Nome: Erick Henrique Silva RA: 11201812308

Sobre o projeto:

O objetivo do código é simular e implementar, de maneira simplificada, um sistema distribuído, que contenha um cliente, um servidor centralizado (middleware) que realizará a distribuição de tarefas e os servidores que realizarão os trabalhos distribuídos pelo middleware.

O cliente deve ser capaz de selecionar um arquivo e envia-lo ao servidor principal, onde o arquivo será salvo e realizado o seu backup sem que o cliente tenha que se preocupar com estas tarefas, dando a ele transparência de acesso, localização, replicação, concorrência e distribuição.

O middleware deve ser capaz de receber as requisições e a partir delas entender qual o servidor mais adequado para receber a requisição que será processada, levando em conta a capacidade e o uso de cada um dos servidores, procurando sempre alocar aqueles que estão mais livres para poder garantir uma melhor experiência ao cliente.

Os servidores de trabalhos devem ser capazes de se conectar ao middleware, indicando que estão prontos para receber tarefas e informando as suas capacidades de trabalho para que possam receber uma quantidade de tarefas mais ou menos intensa.

Em nosso projeto teremos como função principal o envio de arquivos, com a capacidade de realizar backups entre os servidores. Assim, toda vez que o cliente enviar um arquivo ele deve ser salvo como arquivo principal em um servidor e logo em seguida uma réplica desse arquivo deve ser salva em outro servidor.

Comunicação:

Toda a comunicação, entre cliente, middleware e servidores, foi realizada utilizando a tecnologia de sockets. Para isso foi necessário criar uma forma de comunicação que seja coesa, tenha baixo acoplamento e seja simples, a fim de facilitar a comunicação e escalabilidade do sistema.

Para essa comunicação foi definido o padrão de comunicação via objetos, onde teremos em toda transmissão o envio de mensagens que seguirão um padrão, contendo o campo do canal, que indica qual a ação/objetivo daquela comunicação e outro campo contendo o corpo da requisição, que terá o que será utilizado para processamento, como podemos ver no exemplo da imagem 1.

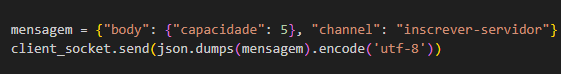


Imagem 1: Exemplo de corpo de uma requisição

Agentes do sistema:

- Cliente:

O cliente deve ser capaz de selecionar e enviar um arquivo para o middleware, sem que conheça das complexidades de transporte, backup, armazenamento, distribuição, etc. Deve apenas saber que a sua operação foi concluída com sucesso.

Para isso foi desenvolvido um código que ao ser iniciado se conecta ao middleware, após isso entrará em um loop para digitar o arquivo que deseja enviar ou sair. Ao selecionar o arquivo o script irá ler o arquivo e armazenar em uma mensagem seguindo os padrões pré-estipulados, enviando para o canal enviar-arquivo e com o nome do arquivo e os seus dados no corpo.

Após o envio com sucesso o usuário poderá enviar outro arquivo, repetindo o processo ou sair da execução, o que encerra a conexão com o middleware e finaliza a execução.

- Middleware:

A responsabilidade do Middleware é realizar a distribuição das requisições entre os servidores a fim de encontrar um equilíbrio entre os servidores. Para que as requisições sejam sempre atendidas pelos servidores que estão menos atarefados e conseguir responde-las o mais rapidamente possível.

Para a escolha do servidor que irá processar a requisição e o backup foi utilizado um algoritmo inspirado no algoritmo de least connections. Ao iniciar cada servidor envia para o middleware uma mensagem indicando que ele é um servidor de trabalho e qual a sua capacidade de carga.

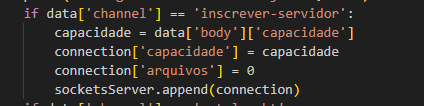


Imagem 2: Código que inscreve o servidor para receber trabalhos

Após isso o middleware adiciona o servidor em uma lista de servidores, onde cada elemento da lista é um objeto que contem o socket da conexão com o servidor, a sua capacidade, o uso atual e a quantidade de arquivos que o servidor contém.

Ao receber uma requisição o middleware irá consultar a lista de servidores disponíveis, encontrar os dois menos atarefados e com menos arquivos em relação a sua capacidade e aloca-los para a tarefa, atualizando os pesos dos servidores e criando uma tarefa em uma lista de tarefas, para poder devolver a requisição para o cliente quando for concluída.

