

Prática Offline 1

1. Balanceamento de carga.

- O balanceamento de carga é uma técnica essencial em sistemas distribuídos para distribuir uniformemente o trabalho entre vários recursos, como servidores, para garantir eficiência, alta disponibilidade e evitar sobrecarga.
- A seguir, alguns dos principais algoritmos de balanceamento de carga são brevemente apresentados:
- **Round Robin (RR)**
 - No algoritmo **Round Robin**, as requisições são distribuídas de forma sequencial e cíclica entre os servidores disponíveis.
 - Por exemplo, se houver três servidores, o primeiro pedido vai para o servidor 1, o segundo para o servidor 2, o terceiro para o servidor 3, e assim por diante, reiniciando o ciclo.
 - É simples e fácil de implementar, mas não considera a capacidade ou carga atual dos servidores, o que pode levar a uma distribuição desigual de carga.
 - Considere um sistema com três servidores (S1, S2, e S3), que recebe uma série de requisições de clientes.
 - Exemplo: Suponha que cinco requisições são recebidas (R1, R2, R3, R4, R5).
 - R1 → S1
 - R2 → S2
 - R3 → S3
 - R4 → S1
 - R5 → S2
 - Resumo:
 - O algoritmo **Round Robin** distribui as requisições de maneira sequencial e cíclica.
 - Após atingir o último servidor, ele volta para o primeiro, independentemente da carga ou capacidade dos servidores.
- **Least Connections (LC)**
 - O algoritmo **Least Connections** distribui as requisições para o servidor com o menor número de conexões ativas no momento.
 - Isso ajuda a garantir que nenhum servidor fique sobrecarregado, especialmente útil quando as requisições têm tempos de processamento variáveis.
 - Esse método é mais dinâmico do que o **Round Robin**, pois leva em consideração a carga atual dos servidores.
 - Exemplo: Suponha que, inicialmente, todos os servidores tenham zero conexões. Recebemos cinco requisições (R1, R2, R3, R4, R5).
 - R1 → S1 (Conexões: S1=1, S2=0, S3=0)
 - R2 → S2 (Conexões: S1=1, S2=1, S3=0)
 - R3 → S3 (Conexões: S1=1, S2=1, S3=1)
 - R4 → S1 (Conexões: S1=2, S2=1, S3=1)
 - R5 → S2 (Conexões: S1=2, S2=2, S3=1)
 - Resumo:
 - Cada nova requisição é enviada para o servidor com o menor número de conexões ativas.
 - Isso distribui a carga de maneira mais uniforme, especialmente útil quando as requisições têm tempos de processamento variáveis.
- **Least Response Time (LRT)**

- Este algoritmo atribui a requisição ao servidor que possui o menor tempo de resposta ou latência, combinando a métrica de número de conexões ativas com a velocidade de resposta.
- É útil em sistemas onde o tempo de resposta é uma métrica crítica, garantindo que as requisições sejam atendidas o mais rápido possível.
- Exemplo: Supomos que os tempos de resposta atuais sejam: S1=10ms, S2=20ms, S3=15ms. Recebemos três requisições (R1, R2, R3).
 - R1 → S1 (melhor tempo de resposta de 10ms)
 - R2 → S1 (tempo de resposta agora aumentado, mas ainda o menor)
 - R3 → S3 (tempo de resposta de S1 aumentou, agora S3 é o menor)
- Resumo:
 - As requisições são enviadas para o servidor com o menor tempo de resposta atual, levando em conta tanto o número de conexões quanto a velocidade de resposta.

- **Hashing (Consistent Hashing)**

- No algoritmo de hashing, um valor hash é calculado para cada requisição com base em atributos como o IP do cliente ou a URL solicitada.
- O hash resultante é usado para decidir qual servidor receberá a requisição.
- O **Consistent Hashing** é uma variante que lida bem com a adição ou remoção de servidores, minimizando a redistribuição de cargas.
- Isso é particularmente útil em sistemas escaláveis e distribuídos como caches e sistemas de armazenamento de dados distribuídos.
- Exemplo: Usamos o endereço IP dos clientes para determinar o servidor. Suponhamos que tenhamos três clientes com IPs que, quando passarem pela função de hash, resultam nos seguintes mapeamentos:
 - IP do cliente A → hashA → S2
 - IP do cliente B → hashB → S3
 - IP do cliente C → hashC → S1
- Resumo:
 - O hash do IP do cliente é mapeado para um servidor específico. No caso de adição ou remoção de servidores, o **Consistent Hashing** minimiza a redistribuição das requisições.

