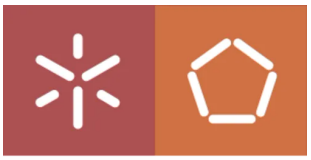


Universidade do Minho
Escola de Engenharia

Sistemas Distribuídos
Licenciatura em Engenharia Informática
Universidade do Minho
Grupo 27
Link GitHub https://github.com/a104360/sd


David Figueiredo
(A104360)


Diogo Ferreira
(A104266)


João Carvalho
(A104533)


Rui Cruz
(A104355)

Contents

1 Introdução

Este projeto tem como objetivo a implementação de um serviço de armazenamento de dados partilhado, proporcionando uma solução eficiente e robusta para a inserção e consulta de informações. A arquitetura baseia-se num servidor central que mantém os dados em memória e permite o acesso remoto por parte dos clientes através de sockets TCP. A interação é realizada utilizando um formato chave-valor, garantindo assim simplicidade e flexibilidade na gestão de informação.
O servidor é projetado para atender múltiplos clientes simultaneamente, implementando estratégias de concorrência que buscam otimizar o desempenho. O nosso trabalho possui todas as funcionalidades básicas descritas no enunciado, tal como a operação de leitura condicional, assim sendo, este sistema oferece uma solução prática e escalável para o armazenamento de informação em ambientes destribuidos.

2 Client

Client é a entidade que vai interagir com o servidor. Ao se conectar com o servidor o cliente vai passar pela fase de autenticação onde terá a opção de se registar ,caso seja a primeira vez que interage com o servidor, a opção de fazer login caso já tenha criado conta, a opção de mudar a password e as opções de sair (caso não esteja autenticado) ou ir para o menu principal (se estiver autenticado).

Quando o cliente se regista, os seus dados são enviados para o servidor que verifica se o username já está em uso. Caso não exista nenhum tipo de conflito os dados do cliente são armazenados na base de dados. Se o cliente optar pela opção de log in, o servidor verifica se a password introduzida existe na base de dados. Caso a password exista, o servidor vai pegar na lista de clientes com a password introduzida e verificar se alguem possui as credenciais iguais às que foram introduzidas. Caso o cliente selecione a opção de alterar a password, este deve enviar as suas credenciais atuais ao servidor juntamente com a nova password. Em qualquer um dos cenários mencionados o servidor irá notificar o cliente informando que a operação foi bem/mal sucedida dizendo ainda, no caso de ser mal sucedida, o motivo pelo qual não foi possível realizar a operação. A opção de exit aparecerá para o cliente no caso dele ainda não ter iniciado sessão, permitindo o cliente fechar conexão com o servidor sem antes iniciar sessão. Caso o cliente já tenha iniciado sessão ou registado uma nova conta, a opção de exit será substituída pela opção que levará o cliente ao menu principal.

Após o inicio de sessão o cliente é capaz de enviar dados para o servidor através do put e do multiput. O put vai permitir ao cliente inserir uma key e um value que serão enviados para o servidor para que este os guarde em memória. O multiput, tal como o nome sugere, permite ao cliente enviar vários pares chave-valor ao servidor sem necessitar de realizar várias vezes a operação put. Quando o cliente opta pelo multiput deve fornecer inicialmente a quantidade de valores que vai querer enviar para o servidor, entrando num

ciclo até ter inserido a quantidade que forneceu. Por outro lado, se pretender obter valores armazenados no servidor, o cliente pode utilizar o get, o multiget e o getwhen . O get e o multiget têm uma lógica semelhante ao put e ao multiput, mas no caso do get, o cliente vai apenas enviar ao servidor a chave e o servidor vai à sua base de dados procurar o valor correspondente à chave fornecida. A outra funcionalidade disponibilizada para o cliente, denominada como getwhen, permite que este obtenha o valor de uma chave que ele forneça apenas quando uma segunda chave fornecida possua o valor que o cliente indicar. Por fim o cliente possui uma opção que lhe permite terminar conexão com o servidor. Para qualquer operação que o utilizador realize, este receberá a informação de que a operação foi bem sucedida ou não, no caso do get e do multiget, se a chave introduzida não existir, o servidor irá enviar para o cliente o valor como nulo.

2.1 ClientLibrary

A interface ClientLibrary possui os métodos que devem ser implementados pelo Client, sendo este o put, o get, o multiput, o multiget e o getwhen.

3 FramedConnection

A classe Connection desempenha o papel de intermediária na comunicação entre clientes e o servidor, gerenciando o envio e a receção de mensagens através do socket associado. Esta classe permite transmitir mensagens formatadas adequadamente ou processar os dados recebidos pelo socket, convertendo-os num formato compreensível para o cliente/servidor. Para realizar essas tarefas, a classe utiliza DataInputStream e DataOutputStream, facilitando a troca de informações entre as partes envolvidas na conexão.

