Camada de Aplicação

Modelo *Client-Server*

O modelo *client-server* é constituído por entidades chamadas clientes que se conectam a entidades chamadas servidores para obter acesso a recursos e serviços. Os servidores são responsáveis por armazenar os dados e as funcionalidades do sistema, bem como responder às solicitações feitas pelos clientes. Os clientes são responsáveis por fornecer a interface aos utilizadores e por enviar solicitações aos servidores.

As vantagens deste modelo são:

- Escalabilidade: é fácil de adicionar mais recursos e capacidade de processamento aos servidores, sem afetar os clientes.
- Segurança: os dados e as funcionalidades são armazenados e geridos pelo servidor, o que facilita proteger o sistema contra acessos não autorizados.
- Centralização: os dados e as funcionalidades estão centralizados no servidor, o que torna mais fácil a manutenção e gestão do sistema.
- Compartilhamento de recursos: os recursos podem ser compartilhados entre os clientes, podendo melhor a eficiência do sistema.
- Flexibilidade: os clientes podem ser facilmente atualizados ou substituídos, sem afetar os servidores.
- Acessibilidade: os clientes podem ter acesso aos recursos e serviços fornecidos pelos servidores em qualquer lugar, desde que tenham acesso à rede.

Algumas desvantagens são:

- Se os servidores falharem, os clientes deixam de ter acesso aos recursos e serviços.
- Se os servidores ficarem sobrecarregados, o desempenho dos clientes será afetado.
- Os administradores dos servidores têm controlo exclusivo sobre os recursos, podendo tomar decisões não benéficas para os clientes.

Casos de aplicação comuns do modelo cliente-servidor incluem:

- Sistemas de gerenciamento de banco de dados.
- Serviços de e-mail.
- Sistemas de arquivos remotos.
- Sistemas de gerenciamento de conteúdo.

Modelo *Peer-to-Peer*

No modelo *peer-to-peer*, as entidades principais são conhecidas como *peers*, que podem atuar tanto como cliente, quanto como servidor. Por outras palavras, os *peers* podem

fornecer serviços e recursos a uma determinada entidade, assim como solicitar recursos e serviços a uma outra entidade.

Algumas das suas vantagens são:

- Descentralização: Não há necessidade um ponto único de falha, o que aumenta a confiabilidade no sistema.
- Escalabilidade: Quanto mais *peers* forem adicionados, maior será a capacidade de armazenamento e de processamento.
- Eficiência: Os recursos podem ser diretamente compartilhados entre diferentes *peers*, sem terem de passar por um servidor intermediário.
- Privacidade: Os clientes podem compartilhados recursos entre si sem terem de revelar a sua identidade ou informações pessoais, garantindo a sua privacidade.

Algumas das suas desvantagens são:

- Pode ser mais complicado garantir que todos os *peers* estão atualizados e sincronizados.
- A qualidade dos conteúdos partilhados pode ser inconsistente já que os *peers* podem não manter os seus arquivos de conteúdo atualizados ou desativar a sua conexão.
- Pode ser difícil garantir que os recursos compartilhados são seguros e confiáveis, proporcionando problemas de segurança, como a propagação de *malware*, acessos não autorizados a arquivos pessoais e outras ameaças. É importante que as aplicações *peer-to-peer* implementem medidas de segurança adequadas para proteger os usuários e seus dados.

Casos de aplicação comuns do modelo *peer-to-peer* são:

- Compartilhamento de ficheiros/arquivos, como o *BitTorrent*.
- Comunicação instantânea, como o *Skype*.

Tempo de distribuição mínimo usando o modelo *client-server* é calculado a partir de:

$$D_{C-S} \geq \max\{{}^{N*F}/_{u_s}, {}^{F}/_{d_{min}}\}$$

Sendo N o número de clientes, F o tamanho do ficheiro, u_s a capacidade de *upload* do servidor e d_{min} a capacidade de *download* mínima dos clientes.

Tempo de distribuição mínimo usando o modelo *peer-to-peer* é calculado a partir de:

$$D_{P2P} \geq \max\{{}^{N*F}/_{u_s}, {}^{F}/_{d_{min}}, {}^{N*F}/_{u_s + \sum u_i}\}$$

Sendo u_i a capacidade de *upload* de cada um dos clientes.

