**Avaliação 1 – Sistemas Distribuídos - Ciência da Computação e Sistemas de Informação - UFN - 2025**

**Nome:\_\_Eduardo C. Ceretta\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Parte 1 – Questões Teóricas

1. **Definição conceitual:** Explique o que é um **sistema distribuído** e cite **três exemplos reais** de sistemas distribuídos utilizados no dia a dia.

**R:** São sistemas geograficamente distribuídos (fracamente acoplados), que se conectam por uma rede, eles cooperam entre si para realizar tarefas. Cada máquina possuí hardwares diferentes e diferentes SO.   
Exemplos: Sistemas bancários, Internet das coisas (Comunicação entre dispositivos e servidores centrais), Serviços de streaming (Netflix, Prime Video...).

1. **Objetivos principais:** Quais são os principais objetivos de um sistema distribuído? Explique como eles se relacionam com **transparência** (de acesso, localização, replicação, falha).

**R: Objetivos principais:** Compartilhamento de recursos, Abertura, Escalabilidade, Tolerância a falhas

**Transparência de Acesso:** Independentemente de um recurso ser local ou remoto, o usuário ou a aplicação o acessa da mesma maneira. Fortemente utilizada junto com o compartilhamento de recursos e abertura (Padronizando a forma de acesso).

**Transparência de localização:** Os usuários acessam os recursos por seus nomes, sem precisar saber o endereço IP ou a máquina onde estão hospedados. Fortemente ligada a Escalabilidade (Permite que recursos sejam mudados de máquina) e Compartilhamento de recursos (Facilitando a busca)

**Transparência de Replicação:** Oculta o fato de que um recurso pode existir em múltiplas cópias (réplicas) em diferentes locais da rede. Fortemente ligada a Tolerância a falhas (Se uma máquina que hospeda uma réplica falhar, o sistema pode simplesmente redirecionar as requisições para outra réplica disponível) Escalabilidade (As requisições podem ser distribuídas entre as várias réplicas, balanceando a carga, ou para a réplica mais próxima).  
  
**Transparência de Tolarerância a Falhas:** A transparência de falha é o que torna um sistema verdadeiramente robusto, mascarando erros e garantindo que o sistema como um todo permaneça operacional e consistente.

1. **Comunicação em sistemas distribuídos:** Diferencie **comunicação síncrona** e **comunicação assíncrona** em sistemas distribuídos. Cite vantagens e desvantagens de cada uma.

R:

Comunicação Sincrona: O processo que envia a mensagem fica bloqueado, até que o remetente responda, assim ele pode continuar a sua execução.

Vantagens: Simplicidade de programação, Tratamento de erros significativos e maior Previsibilidade

Desvantagens: Baixo desempenho, Forte acoplamento, Menor escalabilidade

Comunicação Assincrona: O processo que envia a mensagem não fica bloqueado. Ele envia a mensagem e continua sua execução imediatamente, sem esperar por uma resposta.

Vantagens: Alto desempenho, Desacoplamento, Maior escalabilidade e Responsividade

Desvantagens: Maior complexidade de programação, Tratamento de erros complexos, Dificuldade de Debug

1. **Arquiteturas:** Compare as arquiteturas **cliente-servidor** e **ponto-a-ponto (P2P)** em sistemas distribuídos. Dê um exemplo de aplicação que utiliza cada modelo.

R: Cliente-Servidor: Servidor centralizado, Distinção clara de papeis (Cliente e servidor), Controle de dados feita pelo servidor, Ponto único de falha (compromete o funcionamento), Mais seguro.

Ex: Sistema Web Minha UFN

Ponto-a-Ponto: Descentralizado, Cada par é autônomo podendo ser tanto cliente como servidor, Não um ponto único de falha (não compromete o funcionamento), Menos segura (precisa validar cada par)

Ex; BitTorrent

1. **Falhas:** Quais são os principais tipos de falhas em sistemas distribuídos (falhas de comunicação, falhas de processo, falhas de hardware)? Dê exemplos práticos.

