Questões Cap. 1

1.1 Cite cinco tipos de recurso de hardware e cinco tipos de recursos de dados ou de software que possam ser compartilhados com sucesso. Dê exemplos práticos de seu compartilhamento em sistemas distribuídos.

Hardware: discos, impressora, memória ram, processador, switch.

Software: arquivos, banco de dados, protocolos de comunicação, funções do sistema operacional, páginas web.

Exemplos:

- Compartilhamento de impressora entre vários computadores em um escritório;
- Memória principal de uma máquina que é compartilhada por todos os processadores através de uma barramento que os interliga;
- Compartilhamento de documentos no google docs.

1.2 Como os relógios de dois computadores ligados por uma rede local podem ser sincronizados sem referência a uma fonte de hora externa? Quais fatores limitam a precisão do procedimento que você descreveu? Como os relógios de um grande número de computadores conectados pela Internet poderiam ser sincronizados? Discuta a precisão desse procedimento.

Para sincronizar os relógios de dois computadores em uma rede local sem referência a uma fonte de hora externa, podemos utilizar protocolos de sincronização como o **Algoritmo de Berkeley** ou o **Cristian's Algorithm**

Fatores que limitam a precisão:

- **Variação de latência**: em redes locais, o tempo de viagem da mensagem pode oscilar, dificultando a estimativa exata do deslocamento do horário.
- **Desvios dos relógios**: cada relógio tem um desvio próprio (drift), o que implica que, sem sincronização contínua, os relógios vão se desincronizar.
- **Atraso de processamento**: o tempo que leva para processar a mensagem de sincronização no cliente e no servidor também contribui para erros.

Sincronização de Vários Computadores na Internet

Para sincronizar um grande número de computadores pela Internet, usa-se normalmente o **Protocolo de Tempo de Rede (NTP)** que é amplamente utilizado na Internet para sincronização de horários.

Precisão da sincronização na Internet

- **Latência de rede**: em uma rede global, a latência pode ser significativa e variar bastante, o que afeta a precisão da sincronização.
- **Número de saltos de rede**: à medida que aumenta o número de saltos entre cliente e servidor, a precisão da sincronização se reduz devido ao aumento de latência.

1.3 Considere as estratégias de implementação de MMOG (massively multiplayer online games) discutidas na Seção 1.2.2. Em particular, quais vantagens você vê em adotar a estratégia de servidor único para representar o estado do jogo para vários jogadores? Quais problemas você consegue identificar e como eles poderiam ser resolvidos?

Vantagens:

Consistência Do Estado do jogo:
Um servidor único centralizado facilita manter um estado do jogo consistente, uma vez que todos os jogadores interagem com a mesma fonte de dados.

Simplificação da lógica do jogo:
A lógica de sincronização é simplificada, pois o servidor é a única fonte da verdade para o estado do jogo. Todas as interações, regras e eventos ocorrem diretamente no servidor, o que torna o desenvolvimento e a manutenção da lógica de jogo mais gerenciáveis.

Problemas:

Escalabilidade e Limitações de Desempenho:
Um único servidor pode se tornar um gargalo, especialmente à medida que a base de jogadores cresce, causando latência, sobrecarga e degradação no desempenho.
Uma solução seria o uso de escalabilidade horizontal, replicando o servidor ou usando uma arquitetura de cluster.

Latência e Problemas de Conectividade:
A comunicação de um servidor central com jogadores globalmente distribuídos pode introduzir latência significativa, afetando a experiência dos jogadores. Como solução, pode-

se implementar servidores distribuídos em diferentes regiões geográficas (servidores de borda), onde cada servidor mantém uma sincronização parcial com o servidor principal.

1.4 Um usuário chega a uma estação de trem que nunca havia visitado, portando um PDA capaz de interligação em rede sem fio. Sugira como o usuário poderia receber informações sobre serviços locais e comodidades dessa estação, sem digitar o nome ou os atributos da estação. Quais desafios técnicos devem ser superados?

O PDA do usuário pode detectar pontos de acesso Wi-Fi na estação, que transmitiriam automaticamente informações locais, como serviços e comodidades. Esses dispositivos enviariam a localização e dados relevantes diretamente ao PDA, ativando conteúdo contextual assim que o usuário se aproximasse.

Desafios técnicos:

- **Compatibilidade de Protocolos**: Garantir que o PDA reconheça e interprete os sinais dos beacons ou Wi-Fi.
- **Privacidade e Segurança**: Proteger os dados pessoais do usuário e impedir acessos não autorizados.

1.5 Compare e contraste a computação em nuvem com a computação cliente-servidor mais tradicional. O que há de novo em relação à computação em nuvem como conceito?

