



Sistemas Distribuídos

Núcleo de Educação a Distância

www.unigranrio.com.br

Rua Prof. José de Souza Herdy, 1.160

25 de Agosto – Duque de Caxias - RJ



Reitor

Arody Cordeiro Herdy

Pró-Reitoria de Programas de Pós-Graduação

Nara Pires

Pró-Reitoria Administrativa e Comunitária

Carlos de Oliveira Varella

Pró-Reitoria de Programas de Graduação

Livia Maria Figueiredo Lacerda

Núcleo de Educação a Distância (NEAD)

Márcia Loch

1ª Edição

Produção: Gerência de Desenho Educacional - NEAD

Desenvolvimento do material: Ana Cláudia de Moura
Laurentino


Copyright © 2020, Unigranrio

Nenhuma parte deste material poderá ser reproduzida, transmitida e gravada, por qualquer meio eletrônico, mecânico, por fotocópia e outros, sem a prévia autorização, por escrito, da Unigranrio.


Sumário

Sistemas Distribuídos

Para início de conversa	05
Objetivo	06
1. Arquiteturas Utilizadas em Sistemas Distribuídos	07
1.1 Peer-to-peer	07
1.1.1 Peer-to-peer Pura	08
1.1.2 Peer-to-peer Híbrida	09
1.1.3 Peer-to-peer Centralizada	09
1.2 Cliente-servidor	10
1.2.1 Arquitetura de Duas Camadas	11
1.2.2 Arquitetura de Três Camadas	12
1.2.3 Arquitetura de Quatro Camadas	13
2. Infraestrutura de Sistemas Distribuídos	14
2.1 Rede	14
2.2 <i>Hardware</i>	15
2.3 <i>Software</i>	16
2.3.1 Sistemas Operacionais de Rede	16
2.3.2 Sistemas Operacionais Distribuídos	16



2.4	<i>Middleware</i>	17
3.	Comunicação em Sistemas Distribuídos	17
3.1	Comunicação via Sockets	17
3.1.1	Como é Feita a Comunicação	18
3.2	Comunicação por RPC	20
	Referências	22





Para início de conversa...

Este capítulo será dedicado aos estudos dos sistemas distribuídos, para que sejam desenvolvidos conhecimentos que permitam a você definir, dentre as arquiteturas disponíveis, quais devem ser utilizadas em determinados contextos para atender aos interesses do ambiente em que o sistema será construído.

Em um primeiro momento, o conteúdo dará ênfase às arquiteturas, suas características, seus prós e os contras e como esses aspectos afetam o funcionamento do sistema distribuído. Depois disso, discutiremos a infraestrutura de um sistema distribuído, analisando como ela deve ser projetada e construída. Por fim, falaremos da comunicação nos sistemas distribuídos e como ela integra seus diferentes componentes.

Ao final deste capítulo, você obterá conhecimentos importantes para delinear um sistema distribuído de acordo com as necessidades de cada situação.



Objetivo

Projetar sistemas distribuídos.



1. Arquiteturas Utilizadas em Sistemas Distribuídos

Os sistemas distribuídos são projetados para que seu *software* e *hardware* trabalhem de forma coordenada a fim de atender às demandas a que se destina. Uma das decisões mais importantes na construção de um sistema distribuído é a definição de qual arquitetura utilizar.

Uma arquitetura de sistema de distribuição nada mais é que uma proposta de organização ou estruturação de seus componentes para que se construa um sistema que seja confiável, fácil de controlar e manter, e que atenda ao seu propósito.

Por trás de um sistema distribuído, existe um *software* que se responsabiliza pelo gerenciamento das diferentes partes que o compõem, permitindo que elas se comuniquem e troquem informações para trabalharem, de fato, em conjunto. Isso vai ao encontro da afirmação de Tanenbaum e Steen (2007), que define um sistema distribuído como uma combinação de vários computadores que parecem ser apenas um para o usuário.

As estruturações dos sistemas distribuídos, de forma genérica, podem pertencer a três classes diferentes: arquitetura peer-to-peer, arquitetura cliente-servidor e arquitetura em camadas.

