Sistemas Distribuídos - Projeto 2

Daniel José Gomes de Lima
RA 11201921053

Sistemas Distribuídos DA2 2023.2

O link do vídeo demonstração do relatório abaixo pode ser acessado através do Youtube através do link https://youtu.be/MPTGEIMQk-A

O sistema do **cliente** é parte integrante de um sistema distribuído, onde existem vários servidores que armazenam os dados. O objetivo do cliente é fornecer uma interface para que os usuários possam interagir com o sistema, realizando operações de armazenamento (PUT) e recuperação (GET) de dados.

O sistema do **cliente** oferece as seguintes **funcionalidades**:

- **Armazenamento de Dados (PUT):** Permite que os usuários armazenem pares chave-valor no sistema distribuído. Os dados são armazenados localmente no cliente e replicados nos servidores adequados.

- **Recuperação de Dados (GET):** Permite que os usuários recuperem o valor associado a uma chave específica do sistema distribuído. O cliente consulta os servidores para obter o valor atualizado.

O sistema do **cliente** possui os seguintes **métodos**:

- **__init__():** Inicializa o sistema do cliente, definindo os servidores disponíveis e criando um mapa vazio para armazenar os pares chave-valor.

- **put(key, value):** Armazena um par chave-valor no mapa local do cliente. O par chave-valor é inserido no mapa local e posteriormente replicado nos servidores adequados.

- **get(key):** Retorna o valor associado a uma chave específica no mapa local do cliente. O cliente consulta o mapa local e, se a chave não estiver presente, faz uma solicitação aos servidores para obter o valor atualizado.

- **sendRequest(message):** Envia uma solicitação para um servidor especificado e retorna a resposta recebida. A solicitação é enviada em formato JSON e a resposta é decodificada e retornada ao chamador.

- **requestInit():** Solicita ao usuário que digite "INIT" para inicializar o cliente. Essa etapa é necessária antes de executar qualquer operação.

- **doPut(targetServer, key, value):** Executa a operação PUT em um servidor específico, enviando a chave e o valor correspondentes. A operação PUT é realizada por meio de uma conexão TCP, onde o cliente envia a solicitação para o servidor alvo.

- **doGet(targetServer, key):** Executa a operação GET em um servidor específico, buscando o valor associado a uma chave. O cliente envia uma solicitação para o servidor alvo e recebe a resposta contendo o valor correspondente.

- **run():** Responsável por executar o sistema do cliente. Ele apresenta um menu para o usuário escolher entre realizar uma operação PUT ou GET e inicia uma nova thread para executar a operação selecionada.

- **callMethods():** Chama continuamente o método run(), permitindo que o cliente execute operações repetidamente. Esse método mantém o sistema do cliente em execução até que seja interrompido.

O **fluxo de execução** do sistema do **cliente** é o seguinte:

O cliente é iniciado e o usuário é solicitado a digitar "INIT" para inicializar o sistema. Após a inicialização, o cliente apresenta um menu com as opções PUT e GET.

Quando o cliente seleciona a opção PUT e insere a chave e o valor correspondente, ele inicia uma nova thread para executar a operação PUT. O cliente não precisa se preocupar em saber se o servidor em que está enviando a requisição é o líder ou não. Essa decisão é tratada pelos próprios servidores do sistema. Ao receber a resposta do servidor após a operação PUT, o cliente atualiza sua lista local com o valor recebido. Essa atualização é realizada independentemente de o servidor ser o líder ou não. O cliente confia no sistema distribuído para realizar a replicação e garantir a consistência dos dados.

Essa abordagem permite que o cliente interaja com o sistema de forma simplificada, sem a necessidade de entender os detalhes internos do sistema distribuído. O cliente confia que as operações são executadas corretamente pelos servidores, independentemente de qual servidor seja o líder responsável por coordenar as operações de replicação.