A nuvem oferece escalabilidade automática, permitindo o aumento ou a redução de recursos conforme necessário. Na computação cliente-servidor tradicional, a escalabilidade é limitada pela infraestrutura física do servidor. A computação em nuvem fornece acesso sob demanda a recursos e dados de qualquer lugar e a nuvem funciona com base em pagamento por uso, o que reduz custos iniciais. Na arquitetura cliente-servidor tradicional, os custos são mais elevados devido à necessidade de manter e gerenciar servidores locais. O que é inovador na nuvem é a **abstração da infraestrutura** para o usuário, que pode alocar, configurar e usar recursos com facilidade, sem gerenciar hardware ou servidores diretamente.

1.6 Use a World Wide Web como exemplo para ilustrar o conceito de compartilhamento de recursos, cliente e servidor. Quais são as vantagens e desvantagens das tecnologias básicas HTML, URLs e HTTP para navegação

em informações? Alguma dessas tecnologias é conveniente como base para a computação cliente-servidor em geral?

HTML: Define a estrutura e o conteúdo das páginas. É simples e amplamente suportado, mas limitado para criar interfaces interativas complexas.

URLs: Identificam de forma única os recursos na web, facilitando o acesso. No entanto, URLs podem expor detalhes de estrutura de servidor e são pouco flexíveis para dados dinâmicos.

HTTP: Permite a comunicação entre cliente e servidor, mas é stateless (sem estado), o que dificulta o gerenciamento de sessões e interação contínua.

As tecnologias são leves, fáceis de usar e ideais para navegação básica e acesso rápido a informações. **Desvantagens**: São limitadas para aplicativos interativos, pois não retêm estado entre requisições, exigindo soluções adicionais para aplicações mais complexas. Essas tecnologias são convenientes para sistemas de **informação estática e simples**. Contudo, para computação cliente-servidor mais robusta e interativa, é preciso complementá-las com tecnologias adicionais, como APIs REST, para melhor gerenciamento de estado e interatividade.

1.7 Um programa servidor escrito em uma linguagem (por exemplo, C++) fornece a implementação de um objeto BLOB destinado a ser acessado por clientes que podem estar escritos em outra linguagem (por exemplo, Java). Os computadores cliente e servidor podem ter hardware diferente, mas todos eles estão ligados em uma rede. Descreva os problemas devidos a cada um dos cinco aspectos da heterogeneidade que precisam ser resolvidos para que seja possível um objeto cliente invocar um método no objeto servidor.

A heterogeneidade se aplica aos seguintes aspectos:

- **redes:** É necessário um protocolo de comunicação comum (como HTTP ou gRPC) que garanta que os dados trafeguem corretamente entre redes distintas.
- **hardware de computador**: Devem ser utilizados formatos padronizados para os dados transmitidos (como JSON ou Protobuf) que sejam independentes de hardware.
- **sistemas operacionais**: bibliotecas de abstração que funcionem em múltiplos sistemas operacionais devem ser usadas para permitir a compatibilidade.
- **linguagens de programação**: Como os clientes e o servidor podem ser escritos em diferentes linguagens, é necessário um middleware ou um formato de serialização de

dados comum (por exemplo, usando gRPC, ou uma API RESTful).
- **implementações de diferentes desenvolvedores:** Cada desenvolvedor pode adotar convenções, padrões e bibliotecas diferentes, o que pode afetar a consistência e a interoperabilidade. Para mitigar isso, o uso de contratos de API bem definidos e documentados é essencial, assim como uma interface padronizada que todos sigam.

1.8 Um sistema distribuído aberto permite que novos serviços de compartilhamento de recursos (como o objeto BLOB do Exercício 1.7) sejam adicionados e acessados por diversos programas clientes. Discuta, no contexto desse exemplo, até que ponto as necessidades de abertura do sistema diferem das necessidades da heterogeneidade.

Em um sistema distribuído aberto, a **abertura** permite que novos serviços sejam facilmente integrados e acessados por diferentes clientes, independente de quem desenvolveu esses serviços. Para isso, é fundamental definir interfaces e protocolos de comunicação padrão, o que garante flexibilidade e escalabilidade para agregar novos componentes.

A **heterogeneidade**, por outro lado, refere-se à capacidade do sistema de operar em diferentes ambientes de rede, hardware, sistemas operacionais e linguagens de programação.

Assim, a abertura promove a adição de serviços, enquanto a heterogeneidade assegura que esses serviços sejam acessíveis em ambientes variados.

1.9 Suponha que as operações do objeto BLOB sejam separadas em duas categorias – operações públicas que estão disponíveis para todos os usuários e operações protegidas, que estão disponíveis somente para certos usuários nomeados. Indique todos os problemas envolvidos para se garantir que somente os usuários nomeados possam usar uma operação protegida. Supondo que o acesso a uma operação protegida forneça informações que não devem ser reveladas para todos os usuários, quais outros problemas surgem?

Para garantir que apenas usuários nomeados possam acessar operações protegidas de um objeto BLOB, vários problemas precisam ser considerados:

- **Autenticação**: Verificação da identidade do usuário por métodos como senhas ou tokens.