1.1 Peer-to-peer

A arquitetura de sistema distribuído peer-to-peer (PSP) é caracterizada por apresentar todos os seus nós conectados entre si para que possam compartilhar seus recursos. A aplicação desse tipo de organização só pode ocorrer em situações nas quais não haja um servidor dedicado, que também pode ser chamado de centralizado, como nas arquiteturas cliente-servidor. Os sistemas organizados com a arquitetura P2P permitem que qualquer um dos nós execute funções de servidor ou cliente em algum momento.

A comunicação P2P permite que todos os nós conectados conversem uns com os outros, o que faz com que a rede tenha uma estrutura mais livre no que diz respeito às funcionalidades dos nós.

O uso dos sistemas P2P é bastante comum em soluções de compartilhamento de dados, como no BitTorrent e no antigo Emule. Nessas aplicações, os arquivos que serão compartilhados não estão em um único servidor, mas nos computadores de todos os nós que já o baixaram. Quando um novo nó deseja baixar um determinado arquivo, o *software* procura uma lista dos nós mais próximos que o possuem, e a aplicação o conecta com um deles para que possam trocar os dados desejados. Então, fica evidente que o câmbio de informações acontece de forma direta, não havendo um intermediário dessa ação.



Comentário

O mais comum é que na arquitetura P2P não exista um controlador central, o que significa que se um nó da rede cair, os demais continuam aptos a se comunicar. Dessa forma, nem um peer consegue ter uma visão ampla dos demais participantes do sistema distribuído, mas é permitido comunicar-se com qualquer um deles.

É importante ressaltar que com essa organização, a escalabilidade é simples, ou seja, é fácil inserir novos nós e expandir os recursos disponíveis dentro da estrutura. E esse tipo de organização é bastante heterogêneo, permitindo a comunicação entre nós com diferentes configurações, condição propiciada por um *middleware* que coordena esse processo de transmissão de dados entre eles.

Uma das desvantagens da rede P2P pura é que, para descobrir quais peers possuem o arquivo ou recurso desejado, é preciso enviar requisições para todos os peers conectados, a não ser que o *middleware* possua uma solução mais elaborada ou que exista um nó que concentre essas informações.

Diante desse contexto, vamos conhecer as principais variações das redes peer-to-peer.

1.1.1 Peer-to-peer Pura

Na arquitetura P2P pura, há uma rede completamente descentralizada, ou seja, sem a presença de qualquer elemento central que intermedeie o compartilhamento de recursos.

Então, toda solicitação de recursos é encaminhada para todos os nós, a fim de descobrir qual deles pode atendê-la. Essa situação recebe o nome de descoberta por inundação, e é bastante custosa, pois faz com que a rede fique sobrecarregada com comunicações em excesso. Para resolver essa questão, tornou-se comum que as redes P2P sejam adaptadas, acrescentando elementos centralizadores para controle de determinadas situações.

1.1.2 Peer-to-peer Híbrida

As arquiteturas P2P híbridas são modificações da pura, acrescentando alguns elementos centralizadores que são chamados de supernós, aos quais são atribuídas algumas funções importantes e capazes de melhorar o funcionamento da estrutura como um todo.

Dentre as funções que podem ser conferidas aos supernós e que otimizam o funcionamento da rede, estão:

- Controle do acesso à rede, ou seja, o supernó fica responsável por verificar as credenciais e permissões do nó, antes de deixá-lo acessar a rede.
- O supernó pode ser utilizado para guardar um índice dos demais nós e seus recursos, de modo que não seja necessário consultar todos os peers para saber qual deles possui o recurso desejado; após isso, eles se comunicam diretamente.



Importante

Para evitar problemas com a falha do supernó, a rede pode ser programada para repassar as funções antes atribuídas a ele para outro de maneira automática, permitindo, assim, a continuidade do serviço prestado.

1.1.3 Peer-to-peer Centralizada

A arquitetura P2P centralizada também é considerada híbrida, só que nela existem, de fato, servidores centralizados para a gestão de entrada e saída

de nós da estrutura. Além desse controle, ao entrar na rede o nó registra no servidor central os recursos que está disposto a compartilhar, o que compõe um índice que facilita as consultas dos nós.

O servidor central recebe as requisições de consulta de recursos, indica o peer que o possui, define a banda e o processamento. Depois disso, a comunicação ocorre diretamente entre os peers.