R:

Falhas de comunicação: Perda de mensagem, Corrupção de mensagem, Delay, Duplicação de mensagem

Ex: O download de um arquivo é feito, mas não é possível abri-lo pois o arquivo corrompeu (perda de pacotes) enquanto fazia o download

Falhas de processo: falha de parada, falha de omissão, falha bizantina

Ex: Crash, o servidor de um app de banco de daos para de funcionar devido a um bug (estouro de memória)

Falhas de hardware: Queima de componente, Falha de armazenamento, Falha de energia, Falha de rede

Ex: Um raio atinge um grande data center, causando uma queda de energia.

1. **Relógios:** Explique a diferença entre **relógios físicos** e **relógios lógicos** em sistemas distribuídos. Por que os relógios lógicos são importantes?  
   R: Relógios físicos: Cada hardware possuí um relógio, se preocupa com o tempo real, para sincronizar é utilizado o Network Time Protocol (que sincroniza os relógios via rede), ainda podem ocorrer atrasos devido a latência da rede.

Relógios lógicos: Se preocupa com a ocorrência dos eventos (Qual aconteceu antes?), cada método tem um contador, por exemplo o método que manda a msg incrementa seu contador e o método que recebe a msg incrementa seu contador de modo a ficar maior que o do primeiro.

São importantes porque: Garantem a consistência das operações, consistência dos dados, algoritmos de exlusão mútua, detecção de eventos globais (Snapshots).

1. **Relógio de Lamport:** Descreva como funciona o **algoritmo de Lamport** para relógios lógicos. O que significa a relação de precedência de eventos →?

**R: Algoritmo de Lambert:** Seu objetivo não é medir o tempo real, mas sim atribuir um número (um *timestamp*) a cada evento, de forma que a ordem desses números seja consistente com a causalidade.

A expressão "a -> b" é lida como "a aconteceu antes de b". Ela é verdadeira se uma das seguintes condições for atendida:

**Mesmo Processo:** Se os eventos a e b ocorrem no mesmo processo, e a ocorre antes de b na sequência de execução desse processo

**Troca de Mensagens:** Se a é o evento de envio de uma mensagem por um processo e b é o evento de recebimento dessa mesma mensagem por outro processo, então a -> b.

**Transitividade:** Se a -> b e b -> c, então a -> c.

1. **Threads em sistemas distribuídos:** Explique a diferença entre **threads com memória compartilhada** e **threads com comunicação por mensagens**. Quais vantagens e desvantagens existem em cada abordagem?

R:

Thread com memória compartilhada: Múltiplas threads rodam dentro do mesmo processo e, portanto, compartilham o mesmo espaço de endereçamento de memória. Não há cópia de dados, todas as threads operam sobre a mesma instância da informação.

Vantagens: Alta performance e eficiência, Facilidade de Compartilhamento, Modelo de programação natural

Desvantagens: Complexidade de Sincronização, Erros difíceis de depurar, Não escalável para múltiplas máquinas

Thread sem memória compartilhada: Cada thread possui seu **próprio** espaço de memória privadoe isolado. Uma thread não pode acessar diretamente a memória de outra. Há cópia de dados.

Vantagens: Segurança e isolamento, Ideal para sistemas distribuídos, menos propenso a erros de concorrência.

Desvantagens: Sobrecarga de comunicação, Complexidadde no compartilhamento de dados, Possiveis problemas de comunicação.

1. **Escalabilidade:** O que significa dizer que um sistema distribuído é **escalável**? Cite três técnicas que podem ser utilizadas para aumentar a escalabilidade.

**R:** Escalabilidade: Significa que tem a capacidade de manter ou melhorar seu desempenho e sua eficiência quando há um aumento significativo na carga de trabalho.

Replicação e Balancemanto de carga: Carga dividida entre diversos servidores, criação de réplicas do servidor ou banco de dados.

Particionamento ou fragmentação: Em vez de armazenar todos os dados em um único grande banco de dados, os dados são divididos em conjuntos menores e independentes, chamados de partições. Cada partição é armazenada em um servidor de banco de dados separado. Redução de carga de leitura e escrita

Caching: Técnica de armazenar em uma camada de memória de acesso rápido (o *cache*) os resultados de operações caras ou os dados que são acessados com muita frequência. Redução de latência, alivio de carga nos sistemas back-end.