- **Autorização**: Sistema que valida se o usuário tem permissão para acessar a operação protegida.
- **Segurança dos Dados**: Criptografia de informações sensíveis em trânsito e em repouso.

Além disso, se a operação protegida fornece informações sensíveis, surgem desafios como:

- **Confidencialidade e Integridade**: Garantia de que os dados sejam mantidos secretos e não alterados.
- **Requisitos de Conformidade**: Cumprimento de regulamentos que exigem proteção de dados sensíveis.

1.10 O serviço INFO gerencia um conjunto potencialmente muito grande de recursos, cada um dos quais podendo ser acessado por usuários de toda a Internet por intermédio de uma chave (um nome de string). Discuta uma estratégia para o projeto dos nomes dos recursos que cause a mínima perda de desempenho à medida que o número de recursos no serviço aumenta. Sugira como o serviço INFO pode ser implementado de modo a evitar gargalos de desempenho quando o número de usuários se torna muito grande.

Para minimizar a perda de desempenho à medida que o número de recursos aumenta, os nomes dos recursos devem ser projetados usando um esquema hierárquico, semelhante a um sistema de arquivos, onde os nomes contêm informações estruturais que facilitam a localização (ex.: `tipo/identificador`).

Para evitar gargalos de desempenho, o serviço INFO pode ser implementado usando uma arquitetura de microserviços com balanceamento de carga. O uso de caches em memória (como Redis) também pode melhorar a velocidade de acesso aos recursos mais solicitados, reduzindo a carga nos servidores de back-end.

1.11

Os três principais componentes que podem falhar ao chamar um método em um objeto servidor são:

1. **Cliente**: O cliente pode falhar por perda de conexão ou erro de software. Exemplo: um aplicativo cliente travando.
    - **Tolerância**: Implementação de reintentos automáticos e gerenciamento de estado local.

2. **Rede**: Falhas de rede podem causar perda de pacotes ou latência. Exemplo: um roteador falhando temporariamente.
    - **Tolerância**: Uso de protocolos de comunicação resilientes, como TCP, e retransmissões.
3. **Servidor**: O servidor pode falhar devido a erros no processamento. Exemplo: uma exceção não tratada no código do servidor.
    - **Tolerância**: Implementação de redundância e balanceamento de carga, além de um sistema de monitoramento que reinicia o servidor em caso de falha.

1.12

Argumentos contra execução concorrente:

- **Interferência**: A concorrência pode levar a condições de corrida, onde duas operações simultâneas alteram o mesmo dado, resultando em inconsistências.
- **Gerenciamento de Estado**: Pode ser complexo gerenciar o estado de um objeto compartilhado entre múltiplos clientes.

Argumentos a favor:

- **Desempenho**: A execução concorrente pode aumentar a eficiência e reduzir o tempo de resposta, especialmente para operações independentes.

Interferência: Se dois clientes tentarem modificar o mesmo BLOB simultaneamente, um cliente pode sobrescrever as alterações do outro, levando a dados corrompidos.

Como evitar interferência: Implementar mecanismos de **bloqueio** ou **controle de versão** para garantir que as operações sejam executadas de forma segura e ordenada.

1.13

Recursos podem ser transferidos entre servidores para balanceamento de carga, recuperação de falhas ou atualizações de sistema. Isso assegura que os serviços permaneçam disponíveis e escaláveis.

Fazer multicast de todos os pedidos para um grupo de servidores não é satisfatório, pois isso pode causar sobrecarga desnecessária, latência e inconsistências nas respostas. Uma abordagem mais eficiente seria implementar um serviço de descoberta para direcionar as requisições ao servidor apropriado.

1.14

As iniciais URL denotam "Uniform Resource Locator". Exemplos de diferentes tipos de recursos que podem ser nomeados por URLs incluem:

1. **Página Web**: `https://www.example.com/index.html`
2. **Imagem**: `https://www.example.com/images/picture.jpg`
3. **Arquivo de Vídeo**: `https://www.example.com/videos/movie.mp4`

1.15

Exemplo de URL HTTP: `https://www.example.com:8080/path/to/resource?query=value#fragment`

Componentes principais de um URL HTTP:

- **Protocolo**: `https` (indica o uso de HTTP seguro).
- **Host**: `www.example.com` (nome do servidor).
- **Porta**: `8080` (opcional, especifica a porta do servidor).
- **Caminho**: `/path/to/resource` (localização do recurso no servidor).
- **Consulta**: `?query=value` (parâmetros adicionais).
- **Fragmento**: `#fragment` (identificador de parte da página).

Limites são denotados por caracteres especiais, como `://`, `:`, `?`, e `#`.

Um URL HTTP tem **transparência de localização** em parte, pois os usuários não precisam conhecer a localização física dos recursos. No entanto, a especificação do servidor e do caminho é essencial para acessar os dados desejados.