1.2 Cliente-servidor

A arquitetura cliente-servidor é um modelo que propõe uma divisão entre os computadores que oferecem serviços (servidores) e os que utilizam serviços (clientes). Segundo Tanenbaum e Wetherall (2011), as redes clientes-servidores se caracterizam por apresentarem computadores conectados, chamados de clientes, que acessam serviços ou informações presentes em um servidor remoto.

Nessa arquitetura, os clientes que normalmente estão em computadores comuns enviam requisições aos servidores, que costumam ser máquinas mais potentes e projetadas para serem capazes de realizar tarefas mais complexas.



Figura 1. Demonstração da arquitetura cliente-servidor. Fonte: Dreamstime.



Os clientes enviam requisições ao servidor, que tem a responsabilidade de processá-las e de devolver os resultados ao cliente. Como os servidores são máquinas mais potentes, é comum que neles estejam concentradas as tarefas mais pesadas, como o banco de dados. Para preservar a comunicabilidade nessa arquitetura, as diferentes máquinas precisam utilizar o mesmo protocolo de comunicação.

A arquitetura de cliente-servidor também é chamada de multicamada, de modo que suas variações são nomeadas de acordo com a quantidade de camadas que possuem. A seguir, vamos conhecer essas possíveis variações e, para exemplificá-las, utilizaremos a situação de um cliente que precisa acessar um servidor de banco de dados.

1.2.1 Arquitetura de Duas Camadas

A arquitetura de duas camadas foi a primeira criada a partir da aplicação da proposta cliente-servidor. Nela, o cliente recebia um *software* que continha todas as regras de negócio para que ele pudesse utilizar os serviços do servidor. Isso significa que todas as regras para manipulação do banco de dados estavam concentradas no *software* do cliente, e não no servidor. Dessa forma, o servidor tinha apenas a função de realizar a consulta.

A organização mencionada apresentava duas camadas:

1. **Camada do cliente:** Contava com o *software* que tinha as funcionalidades para geração de requisições, lógica de negócio e visualização dos resultados das consultas.
2. **Camada do servidor de banco de dados:** Camada com apenas um sistema de gerenciamento de banco de dados (SGBD).

Com o passar do tempo, foi possível perceber que essa organização passou a não suportar as mudanças constantes nas regras de negócio e legislação, ou seja, a arquitetura apresentou grandes dificuldades para sua manutenção.

Para melhorar essa condição, surgiram novos modelos, aprimorando, assim, a técnica de cliente-servidor.

1.2.2 Arquitetura de Três Camadas

A arquitetura de três camadas promove uma mudança considerável em relação à de duas camadas, removendo, do lado do cliente, toda a lógica de negócio. Isso torna o desenvolvimento da lógica de negócio mais complexo, porque deve considerar as diferentes plataformas e situações que podem ser enfrentadas, visto que estará em uma aplicação única colocada no servidor de aplicação.

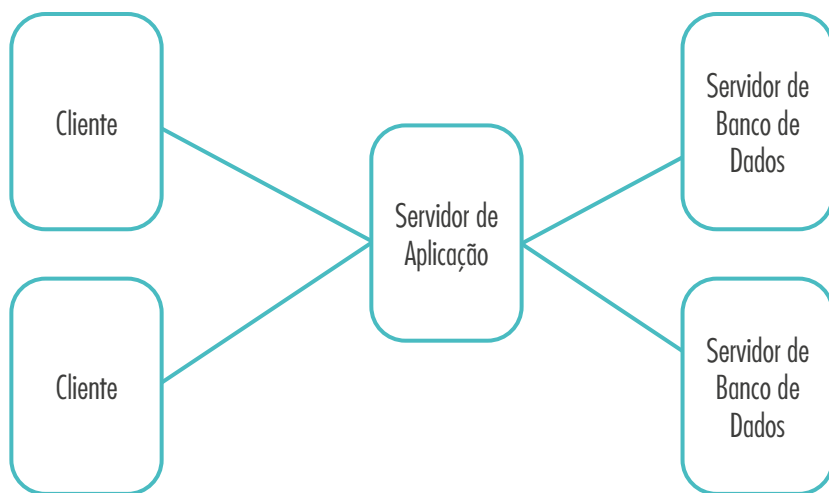


Figura 2: Estrutura de arquitetura com três camadas. Fonte: Elaborada pela autora.