- **Weighted Round Robin (WRR)**

- É uma extensão do **Round Robin**, onde a cada servidor é atribuído um peso que determina a proporção de requisições que ele deve receber.
- Servidores mais poderosos ou menos carregados podem receber mais requisições do que outros. Isso permite uma distribuição mais equilibrada considerando as capacidades dos servidores.
- Exemplo: Suponhamos que temos os pesos S1=1, S2=2, S3=3. Recebemos seis requisições (R1, R2, R3, R4, R5, R6).
 - R1 → S1 (peso 1)
 - R2 → S2 (peso 2)
 - R3 → S2 (peso 2)
 - R4 → S3 (peso 3)
 - R5 → S3 (peso 3)
 - R6 → S3 (peso 3)
- Resumo:
 - O **Weighted Round Robin** atribui requisições aos servidores com base em seus pesos. S3, com o peso mais alto, recebe mais requisições.

- **Weighted Least Connections (WLC)**

- Semelhante ao **Least Connections**, mas aqui cada servidor recebe um peso, e a decisão de balanceamento é feita com base no número de conexões ativas ponderado pelo peso do servidor.
- Servidores mais poderosos recebem mais requisições proporcionalmente ao seu peso.
- Exemplo 1: Pesos são atribuídos aos servidores com base em sua capacidade (S1=1, S2=2, S3=3). Suponha que recebemos quatro requisições (R1, R2, R3, R4).

- R1 → S1 (Conexões ponderadas: S1=1, S2=0, S3=0)
- R2 → S2 (Conexões ponderadas: S1=1, S2=2, S3=0)
- R3 → S3 (Conexões ponderadas: S1=1, S2=2, S3=3)
- R4 → S1 (Conexões ponderadas: S1=2, S2=2, S3=3)
- Exemplo 2: Considere 10 requisições (primeiro testa o C para depois estar o S*).
- R1 → S1 (Conexões ponderadas: S1=1 (C = 1), S2=0 (C = 0), S3=0 (C = 0))
- R2 → S2 (Conexões ponderadas: S1=1 (C = 1), S2=2 (C = 1), S3=0 (C = 0))
- R3 → S3 (Conexões ponderadas: S1=1 (C = 1), S2=2 (C = 1), S3=3 (C = 1))
- R4 → S1 (Conexões ponderadas: S1=2 (C = 2), S2=2 (C = 1), S3=3 (C = 1))
- R5 → S2 (Conexões ponderadas: S1=2 (C = 2), S2=4 (C = 2), S3=3 (C = 1))
- R6 → S3 (Conexões ponderadas: S1=2 (C = 2), S2=4 (C = 2), S3=6 (C = 2))
- R7 → S1 (Conexões ponderadas: S1=3 (C = 3), S2=4 (C = 2), S3=6 (C = 2))
- R8 → S2 (Conexões ponderadas: S1=3 (C = 3), S2=6 (C = 3), S3=6 (C = 2))
- R9 → S3 (Conexões ponderadas: S1=3 (C = 3), S2=6 (C = 3), S3=9 (C = 3))
- R10 → S1 (Conexões ponderadas: S1=4 (C = 4), S2=6 (C = 3), S3=9 (C = 3))
- Resumo do Exemplo 2:
 - R1 é atribuída a S1 porque todos os servidores têm zero conexões, e S1 tem o menor peso, resultando em uma conexão ponderada de 1.
 - R2 vai para S2, que tem o próximo menor número de conexões ponderadas (2).
 - R3 vai para S3, que tem o menor número de conexões ponderadas após R1 e R2.
 - R4 volta para S1, que, após R1, tem uma conexão ponderada de 1, enquanto S2 e S3 têm valores mais altos.
 - R5 vai para S2, que agora tem uma conexão ponderada menor que S3.
 - R6 vai para S3, que tem a menor carga ponderada após R5.
 - R7 é atribuída a S1, que ainda tem uma carga ponderada mais baixa que os outros dois servidores.
 - R8 vai para S2, aumentando suas conexões ponderadas para 6.
 - R9 é direcionada para S3, elevando sua carga ponderada para 9.
 - R10 é enviada para S1, que agora tem uma carga ponderada de 4, ainda menor que os outros servidores.
- Resumo:
 - As requisições são atribuídas com base no número de conexões ponderadas pelos pesos dos servidores, proporcionando uma distribuição de carga mais equilibrada.

● Balanceamento de Carga Dinâmico

- Algoritmos de balanceamento de carga dinâmicos ajustam suas estratégias com base em informações em tempo real sobre o estado do sistema, como carga do servidor, latência de rede, e outras métricas de desempenho.
- Eles podem usar uma combinação de técnicas para otimizar a distribuição de carga de maneira adaptativa.
- Exemplo: Suponha que usamos uma combinação de métricas, como o uso da CPU, memória e tempo de resposta. Recebemos duas requisições (R1, R2).
 - R1 → S3 (melhor combinação de métricas no momento)
 - R2 → S1 (após R1, as métricas de S3 mudaram e S1 se tornou a melhor escolha)
- Resumo:
 - O balanceador de carga avalia continuamente o estado dos servidores e ajusta a distribuição de requisições com base em métricas em tempo real.
 - Isso pode incluir o uso de recursos, latência, e outros fatores de desempenho.