A operação GET é tratada de forma a garantir que o cliente receba informações corretas e atualizadas. Quando o cliente solicita a operação GET para uma chave específica, o servidor verifica a presença da chave em seu mapa local. Se a chave estiver presente e o timestamp associado for maior ou igual ao solicitado pelo cliente, o servidor envia a mensagem "GET_OK" de volta ao cliente, incluindo o valor associado à chave e o timestamp correspondente. Isso assegura que o cliente receba o valor mais atualizado. Por outro lado, se a chave não estiver presente no mapa local do servidor, o servidor envia a mensagem "NULL" para indicar que a chave não existe. Essa resposta informa ao cliente que a chave solicitada não foi encontrada no sistema. Além disso, se o timestamp associado à chave no servidor for menor do que o solicitado pelo cliente, o servidor envia a mensagem "TRY_OTHER_SERVER_OR_LATER". Essa mensagem indica ao cliente que o

valor solicitado não está disponível no momento. O cliente é sugerido a tentar outro servidor ou aguardar uma atualização posterior para obter um valor mais atualizado. Ao enviar o timestamp associado à chave, o servidor permite que o cliente compare e verifique se o valor obtido é mais atual do que o valor que ele já conhece. Dessa forma, o cliente pode garantir que não obtenha um valor anterior ao que já viu.

Utilização de **threads** do **Cliente**:

O cliente utiliza threads para executar as ações PUT e GET de forma assíncrona. Isso evita bloqueios e permite que o programa seja responsivo, processando várias solicitações ao mesmo tempo. As threads garantem que as ações não interfiram umas nas outras, melhorando a eficiência e a experiência do usuário. É importante garantir a sincronização adequada ao compartilhar recursos compartilhados entre as threads para evitar problemas de concorrência. Em resumo, as threads permitem que o cliente execute várias operações de forma paralela, garantindo um programa ágil e responsivo.

O sistema do **servidor** é responsável por receber as requisições dos clientes, processá-las e fornecer as respostas apropriadas. O servidor é parte integrante de um sistema distribuído, onde existem vários servidores que armazenam os dados.

O **servidor** possui as seguintes **funcionalidades**:

- **Inicialização:** O servidor é inicializado ao definir seu endereço IP, porta e outras configurações necessárias. Ele cria um socket para receber conexões dos clientes.

- **Armazenamento de Dados (PUT):** Quando o servidor recebe uma requisição PUT, ele verifica se é o líder do sistema. Se for o líder, o servidor insere ou atualiza a informação recebida em uma tabela de hash local, associando um timestamp à chave. Se não for o líder, encaminha a requisição para o líder correspondente.

- **Replicação de Dados:** Após realizar a inserção ou atualização dos dados na tabela de hash local, o servidor líder replica essas informações nos outros servidores. Isso é feito enviando uma mensagem REPLICATION contendo a chave, o valor e o timestamp para cada servidor. Os servidores destinatários inserem essas informações em suas próprias tabelas de hash locais.

- **Recuperação de Dados (GET):** Quando o servidor recebe uma requisição GET, ele verifica se a chave está presente em sua tabela de hash local. Se estiver presente e o timestamp associado for igual ou maior ao solicitado pelo cliente, o servidor envia a mensagem GET_OK de volta ao cliente, juntamente com o valor associado à chave. Se a chave não estiver presente,

o servidor envia a mensagem NULL. Se o timestamp associado à chave for menor do que o solicitado pelo cliente, o servidor envia a mensagem TRY_OTHER_SERVER_OR_LATER, indicando que o valor solicitado não está disponível no momento.

- **Sincronização da Replicação:** O servidor líder espera receber a mensagem REPLICATION_OK de todos os servidores antes de enviar a resposta PUT_OK ao cliente. Isso garante que a replicação tenha ocorrido com sucesso em todos os servidores antes de confirmar a conclusão da operação PUT.

- **Gerenciamento de Conexões:** O servidor utiliza threads para lidar com as conexões dos clientes de forma concorrente. Cada conexão é tratada em uma thread separada, permitindo que várias conexões sejam atendidas simultaneamente sem bloquear o servidor.

- **Tratamento de Requisições:** Cada thread de conexão é responsável por receber as requisições dos clientes, analisá-las e determinar o tipo de operação (PUT, GET, REPLICATION, etc.). Com base no tipo de operação, o servidor executa as ações apropriadas, como inserir, atualizar ou buscar dados na tabela de hash local.

- **Envio de Respostas:** Após processar uma requisição, o servidor envia uma resposta ao cliente correspondente. As respostas são enviadas em formato JSON, contendo as informações necessárias, como o status da operação (PUT_OK, GET_OK, NULL, TRY_OTHER_SERVER_OR_LATER) e os valores associados à chave.

