: redes pequenas como e.g. rede de empresa
- soluções: super-nós (indexação), caching de pedidos

Modelo por Inundação de Pedidos



Modelo por Inundação de Pedidos



Modelo de Roteamento do Documento



- a cada par é atribuído um ID que conhece um conjunto de outros pares
- um documento, ao ser publicado, recebe um ID de acordo com o seu conteúdo e nome e é encaminhado ao par com o ID mais próximo do seu
 - neste encaminhamento cada par guarda uma cópia local
- os pedidos de documentos são redirecionados ao par com o ID mais próximo ao ID do documento requisitado
 - o processo se repete até o documento ser encontrado, sendo depois transferido ao requisitante
- desvantagem: o ID do documento deve ser conhecido antes de um par requisitá-lo
- exemplo: FreeNet
- aplicações: redes de grande dimensão

Modelos de Sistemas P2P: Classificação 3



- Modelo Centralizado
 - garantia de localização e buscas flexíveis
 - Napster, mensagens (ICQ, etc)
- Modelo Descentralizado e Não Estruturado
 - super-Nós: KaZaA
 - inundação: Gnutella
 - buscas flexíveis e sem garantia de sucesso
- Modelo Descentralizado e Estruturado
 - DHT
 - topologias bem definidas: anel, árvore, grade, hipercubo, etc.
 - buscas exatas e garantia de sucesso em $\log(N)$
 - Chord, etc.

Redes P2P Descentralizadas e Estruturadas



- segunda geração de redes P2P
- auto-organizável
- balanceamento de carga
- tolerante a falhas
- garantias quanto ao número de saltos para responder a uma solicitação
 - principal diferença com relação às redes não-estruturadas
- baseada em DHT

Distributed Hash Tables (DHT)



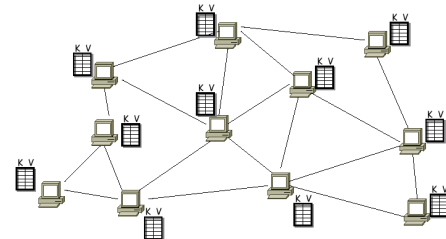
- versão distribuída de uma tabela hash
- armazena pares (chave, valor) \Rightarrow (key, value)
 - chave (key) – nome de um arquivo
 - valor (value) – pode ser o conteúdo do arquivo ou a localização do conteúdo
- objetivo: inserir/procurar/apagar pares (chave, valor)
- cada par (peer) armazena um subconjunto de pares (chave, valor) do sistema
- funcionamento de base:
 - achar o nó responsável por uma determinada chave
 - mapeamento chave \Rightarrow nó
 - rotear eficientemente pedidos de insere/procura/apaga a este nó

DHT Generic Interface

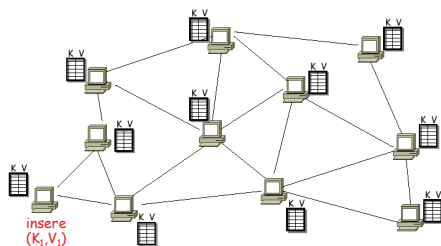


- **Node id:** m-bit identifier (similar to an IP address)
- **Key:** sequence of bytes
- **Value:** sequence of bytes
- **put(key, value)**
 - store (key,value) at the node responsible for the key
- **value = get(key)**
 - retrieve value associated with key from the appropriate node

