Nome: João Augusto Zanardo de Lima **RA:** 11201920195

Link para o Vídeo do funcionamento: https://youtu.be/jsdSPzh-4Xw

Funcionalidades do Cliente:

A classe "Cliente" implementa uma aplicação cliente para interagir com um sistema distribuído composto por três servidores que armazenam pares chave-valor de forma replicada e consistente, utilizando o protocolo TCP para a comunicação entre cliente e servidores.

A função "inicializar()" é responsável por coletar os IPs e portas dos servidores do sistema distribuído. O usuário pode optar por usar os IPs padrões "127.0.0.1" e as portas "10097", "10098" e "10099" ou inserir novos IPs e portas manualmente. O cliente então apresenta um menu com três opções:

1. [0] Sair: Permite ao usuário encerrar o programa.
2. [1] PUT: Permite ao usuário inserir um par chave-valor no sistema distribuído.
3. [2] GET: Permite ao usuário obter o valor associado a uma chave no sistema distribuído.

Ao escolher a opção PUT, o cliente coleta a chave e o valor e inicia uma nova thread para enviar a requisição PUT para um dos servidores escolhidos aleatoriamente. A resposta do servidor é tratada, e se a operação for bem-sucedida (PUT_OK), o cliente atualiza o timestamp associado à chave no mapa "timeStamp".

Já no caso da opção GET, o cliente coleta a chave de busca e inicia uma nova thread para enviar a requisição GET para um dos servidores escolhidos aleatoriamente. A resposta do servidor é tratada, e se a chave for encontrada (GET_RESPONSE) com um valor associado, o cliente atualiza o timestamp da chave no mapa "timeStamp".

O cliente mantém um mapa de timestamps chamado "timeStamp", que armazena o timestamp de cada chave inserida ou acessada no sistema. Isso permite ao cliente saber o timestamp mais recente associado a cada chave e evitar retornar valores desatualizados do sistema.

As operações PUT e GET são executadas em threads separadas, o que permite ao cliente interagir com os servidores de forma concorrente. Isso torna o cliente mais responsivo e capaz de continuar operando enquanto aguarda respostas dos servidores.

Para se conectar aos servidores, o cliente utiliza a classe Socket e envia as mensagens através de fluxos de entrada e saída de objetos (ObjectInputStream e ObjectOutputStream). As respostas dos servidores são tratadas para garantir o sucesso das operações PUT e GET, e também é verificado se a chave solicitada existe no sistema antes de realizar a operação GET.

A função "connectToServer()" estabelece a conexão com o servidor escolhido, utilizando o IP e a porta correspondentes. Essa função é utilizada pelas funções "sendPut()" e "sendGet()" para realizar a comunicação com o servidor.

As funções "messagePUT_OK()" e "messageGET_OK()" são responsáveis por formatar as mensagens de resposta do servidor após a realização das operações PUT e GET, respectivamente. As mensagens formatadas são exibidas no console do cliente.

Funcionalidades do Servidor:

O servidor é capaz de realizar operações de inserção (PUT) e recuperação (GET) dos dados armazenados. Além disso, ele é responsável por replicar as operações de inserção para manter a consistência dos dados em outros servidores do sistema.

A tabela "keyValueStorageTable" é um mapa (*HashMap*) utilizada para armazenar os pares chave-valor. Cada chave é uma string e cada valor é um objeto "Mensagem" que contém informações adicionais relacionadas ao valor, como timestamp e remetente.

A função "initialize()" é responsável por inicializar o servidor. Ela cria um "ServerSocket" para aguardar conexões dos clientes. Através de um loop infinito, o servidor fica escutando novas requisições dos clientes e para cada requisição, inicia uma nova thread denominada "ThreadAtendimento" para atender o cliente de forma concorrente.

A classe interna "ThreadAtendimento" é responsável por tratar as requisições dos clientes. Ela recebe uma mensagem do cliente e processa o método da mensagem (PUT, GET ou REPLICATION) para executar a ação correspondente.