O estabelecimento do servidor de aplicação dá origem à terceira camada da arquitetura. Sempre que o cliente desejar enviar uma consulta ao banco de dados, ela passará pelo servidor de aplicação que conterá a lógica de negócio e intermediará todos os acessos ao servidor de banco de dados. Essa arquitetura melhora a escalabilidade e facilita o trabalho de manutenção, se comparada com a arquitetura de duas camadas.

Diante disso, pode-se descrever essa arquitetura com as seguintes camadas:

1. **Camada do cliente:** Contém recursos para requisitar ações do servidor e exibir os resultados retornados de seu processamento.

2. **Camada de aplicação:** No servidor de aplicação estão todas as lógicas de negócio que deverão ser atendidas para realização das operações do cliente no servidor de banco de dados.
3. **Camada de dados:** Consiste no servidor de dados, onde são realizadas as operações sobre os dados armazenados.

1.2.3 Arquitetura de Quatro Camadas

Se comparada à de três, pode-se afirmar que a arquitetura de quatro camadas retira do cliente todas as funções de apresentação e as coloca em servidor chamado web. Nesse caso, o cliente deverá acessar uma aplicação de apresentação que ficará em um servidor web por meio do seu navegador. Isso permitirá que ele dispare ações e visualize resultados de qualquer dispositivo que tenha um navegador.

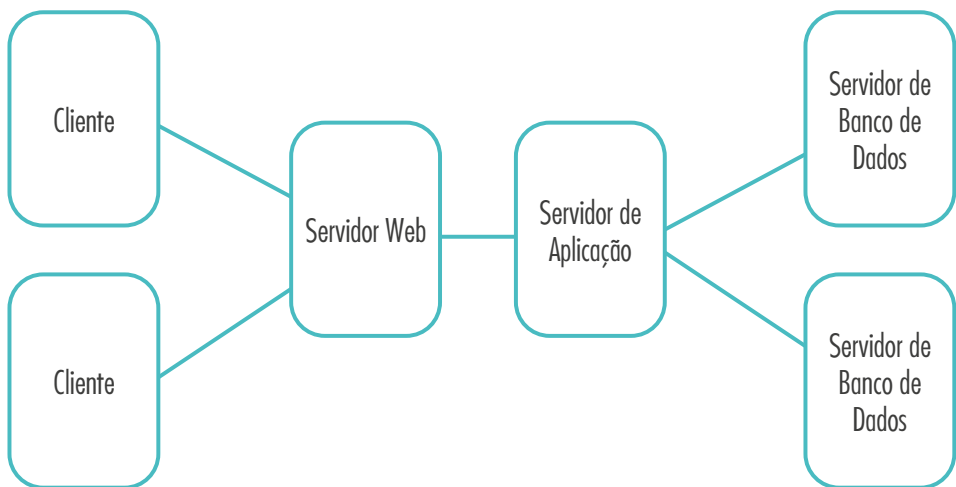


Figura 3: Estrutura de arquitetura com quatro camadas. Fonte: Elaborada pela autora.

Portanto, para enviar alguma requisição ao servidor de banco de dados, o cliente utilizará seu navegador para acessar o servidor web e a aplicação de apresentação por meio da qual fará sua requisição, que passará pelo servidor de aplicação para validação das regras de negócio e seguirá para o servidor de banco de dados no qual serão executadas. Depois disso, o resultado é devolvido para o servidor web, que apresentará os resultados em sua aplicação.

Durante a projeção das redes, é preciso analisar o controle de eventos, verificar se a sincronização seria feita por meio de mensagens e por meio de quais protocolos essa comunicação será efetivada: RMI, RPC, UDP/TCP, Sockets. A comunicação é fundamental para garantir a consistência das ações na rede.

O controle de acesso dos usuários à rede deve ser tratado, considerando a utilização de recursos como: Active Directory, ADAM, LDAP.

2.2 Hardware

O compartilhamento de recursos de *hardware* pode ser feito de duas formas diferentes, e é muito importante que elas estejam presentes, pois a ideia de sistema distribuído se baseia totalmente na possibilidade de se fazer mais com menos, por intermédio do compartilhamento de recursos.