● Random

- O algoritmo de balanceamento de carga **Random** distribui as requisições de forma aleatória entre os servidores disponíveis.
- Embora seja simples, pode não ser eficiente em termos de balanceamento de carga, pois não considera a capacidade ou estado dos servidores.
- Exemplo: Suponha que tenhamos quatro requisições (R1, R2, R3, R4) e três servidores.
 - R1 → S2
 - R2 → S1
 - R3 → S3

- R4 → S1
- Resumo:
 - As requisições são atribuídas aleatoriamente aos servidores disponíveis. Não há uma lógica definida para a escolha do servidor, o que pode levar a uma distribuição não uniforme da carga.

2. Simulação: sistema distribuído para coleta e gerenciamento de dados climáticos.

- Considere um sistema onde quatro drones sobrevoam determinados ambientes (divididos em leste, oeste, norte e sul) e por meio de sensores, coletam dados de importantes elementos climáticos como pressão atmosférica, radiação solar, temperatura e umidade.
 - Drone norte coleta os dados no formato: `pressao-radiacao-temperatura-umidade`.
 - Drone sul coleta os dados no formato: `(pressao;radiacao;temperatura;umidade)`.
 - Drone leste coleta os dados no formato: `{pressao,radiacao,temperatura,umidade}`.
 - Drone oeste coleta os dados no formato: `pressao#radiacao#temperatura#umidade`.
- Os dados coletados são enviados para um centro de dados, que possui, para este fim, dois servidores dedicados a receber e armazenar os dados na base de dados centralizada.
 - Os dados devem ser armazenados no formato: `[temperatura//umidade//pressao//radiacao]`.
- Os drones enviam dados ao centro de dados para estes realizarem operações de escrita na base. As operações de leitura são feitas por usuários de dentro ou de fora do centro de dados.
- **Cenário típico:**
 - Os drones enviam os dados coletados para servidores do centro de dados a cada intervalo de 2 a 5 segundos.
 - A comunicação entre o processo que simula o centro de dados e os servidores pode ser feita usando um *ip multicast* ou por *unicasts* (implementando o balanceamento de carga de conexões).
 - Um servidor faz a escrita dos dados na base (pode ser na memória principal, em um processo dedicado a isso).
 - Após isso, a requisição de atualização do servidor pode ser consumida por usuários.
 - **Modo 1: *ip multicast*.**
 - O servidor envia os dados recebidos dos drones para um grupo de usuários ou para um *ip multicast*.
 - Usuários que necessitem obter esses valores devem receber mensagens a cada nova atualização via grupo ou *ip multicast*.
 - **Modo 2: *unicasts*.**
 - Conexões ponto a ponto entre usuários (cliente) e servidores (implementando o balanceamento de carga).
- **Funcionamento:**
 - A partida da simulação pode ser uma opção, que ao ser digitada no console dos drones, inicia a coleta (números obtidos aleatoriamente, mas dentro de uma faixa considerada aceitável para cada elemento climático).
 - Permita que usuários (três, por exemplo) possam fazer leituras dos dados no sistema.
 - Três processos devem se vincular a um grupo ou *ip multicast*.
 - Estabeleça um tempo de execução da simulação. Por exemplo, três minutos. Ao final, teremos um log com muitos dados coletados.
- **Utilização na implementação:**
 - O serviço de executores para o gerenciamento de threads.
 - Lambda, interfaces funcionais e *streams* com justificativo de uso.
 - Desenvolver cenários de simulação para dois algoritmos de balanceamento de carga de requisições para os servidores.
 - O balanceamento de carga é obrigatório, seja na comunicação **drones < — > centro de dados** ou na comunicação **usuários < — > servidor** do centro de dados.

3. Observações.

- O prazo para a entrega e apresentação dos projetos expira em 05/06/2025 às 23:59h, via SIGAA. Portanto, certifiquem-se do arquivo que vão enviar.
 - A apresentação vai ser feita no laboratório com as simulações executadas em duas máquinas, no mínimo.
 - Para os que enviarem por e-mail, depois do prazo, o projeto valerá 20% a menos.

- Data limite para entrega com atraso: 06/06/2025.
- Avaliação: o projeto vale 100% da nota da 1ª unidade.
 - Perguntas individuais podem ser feitas sobre o código e a apresentação.
- O projeto pode ser individual ou em dupla.
 - Caso seja feito em dupla, enviar os nomes dos componentes até o dia 31/05/2025.
- Os trabalhos devem utilizar as tecnologias vistas, até o momento, na disciplina para desenvolver o projeto.
- Sabe-se que a estrutura de projetos dessa natureza pode ser muito comum. No entanto, a lógica de funcionamento, o armazenamento e a visualização das informações da loja podem ser bem particulares. Cuidado com códigos iguais. A penalidade é a nota ZERO.