A função "put()" realiza a operação PUT no servidor. Se o servidor for o líder, ele adiciona o par chave-valor na tabela de armazenamento local. Caso não seja o líder, a operação PUT é encaminhada para o líder através da função "sendToLeader()". Após a execução da operação PUT, o servidor realiza a replicação dessa operação nos outros servidores através da função "registryReplication()".

Já o método "get()" é acionado quando o servidor recebe uma mensagem GET de um cliente, assim, ele busca o valor associado à chave na tabela de armazenamento local. Se a chave for encontrada, o servidor retorna o valor associado à chave. Caso contrário, é retornada uma mensagem indicando que a chave não foi encontrada.

Para replicar as mensagens, foi desenvolvido a função "replicate()" que realiza a replicação de uma operação PUT nos outros servidores do sistema. Ela adiciona o par chave-valor na tabela de armazenamento do servidor e envia uma mensagem de replicação para cada um dos outros servidores. A replicação ocorre de forma concorrente, com uma thread sendo criada para cada servidor a ser replicado.

A função "registryReplication()" registra a replicação de uma operação PUT nos outros servidores. Ela percorre a lista de portas dos outros servidores e estabelece uma conexão com cada um deles. Em seguida, envia a mensagem de replicação para o servidor e aguarda a resposta. Se a replicação for bem-sucedida, o servidor registra o sucesso.

Para encaminhar uma operação PUT para o líder do sistema usamos o método "sendToLeader()" que cria uma conexão com o líder e envia a mensagem PUT recebida do cliente. Em seguida, aguarda a resposta do líder e a envia de volta para o cliente.

Outras funções auxiliares como "createMessagePUT()", "createMessageGET()", e "createMessageREPLICATION()" são responsáveis por criar mensagens informativas sobre as operações PUT, GET e REPLICATION, que são exibidas no console do servidor.

Funcionalidades do Mensagem:

A classe Mensagem é uma estrutura de dados essencial para a comunicação entre os componentes de um sistema distribuído simulando o funcionamento de um serviço de mensageria. Ela é utilizada para representar mensagens que são trocadas entre os componentes de um sistema distribuído. Ela encapsula as informações importantes de cada mensagem, tornando a comunicação mais organizada e estruturada.

Os atributos method, key, value, timestamp e sender são utilizados para armazenar informações-chave de uma mensagem:

- method: Representa o método associado à mensagem, como PUT, GET ou REPLICATION.
- key: Representa a chave associada à mensagem, que é utilizada em operações PUT e GET.
- value: Representa o valor associado à chave na operação PUT ou REPLICATION.
- timestamp: Armazena o timestamp associado à criação da mensagem, indicando o momento em que foi gerada.
- sender: Representa o remetente da mensagem, ou seja, o endereço do componente que enviou a mensagem.

Ao implementar a interface Serializable, a classe Mensagem permite que seus objetos sejam convertidos em bytes, tornando-os serializáveis e, assim, possam ser transmitidos através da rede ou armazenados em arquivos.

Referências:

https://www.geeksforgeeks.org/transfer-the-file-client-socket-to-server-socket-in-java/

https://stackoverflow.com/questions/67430428/problem-when-sending-messages-from-client-socket-to-server-socket

https://codedamn.com/news/java/transferring-files-from-client-to-server-socket-in-java

Java.io.FileInputStream.read() Method (tutorialspoint.com)

https://www.geeksforgeeks.org/java-fileinputstream-read-method-with-examples/

Java Examples & Tutorials of FileInputStream.read (java.io) | Tabnine

https://www.devmedia.com.br/use-a-serializacao-em-java-com-seguranca/29012

https://pt.stackoverflow.com/questions/88270/qual-a-finalidade-da-interface-serializable

https://www.devmedia.com.br/hashmap-java-trabalhando-com-listas-key-value/2981s1