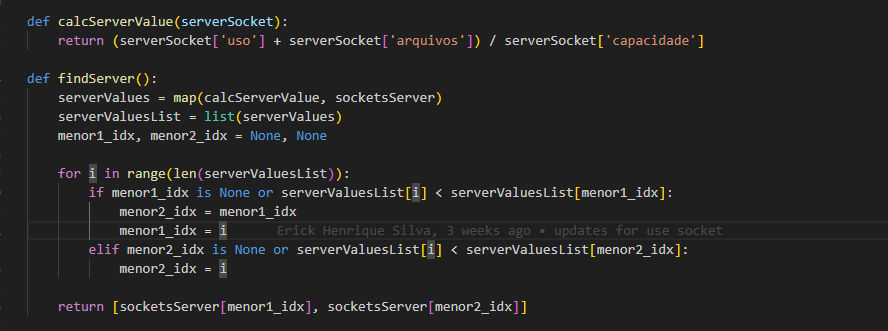
 Imagem 3: Código para encontrar os 2 melhores servidores para a tarefa



Imagem 4: Código para alocar uma tarefa nos servidores

Após o envio o middleware terá a tarefa armazenada em uma lista de tarefas pendentes, para que quando o servidor informar que a tarefa foi concluída seja possível resgatar de onde a requisição foi feita e informar ao cliente que o envio do seu arquivo foi concluído.

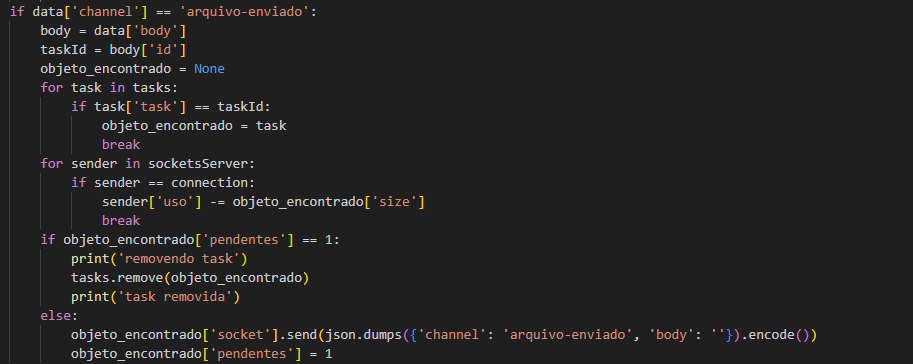


Imagem 5: Código para responder ao cliente que a sua requisição foi concluída

O middleware também foi desenvolvido para conseguir tratar de requisições de múltiplos tamanhos, para isso foi desenvolvida duas técnicas e funcionalidades, um socket pode definir qual será o tamanho das mensagens que irão transitar nas comunicações e o middleware também irá tratar para que, caso cheguem mensagens incompletas, sejam esperadas novas transmissões a fim de completar a mensagem.

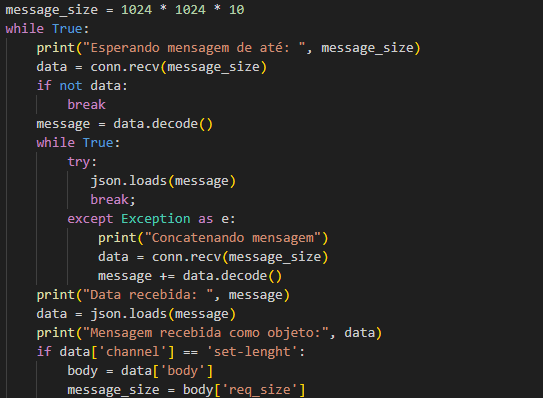


Imagem 6: Código para responder processar mensagens incompletas e definir o tamanho das mensagens

- Servidor:

O servidor é responsável por realizar o processamento de cada requisição entregue pelo middleware, para isso ele irá se conectar ao servidor, informar que está apto a receber requisições e após isso irá esperar por requisições.

Ao receber uma requisição para salvar o arquivo irá salvar o arquivo em uma pasta com o ID da requisição e retornar que obteve sucesso no processamento.