4 Data

A classe Data é uma estrutura de armazenamento utilizada pelo servidor para gerenciar pares chave-valor. Esta classe utiliza um mapa (HashMap) para armazenar os dados e um conjunto de mecanismos de sincronização, ReentrantLock e Condition, para garantir a consistência e coordenação entre as threads que acessam os dados. A sincronização é realizada com um ReentrantLock, que protege todas as operações de leitura e escrita no mapa, garantindo exclusão mútua. Um objeto Condition (waitWhen) associado ao lock é usado para coordenar threads em operações de leitura condicional (getWhen), permitindo que as threads esperem até que certas condições sejam atendidas.

5 Server

O Server é a entidade responsável por gerenciar conexões e comunicações com múltiplos clientes simultaneamente. Ele coordena operações de autenticação, armazenamento e recuperação de dados, garantindo segurança e desempenho através de um controle sincronizado

de threads. O servidor é projetado para suportar um número limitado de sessões ativas, utilizando Locks e Conditions para sincronizar o acesso e permitindo que novos clientes aguardem quando o limite é atingido.

O servidor utiliza um ServerSocket para escutar e aceitar conexões na porta especificada. Cada nova conexão aceite é delegada a uma thread de trabalho (ServerWorker), que trata individualmente das operações solicitadas pelo cliente conectado. O PasswordManager é utilizado para gerenciar a autentiticação de clientes, possibilitando o registo de novos usuários, autenticação de usuários existentes e a troca de password. As credenciais de cada cliente, compostas por nome de usuário e password, são armazenadas numa estrutura map.

O repositório de dados do servidor (Data) é responsável por armazenar pares chave-valor. Este repositório suporta operações de armazenamento e recuperação, realizadas pelas threads de trabalho para atender às solicitações dos clientes. As operações de armazenamento incluem o PUT, que insere um par chave-valor, que permite inserir um par numa única operação. Para recuperação de dados, são oferecidas as operações GET, que retorna o valor associado a uma chave específica e GETWHEN, que fornece o valor de uma chave apenas quando outra chave associada possui um valor específico. As operações MULTIPUT e MULTIGET não estão na classe Data pois não são atómocas, sendo assim a operação de MULTIPUT é feita através da utilização de n operações PUT e a operação MULTIGET é feita através da utilização de n operações GET.

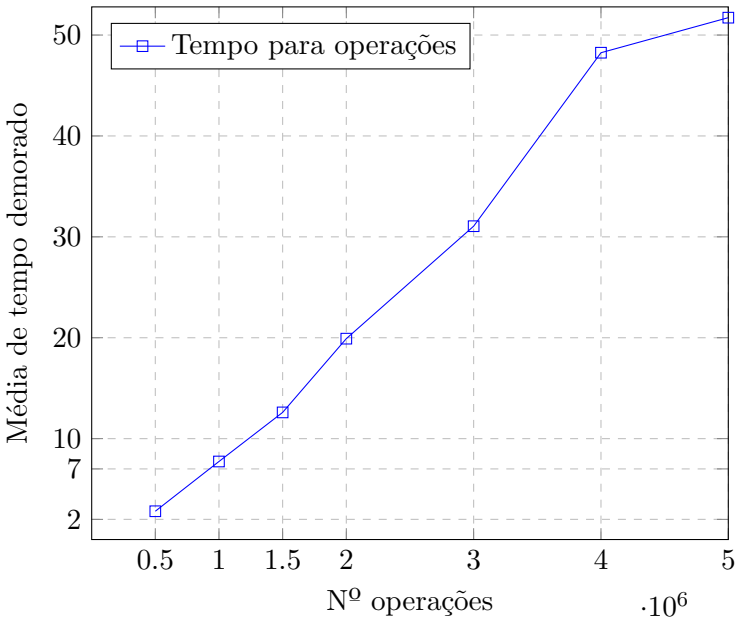
Quando um cliente conecta-se ao servidor, o ServerWorker conduz o processo de autenticação. O cliente pode registar uma nova conta, fazer login, trocar a senha ou encerrar a conexão. Durante o registro, o servidor verifica se o nome do user já está em uso antes de armazenar as credenciais. No login, as credenciais fornecidas são verificadas contra as informações armazenadas. A troca de senha exige as credenciais atuais e a nova password, com o servidor validando as informações antes de concluir a operação. Todas essas interações são realizadas por meio de uma FramedConnection, que facilita a troca de mensagens entre o cliente e o servidor.