Tempo de distribuição de ficheiros – *Client-Server vs. Peer-to-Peer*

No modelo client-server, o tempo de distribuição de um ficheiro por todos os clientes é diretamente proporcional ao número de clientes, ou seja, quanto maior for o número de clientes na rede, maior será o tempo de distribuição do ficheiro por todos. Por outro lado, no modelo peer-to-peer, o tempo de distribuição já não é tão diretamente proporcional ao número de clientes, pois a velocidade de propagação também depende da capacidade de *upload* dos próprios clientes. Deste modo, quanto maior for a capacidade de *upload* de cada cliente, menor será o tempo de distribuição do ficheiro.

Podemos, portanto, concluir que o modelo *peer-to-peer* é mais escalável e flexível que o modelo cliente-server, porque ele é adaptável às variações no número de clientes e na capacidade de *upload* de cada um.

Aplicadas usadas na *Internet*

Web browsing

- Débito (*throughput*) necessário: baixo a moderado, geralmente não requer grandes quantidades de dados a serem transferidos simultaneamente.
- Atraso e/ou Jitter: não é sensível, pouco impacto na experiência do usuário se houver atrasos ou variações no tempo de resposta.
- Perda de dados: pouco sensível, a perda de alguns dados não afeta significativamente a experiência do usuário.

Exemplo: O *Google Chrome* é um exemplo de aplicação de navegação *web*.

Multimedia streaming

- Débito (*throughput*) necessário: moderado a alto, requer fluxos de dados contínuos e uma taxa de bits adequada para reproduzir vídeos e áudios com boa qualidade.
- Atraso e/ou Jitter: sensível, atrasos ou variações no tempo de resposta podem causar problemas de sincronização áudio-vídeo e interrupções na reprodução.
- Perda de dados: moderadamente sensível, perder dados pode causar problemas na qualidade da reprodução, mas geralmente não impede a reprodução completa.

Exemplo: O *Netflix* é um exemplo de aplicação de *streaming* de média.

IP Telephony (VoIP)

- Débito (*throughput*) necessário: baixo a moderado, requer apenas pequenas quantidades de dados para transmitir a voz, mas requer baixa latência e jitter para garantir uma conversação fluida.

- Atraso e/ou *Jitter*: altamente sensível, atrasos ou variações no tempo de resposta podem causar problemas de eco e interrupções na conversação.
- Perda de dados: moderadamente sensível, a perda de alguns dados pode causar problemas na qualidade da reprodução, mas geralmente não impede a reprodução completa.

Exemplo: O *Skype* é um exemplo de aplicação de *IP Telephony (VoIP).*

File transfer/sharing

- Débito (*throughput*) necessário: moderado a alto, requer fluxos de dados contínuos e uma taxa de bits adequada para transferir arquivos de forma rápida e eficiente.
- Atraso e/ou *Jitter*: não é muito sensível, mas pode afetar o tempo total de transferência.
- Perda de dados: sensível, a perda de dados pode impedir a transferência completa de um arquivo e requerer que ele seja retransmitido.

Exemplo: O *BitTorrent* é um exemplo de aplicação de transferência ou compartilhamento de ficheiros.

Interactive Games

- Débito (*throughput*) necessário: alto, é necessário um bom débito para transmitir jogos em tempo real.
- Atraso e/ou *Jitter*: baixo, é necessário tempo real para jogos interativos.
- Perda de dados: intolerável, a perda de pacotes afeta a jogabilidade.

Exemplo: *Fortnite*, *Call of Duty*, *League of Legends*.

Video Conferencing

- Débito (*throughput*) necessário: médio a alto, é necessário um bom débito para transmitir vídeo e áudio de alta qualidade.
- Atraso e/ou *Jitter*: baixo, é necessário tempo real para a comunicação ser natural.
- Perda de dados: intolerável, a perda de pacotes afeta a qualidade da chamada.

Exemplo: *Zoom*, *Skype*, *Google Meet*.