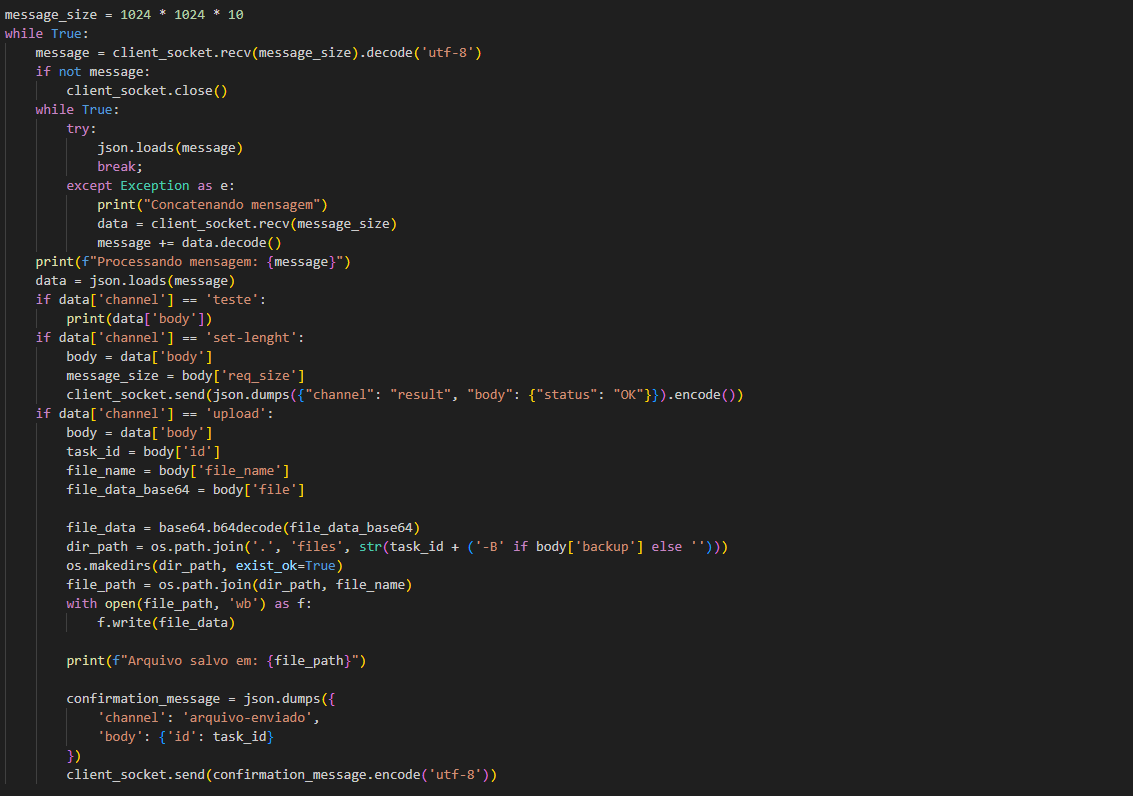


Imagem 7: Código para processar mensagens no servidor

Funcionamento:

Para o funcionamento do sistema deve ser rodado inicialmente o script main.py, com o comando python main.py, após isso deve ser acessado cada uma das pastas com nome server e rodar o script worker.py em cada uma, com o comando python worker.py, por fim deve ser iniciando o cliente com o comando python client.py, novamente no diretório principal.

Resultado:

Foi possível criar um protótipo de um sistema distribuído com um middleware que recebe requisições dos clientes e distribui elas entre os servidores, avaliando as suas capacidades a fim de prover uma maior qualidade ao cliente.

A partir da lógica desenvolvida também foi possível apresentar um sistema que é escalável, podendo conectar mais servidores no middleware sem que seja realizada mudanças no código.

Melhorias:

Para futuras melhorias ao sistema seria útil adicionar sistemas mais eficientes para a alocação dos servidores, como sistemas de árvores ordenadas pelos servidores mais livres, sendo rebalanceadas sempre que um servidor sofre alguma alteração, assim seria possível melhorar a eficiência do middleware.

Também seria possível adicionar novas métricas às informações de cada servidor, como a sua latência, largura de banda e Jitter, com o objetivo de avaliar de maneira mais assertiva para qual servidor deve ser alocada uma tarefa.

Referências utilizadas além da disciplina:

<https://samwho.dev/load-balancing/>