Múltiplos processadores compartilhando memória: É possível promover o compartilhamento de recursos por meio da utilização de múltiplos processadores, que trabalham em conjunto e compartilham uma única memória principal conectada a eles mediante um único barramento.

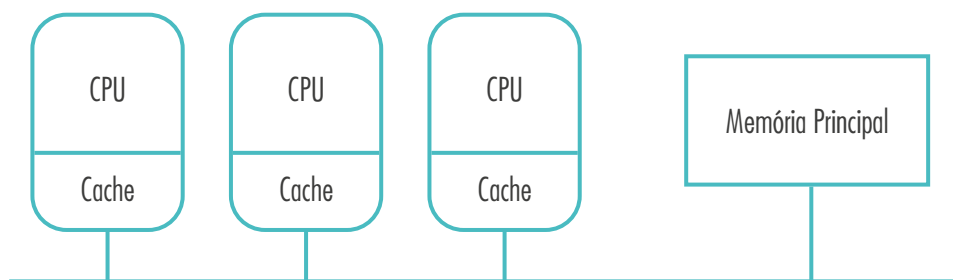


Figura 4: Múltiplos processadores compartilhando uma memória principal. Fonte: Elaborada pela autora.

Múltiplos computadores trabalhando de modo compartilhado: Nesse caso, diferentes computadores, cada um com sua memória, são conectados por meio de uma aplicação e trabalham em conjunto, como se fossem um só.

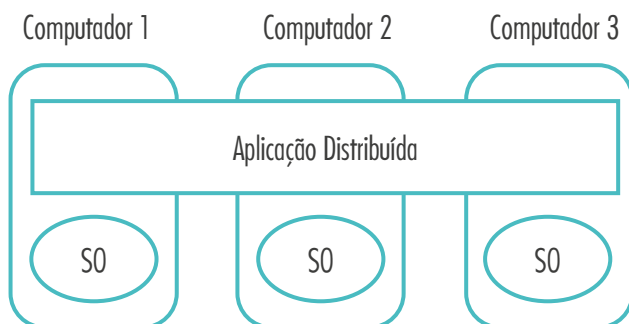


Figura 5: Múltiplos computadores independentes trabalhando em conjunto. Fonte: Elaborada pela autora.

2.3 Software


Os *softwares* também são fundamentais na estruturação de um sistema distribuído, escolhendo sistemas operacionais de rede, sistemas operacionais distribuídos e *middleware*. A seguir, vamos entender o papel de cada um.

2.3.1 Sistemas Operacionais de Rede

Os sistemas operacionais de rede (SOR) complementam os sistemas operacionais (SO) das máquinas, oferecendo recursos que permitem que elas se comuniquem, compondo uma rede de forma transparente ao usuário. O SOR e SO trabalham juntos; o primeiro permite a comunicação entre diferentes máquinas, e o segundo viabiliza o controle dos recursos que cada máquina possui. Dentre os exemplos de sistemas operacionais para rede, podemos citar: Unix, Windows NT, 2000 e XP.

2.3.2 Sistemas Operacionais Distribuídos

Um sistema operacional distribuído (SOD) é um sistema que roda sobre um conjunto de máquinas que são independentes, mas, a partir de sua instalação, trabalham em conjunto, como se formassem uma única e grande máquina.



Esse é um recurso fundamental na composição de um sistema distribuído, oferecendo diversos benefícios: comunicação, controle de concorrência, compartilhamento de recursos e controle de falhas. Os principais sistemas operacionais distribuídos conhecidos são: Chorus, Amoeba e Mach.

2.4 *Middleware*

O *middleware* é um *software* que estabelece uma camada intermediária entre o sistema operacional e os *softwares* de aplicação, e que complementa o sistema operacional, oferecendo alguns serviços não concedidos por ele.

No caso dos sistemas distribuídos, o *middleware* atua viabilizando a comunicação entre aplicações com protocolos diferentes e máquinas de plataformas distintas.

Para composição de *middleware* em sistemas distribuídos, são mais conhecidos os *softwares*: Java RMI, Corba e DCOM.

3. Comunicação em Sistemas Distribuídos

O funcionamento de um sistema distribuído é completamente dependente das comunicações que são realizadas entre os nós, garantindo a coordenação e o pleno desempenho de todo o sistema.

Agora, falaremos sobre dois dos principais tipos de comunicação que podem ser utilizados entre os nós de um sistema distribuído.