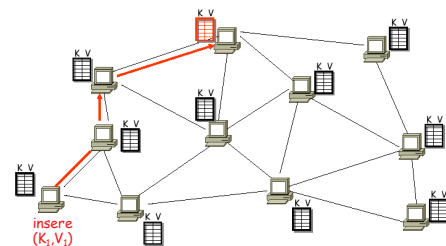
DHT: Idéia Básica

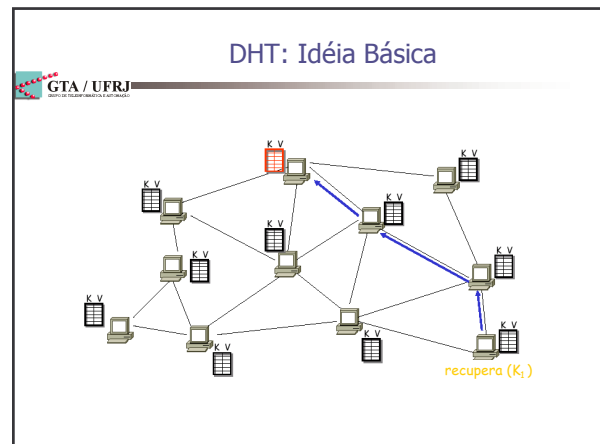
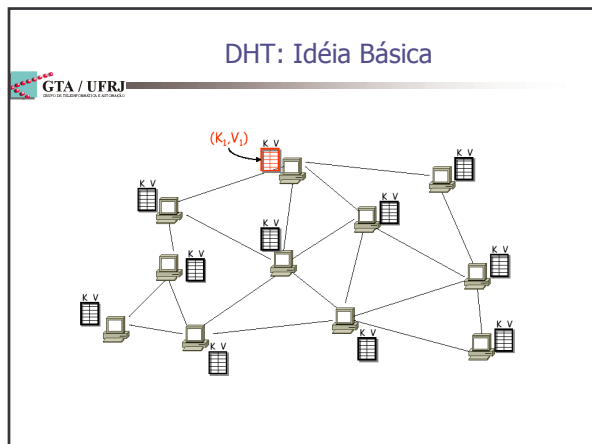


DHT: Idéia Básica



DHT: Idéia Básica





- ### Propriedades Desejáveis da DHT
- chaves mapeadas uniformemente em todos os nós da rede
 - em geral, qualquer função aleatória de hash "boa" é suficiente: padrão SHA-1 (colisão praticamente impossível)
 - cada nó deve manter informação apenas de alguns poucos outros nós
 - o número de vizinhos por nó deve ser escalável
 - mensagens devem ser eficientemente roteadas a um outro nó
 - o diâmetro de busca deve ser escalável ($\log N$)
 - o roteamento deve ser distribuído (sem *hotspots*)
 - a chegada e saída de um nó somente afeta poucos nós
 - precisa re-dividir as chaves pelos nós existentes e reorganizar o conjunto de vizinhos de cada nó
 - mecanismo de "entrada" (bootstrap) na rede, para conectar novos nós na infra-estrutura DHT

- ### Protocolos de Roteamento DHT
- DHT é uma interface genérica
 - existem várias implementações dessa interface:
 - Chord [MIT]
 - Pastry [Microsoft Research UK, Rice University]
 - Tapestry [UC Berkeley]
 - Content Addressable Network (CAN) [UC Berkeley]
 - SkipNet [Microsoft Research US, Univ. of Washington]
 - Kademlia [New York University]
 - Viceroy [Israel, UC Berkeley]
 - P-Grid [EPFL Switzerland]
 - Freenet [Ian Clarke]

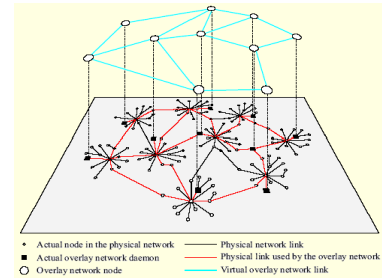
- ### Caching, replicação e migração de conteúdos
- os sistemas p2p baseiam-se na replicação dos conteúdos que disponibilizam de modo a:
 - aumentar a disponibilidade dos conteúdos
 - aumentar o desempenho
 - aumentar a resistência a uma eventual tentativa de censura (tornar conteúdos indisponíveis)

- ### Replicação de conteúdos
- replicação passiva
 - os conteúdos são replicados devido às transferências ocorridas quando um documento é pedido por um par.
 - replicação por caching
 - o documento fica temporariamente guardado nos pares por onde passa até chegar ao par destino.
 - replicação ativa
 - utilizada para melhorar a localidade dos dados (tempo de acesso aos dados) e a disponibilidade (evitar que a falha/inativação de pares torne os documentos indisponíveis)
 - obs.: a replicação de documentos impõe dois requisitos adicionais: consistência e sincronização dos dados.

Outras características

- **segurança**
 - relacionada com a disponibilidade, privacidade, confidencialidade, integridade e autenticidade.
- **anonimato**
 - em relação ao autor do documento; do nó em que está armazenado; a identidade e detalhes do próprio conteúdo; os detalhes da *query* necessária para obter o conteúdo.
- **negação de conhecimento**
 - possibilidade do usuário desconhecer o conteúdo guardado no seu nó (através de codificação sem distribuição de chaves).
 - o usuário não pode ser responsabilizado pelo conteúdo do seu nó uma vez que não tem acesso aos dados.