1. **Estudo de caso:** A Netflix utiliza sistemas distribuídos em grande escala. Explique como conceitos como **replicação, balanceamento de carga e tolerância a falhas** são aplicados em plataformas de streaming.

**R:**

Replicação de Dados: Todas as informações que não são os vídeos em si, como perfis de usuário, histórico de visualização, informações de faturamento, e o catálogo de filmes são replicadas em múltiplos servidores de banco de dados.

Replicação de conteúdo de vídeo: A Netflix não transmite um filme de um único data center nos EUA para o mundo todo. Em vez disso, ela replica os arquivos de vídeo em centenas de pequenos servidores.

Balanceamento de Carga: Distribui as requisições de entrada entre um conjunto de servidores para garantir que nenhum servidor individual fique sobrecarregado.

**Requisições da Aplicação:** Ao abrir o aplicativo da Netflix, todas as suas ações (login, busca, navegação no catálogo) são enviadas para a infraestrutura da Netflix na AWS. Um Balanceador de Carga recebe essas requisições e as distribui de forma inteligente entre os milhares de microsserviços que estão rodando.

**Requisições de Streaming:** Ao clicar em "play", um sistema de balanceamento mais sofisticado entra em ação. Ele decide qual dos servidores OCA (da CDN) deve lhe entregar o vídeo.

Tolerância a Falhas:

Redundância e Failover Automático**:** A replicação é a base da tolerância a falhas. Se um servidor, ou um banco de dados ficar offline, o balanceador de carga detecta a falha e redireciona todo o tráfego para as réplicas saudáveis em outras zonas.

**Arquitetura de Microsserviços:** Isso significa que a plataforma não é um grande programa monolítico. Em vez disso, é composta por centenas de pequenos serviços independentes (um para login, um para recomendações, um para streaming, etc.).

## Parte 2 – Questões Práticas

1. **Thread básica:** Implemente em Python um programa que crie **duas threads**: uma imprime números pares de 0 a 20, e a outra imprime números ímpares de 1 a 19. As threads devem rodar em paralelo.

Feita

1. **Threads com atraso:** Implemente em Python um programa que crie **três threads**. Cada thread deve imprimir seu nome ("Thread 1", "Thread 2", "Thread 3") com intervalos de tempo diferentes (ex.: 1s, 2s, 3s).

Feito

1. **Uso de memória compartilhada (contador):** Implemente em Python um programa com **duas threads** que incrementam uma mesma variável global (contador). Mostre como ocorre a **condição de corrida** sem o uso de locks.  
   Feita
2. **Uso de Lock para sincronização:** Modifique o programa da questão anterior utilizando threading.Lock para evitar a condição de corrida e garantir que o contador final seja consistente.

Feito

1. **Produtor-Consumidor com Queue:** Implemente um programa em Python que utilize threads para simular o problema **Produtor-Consumidor**. Use queue.Queue para comunicação entre threads.  
   Feito
2. **Simulação de relógio lógico:** Implemente em Python um programa com **duas threads** que representam dois processos. Cada thread gera eventos locais e envia mensagens à outra. Utilize um contador lógico simples (Lamport) para marcar os eventos.

**Feito**

1. **Thread com parâmetro:** Implemente em Python um programa que crie uma thread responsável por calcular a soma de uma lista de números inteiros passada como parâmetro. A thread deve retornar o resultado.

**Feito**

1. **Barrier (sincronização de threads):** Implemente em Python um programa que crie **quatro threads**. Todas devem executar uma parte de uma tarefa, mas só podem prosseguir para a próxima etapa quando **todas as threads** terminarem a primeira parte (use threading.Barrier).

**Feito**

1. **Thread e deadlock:** Implemente um exemplo em Python onde duas threads tentam acessar dois recursos compartilhados, gerando um **deadlock**. Em seguida, explique como evitar o problema.

**Feito / Comentario explicando no final do codigo**