O sistema do **servidor** possui os seguintes **métodos**:

- **init():** Esse método é responsável por inicializar o servidor. Ele define o endereço IP, a porta e outras configurações necessárias. Além disso, cria um socket para receber conexões dos clientes e configura o líder inicial como o próprio servidor.
- **setPortSettings():** Esse método é responsável por configurar as portas e definir qual servidor é o líder. Ele verifica quais portas estão ativas e se algum servidor está marcado como líder. Com base nessas informações, define a porta do servidor e se ele é o líder ou não.

- **setLeaderPort():** Esse método é chamado pelo setPortSettings() e permite que o usuário insira a porta do servidor líder. Ele valida se a porta fornecida é válida e se está presente no conjunto de portas válidas dos servidores.

- **setPort():** Esse método é chamado pelo setPortSettings() e permite que o usuário insira a porta do servidor. Ele valida se a porta fornecida é válida e se não está em uso por outro servidor ativo.

- **start():** Esse método inicia o servidor. Ele faz o bind do endereço IP e da porta definidos anteriormente ao socket. Em seguida, o servidor fica em um loop infinito, aceitando conexões de clientes e iniciando uma nova thread para lidar com cada conexão.

- **handleClients(conn, addr):** Esse método é chamado para lidar com uma conexão de cliente específica. Ele recebe os dados enviados pelo cliente e analisa a mensagem recebida para determinar o tipo de operação (isLeader, PUT, GET, REPLICATION). Com base no tipo de operação, o servidor executa as ações apropriadas.

- **doPut(conn, message):** Esse método é chamado quando o servidor recebe uma requisição PUT. Se o servidor for o líder, ele insere ou atualiza a informação recebida em sua tabela de hash local, associando um timestamp à chave. Em seguida, o servidor replica essas informações nos outros servidores, enviando a mensagem REPLICATION. Se o servidor não for o líder, ele encaminha a requisição para o líder correspondente.

- **doGet(conn, message):** Esse método é chamado quando o servidor recebe uma requisição GET. Ele verifica se a chave está presente em sua tabela de hash local. Se estiver presente e o timestamp associado for igual ou maior ao solicitado pelo cliente, o servidor envia a mensagem GET_OK de volta ao cliente com o valor associado à chave. Se a chave não estiver presente, o servidor envia a mensagem NULL. Se o timestamp associado à chave for menor do que o solicitado pelo cliente, o servidor envia a mensagem TRY_OTHER_SERVER_OR_LATER.

- **doReplication(conn, message):** Esse método é chamado quando o servidor recebe uma requisição de replicação (REPLICATION). Ele insere as informações recebidas (chave, valor e timestamp) em sua própria tabela de hash local e envia a mensagem REPLICATION_OK de volta ao líder.

O **fluxo de execução** do sistema do **servidor** é o seguinte:

O servidor é iniciado e as configurações de porta e liderança são definidas. O servidor inicia o socket e entra em um loop infinito para aceitar conexões de clientes. Quando uma conexão é estabelecida, uma nova thread é iniciada para lidar com essa conexão específica. A thread de conexão recebe os dados enviados pelo cliente e analisa a mensagem recebida. Com base no tipo de operação (isLeader, PUT, GET, REPLICATION), o servidor executa as ações correspondentes.

No caso de uma requisição PUT, o servidor verifica se é o líder. Se for, insere ou atualiza a informação na tabela de hash local e replica nos outros servidores. Se não for o líder, encaminha a requisição para o líder correspondente.

No caso de uma requisição GET, o servidor verifica a presença da chave em sua tabela de hash local. Se estiver presente e o timestamp associado for igual ou maior ao solicitado pelo cliente, envia a mensagem GET_OK. Se a chave não estiver presente, envia a mensagem NULL. Se o timestamp for menor, envia a mensagem TRY_OTHER_SERVER_OR_LATER.

No caso de uma requisição de replicação (REPLICATION), o servidor insere as informações recebidas em sua tabela de hash local e envia a mensagem REPLICATION_OK de volta ao líder.

Utilização de **threads** do **Servidor**:

O servidor utiliza threads para lidar com as conexões dos clientes de forma concorrente. Cada conexão é tratada em uma thread separada, permitindo que várias conexões sejam atendidas simultaneamente sem bloquear o servidor. O uso de threads permite que o servidor seja responsivo e eficiente, pois ele pode processar múltiplas requisições concorrentemente.