Após a autenticação, o cliente pode realizar operações de armazenamento e recuperação de dados no servidor. Cada operação é acompanhada de uma resposta indicando sucesso ou falha, incluindo o motivo do erro, caso ocorra. O cliente também pode encerrar a sessão, libertando a conexão com o servidor. O Server também possui funcionalidades que permitem monitorar o estado atual do mesmo, como o número de sessões ativas, credenciais armazenadas e conteúdo do repositório de dados.

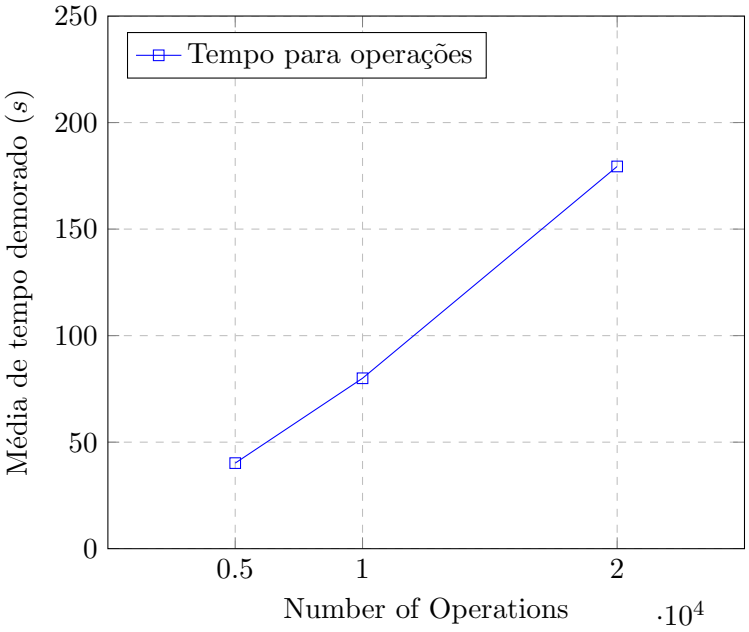
O servidor trabalha em conjunto com a classe Client para oferecer um sistema completo de comunicação e armazenamento distribuído. Enquanto o cliente envia requisições de autenticação e operações de dados, o servidor processa as solicitações e retorna as respostas apropriadas, promovendo uma interação eficiente e segura.

6 Testes

Média de tempo demorado a efetuar operações de put simultaneamente



Média de tempo de operação simultanea de get e put



Os testes apresentam os valores de tempo demorado para a execução do respetivo número de operações, de put simultâneos ou put e get, executados simultaneamente. Em ambos os testes foram utilizados 2 clientes. Concluimos através destes testes, por comparação da ordem de grandeza, que as operações de get ao servidor são mais custosas, temporalmente falando. Isto deve-se ao facto de ser necessário o cliente esperar pela resposta do servidor, que pode ser afetado pelo número de clientes a utilizar o servidor. Também é possível concluir que o servidor é capaz de garantir exclusão mútua, pois é capaz de realizar os testes, acedendo a secções críticas sem comprometer a informação.

Outros testes foram realizados, para testar a capacidade do servidor de atender clientes, que não puderam ser completos com consistência, para uma simulação de pouco menos de 100 clientes, realizando 100 inserções em simultaneo. Era possível perceber, através desses testes, que existiam clientes que não eram atendidos por entrarem num deadlock. Por isso, é possível afirmar que a solução que apresentamos não apresenta uma grande escalabilidade.

7 Conclusão

Este trabalho permitiu-nos consolidar os conhecimentos adquiridos em Sistemas Distribuídos por meio da implementação de um sistema de armazenamento de dados robusto e eficiente. Durante o desenvolvimento, aplicámos conceitos fundamentais, como o uso de sockets TCP, locks, e threads, observando como estes elementos são cruciais em cenários reais de computação distribuída. A experiência prática reforçou a importância desses conceitos, preparando-nos para desafios futuros no âmbito académico e profissional.

Apesar de termos implementado com sucesso a maioria das funcionalidades descritas no enunciado, o requisito de suporte completo a clientes multithreaded não foi alcançado. A ausência dessa funcionalidade reflete uma oportunidade de melhoria, que poderia ser explorada futuramente através da aplicação de estratégias mais avançadas de sincronização e gestão de concorrência

No geral, este trabalho demonstrou que a estrutura desenvolvida é prática e preparada para atender às necessidades de ambientes distribuídos. A implementação bem-sucedida da maioria das funcionalidades propostas reforça que os conhecimentos teóricos adquiridos em aula podem ser aplicados de forma eficaz e inovadora, permitindo o desenvolvimento de sistemas funcionais e alinhados às exigências do mundo real.