3.1 Comunicação via Sockets

A comunicação via **socket** conecta diferentes processos, sendo classificada como *Inter Process Communication* (IPC).



Glossário

Um **socket** é uma abstração de endereços utilizados para comunicação entre processos, os quais são compostos pela junção do endereço da máquina e o identificador da porta que o processo está utilizando. Essa composição permite saber à qual máquina e à qual aplicação as mensagens trocadas devem ser entregues.

Os sockets podem ser criados em dois modelos diferentes:

- **Socket_stream TCP:** socket que trabalha de forma orientada à conexão, respeitando o protocolo TCP, e produz uma conexão mais confiável.
- **Socket_dgram UDP:** trabalha conforme o protocolo UDP e, por isso, não cria uma conexão tão confiável como no modelo anterior.

Os protocolos de comunicação definem as operações que devem ser executadas para a realização de determinadas funcionalidades. Essas operações são repassadas ao sistema operacional que, por sua vez, possuem autonomia para definir qual API irá processar a operação.

Dessa forma, o programador utiliza procedimentos de sockets em seu *software* e pode transportar a sua aplicação para diferentes sistemas operacionais, apenas importando a biblioteca adequada para manipulação dos sockets.

3.1.1 Como é Feita a Comunicação

Para o *software*, os sockets são vistos como recursos para realização de operações de entrada e saída de dados. Assim, ele conta com operações como: *open*, *read*, *write*, *close*. Isso já deixa claro que o uso de um socket sempre incluirá a sua criação e eliminação.

A API de socket requer a especificação de alguns parâmetros para que a comunicação seja feita de forma correta, como: protocolo de transporte, endereço do protocolo e verificação se a aplicação é cliente ou servidor.



O primeiro passo deve ser a criação do socket, de modo que, em seguida, será possível utilizar os métodos para configurá-lo. A seguir, vamos conhecer alguns dos principais métodos para manipulação de sockets.

Para criar um socket, basta seguir a seguinte o seguinte parâmetro

```
primeiroSocket = socket(protofamily, type, protocol);
```

Nesse método de criação de socket, estão disponíveis os seguintes dados:

- **protofamily:** Define a família de protocolos que será utilizada junto ao novo socket.
- **type:** Define se o socket irá se comunicar de forma orientada ou não orientada à conexão, respectivamente, `sock_stream` e `sock_dgram`.
- **protocol:** Define o protocolo para transporte que será utilizado.

Outro método importante é o `bind`, que faz associação do socket com uma determinada porta da máquina. Esse método tem a seguinte estrutura:

```
bind(sockfd, localaddr, addrlen);
```

Os dados integrantes desse procedimento são:


- **sockfd:** Esse é o descritor do socket que é obtido por `socket`.
- **localaddr:** Atribui endereço local para identificação do socket.
- **addrlen:** Argumento para especificar um inteiro com tamanho do endereço, para adaptar o socket a diferentes protocolos de comunicação que possuem tamanhos de endereço distintos.

O estabelecimento da conexão com o servidor também é feito pela utilização de um método específico:

```
connect(sockfd, saddress, saddresslen);
```

Nesse caso, os seguintes dados fazem parte do processo:

- **sockfd:** Envia o descritor que identifica o socket.



Nesse capítulo, podemos conhecer mais sobre a estruturação de um sistema distribuído e quais são as principais precauções a serem consideradas durante esse processo. Além disso, foram apresentadas ferramentas e recursos importantes para que a estrutura seja funcional e cumpra o seu papel.

Nesse contexto, falamos sobre as arquiteturas que são utilizadas na composição de um sistema distribuído e suas características, as quais devem ser consideradas antes do início do desenvolvimento, visto que esse processo com certeza será afetado pela arquitetura escolhida.

Entendendo que a comunicação é a chave para que os diferentes recursos trabalhem em conjunto, conhecemos duas das principais formas de comunicação para esse tipo de aplicação: via socket e RCP.



Referências

TANENBAUM, A. S.; WETHERALL, D. **Redes de computadores**. 5 ed. Rio de Janeiro: Pearson, 2011.

TANENBAUM, A. S.; STEEN, M. V. **Sistemas distribuídos: princípios e paradigmas**. Pearson, 2007.