Redes Sobrepostas (Overlay)



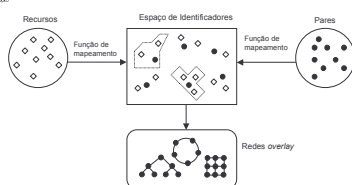
Redes Sobrepostas: Definição

- rede construída sobre uma outra rede pré-existente
- nós de uma rede sobreposta conectados por enlaces, chamados lógicos ou virtuais
 - cada enlace virtual corresponde a caminhos, possivelmente de múltiplos saltos, na rede subjacente
- exemplos e usos
 - redes P2P sobreposta a Internet: busca de informação
 - Internet "discada" sobre a rede pública de telefonia.
 - roteamento (multicast)
 - distribuição de conteúdo: CDN
 - resiliência

Redes Sobrepostas: Construção e Manutenção

- como a rede se inicia?
 - primeiro nó?
- bootstrap
 - como novo nó entra na rede sobreposta?
- manutenção de vizinhança
 - como o nó mantém conhecimento sobre seus vizinhos na rede sobreposta?
- como ocorre fusão (join) de redes?

Redes Sobrepostas P2P



1. Escolha do espaço de identificadores
2. Mapeamento dos recursos e pares para o espaço de identificadores
3. Gerenciamento do espaço de identificadores pelos pares
4. Estrutura lógica da rede
5. Estratégia de roteamento
6. Estratégia de manutenção

Redes Sobrepostas P2P: Resumo

- escolha do espaço de identificadores
- mapeamento dos recursos e pares no espaço de identificadores
 - dependente da aplicação e balanceamento de carga
- gerenciamento do espaço de identificadores pelos pares
 - pode variar com o tempo
 - política do mais próximo
 - redundância (um identificador em vários pares)
- estrutura lógica da rede
 - conexões entre pares
- estratégia de roteamento
 - centralizada, inundação e DHT
- estratégia de manutenção
 - proativas e reativas

Redes Sobrepostas P2P: Chord

| Nós | Chaves |
|-----|-------------|
| 5 | 26-31 e 0-5 |
| 9 | 6-9 |
| 16 | 10-16 |
| 25 | 17-25 |

| Nós | Índice (i) | Identificador (5 + 2 ⁱ) | Nó |
|-----|------------|-------------------------------------|----|
| 5 | 0 | 6 (5 + 1) | 9 |
| | 1 | 7 (5 + 2) | 9 |
| | 2 | 9 (5 + 4) | 9 |
| | 3 | 13 (5 + 8) | 16 |
| 9 | 4 | 21 (5 + 16) | 25 |
| | 0 | 10 (9 + 1) | 16 |
| | 1 | 11 (9 + 2) | 16 |
| | 2 | 13 (9 + 4) | 16 |
| 3 | 3 | 17 (9 + 8) | 25 |
| | 4 | 25 (9 + 16) | 25 |

Histórico do Compartilhamento de Arquivos P2P: Wikipedia

- 1a geração: cliente-servidor
 - lista de arquivos centralizada
 - exemplos:
 - Napster (sistema pago)
 - eDonkey2000 na versão servidor (Overnet e KAD)
 - Limewire
 - web-based sharing
 - em redes sociais: Facebook
- 2a geração: descentralização
 - Gnutella: problemas de escalabilidade
 - solução FastTrack: alguns nós "more equal than others" (super-nós) (indexação)
 - Gnutella (super-nós)
 - DHTs: vários (ou todos) nós indexando hashes
 - vantagem: busca rápida e eficiente
 - desvantagem: não permite busca por palavras-chave (apenas exact-match)
 - exemplos: Gnutella, Kazaa (servidor central para login), eMule (com Kademlia).

Histórico do Compartilhamento de Arquivos P2P: Wikipedia

- 3a geração: indireto e criptografado (anonymous P2P)
 - roteamento através de outros nós e criptografia forte
 - WASTE, JetiAnts, Tor e I2P
 - friend-to-friend
 - exemplos: ANts P2P, RShare, Freenet, I2P, GNUnet e Entropy.
 - Example software includes
- 4a geração: streams over P2P
 - rádio e TV
 - exemplos:
 - geral: Broadcastcatching e Podcast
 - árvore: CoolStreaming e PeerCast
 - Swarming (BT-like): Djingle, Icecast, Joost, MediaBlog, PeerCast, PPLive, PPStream, SopCast, TVUPlayer e Vuze

Bibliografia

- "A survey of Peer-to-Peer Content Distribution Technologies", S. Androutsellis-Theotokis and D. Spinellis, ACM Computing Surveys, Vol. 36, N. 4, December 2004, pp. 335-371.
- "A Definition of Peer-to-Peer Networking for the Classification of P2P Architectures and Applications", R. Schollmeier, Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)