Trabalho Prático (CI1061)

Lucas Emanuel de Oliveira Santos

3 de dezembro de 2025

1 Introdução

Este relatório descreve a implementação de um sistema de arquivos distribuído baseado na arquitetura Cliente-Servidor. O sistema foi desenvolvido utilizando a linguagem Python.

2 Decisões de Projeto

Durante o desenvolvimento do sistema, foram tomadas decisões para definir o funcionamento dos servidores réplica, organizar o armazenamento dos arquivos e simplificar a comunicação entre os componentes.

2.1 Configuração e Funcionamento das Réplicas

O comportamento das réplicas foi definido de forma centralizada no arquivo de configurações. Nele são especificados tanto o número total de réplicas (`NUM_REPLICAS`) quanto as portas que as réplicas devem utilizar. As portas são geradas automaticamente a partir de uma porta base (`BASE_PORT`) igual a 9000, formando uma lista contínua. Por exemplo, para:

$$\texttt{NUM\_REPLICAS} = 3$$

o sistema define as seguintes portas:

9000, 9001, 9002

Cada instância de réplica deve obrigatoriamente ser iniciada utilizando uma das portas definidas no arquivo de configuração. Embora o sistema não permita o uso de portas fora dessa lista, é essencial que todas as portas configuradas sejam efetivamente utilizadas. Caso alguma réplica não seja iniciada, ocorrerá uma falha no processo de replicação dos arquivos.

2.2 Organização do Armazenamento

O servidor primário e as réplicas utilizam estruturas de diretórios semelhantes e independentes. No servidor primário, todos os arquivos são armazenados no diretório raiz `servidor`, onde cada cliente possui um subdiretório próprio:

`servidor/<client_id>/<arquivo>`

Cada réplica também possui seu diretório raiz, nomeado como:

`replica<id>`

O identificador da réplica é determinado a partir da porta na qual ela está executando, utilizando a porta base como referência, sendo definido por:

$$\text{id\_réplica} = (\text{porta} - \text{porta\_base}) + 1.$$

Assim como no servidor primário, cada réplica cria um diretório para cada cliente.

2.3 Modelo de Servidor

O servidor primário foi implementado como um servidor **iterativo**, processando apenas uma conexão de cliente por vez. Isso significa que um cliente só pode iniciar uma operação após a conexão anterior ser encerrada.

3 Detalhes da Implementação

3.1 Arquitetura do Sistema

A implementação foi modularizada em quatro arquivos principais:

- **config.py** — Centraliza todas as constantes globais do sistema, como tamanhos de buffers, número de réplicas e portas utilizadas.
- **cliente.py** — Implementa o cliente, responsável por enviar comandos ao servidor primário e interagir diretamente com o usuário.
- **servidor.py** — Implementa o servidor primário, encarregado de processar as requisições dos clientes, armazenar arquivos localmente e coordenar a replicação para as réplicas.
- **replica.py** — Implementa os servidores réplica, que recebem os arquivos enviados pelo servidor primário e os armazenam para garantir redundância.

3.2 Protocolo de Comunicação

Para coordenar as operações entre cliente, servidor primário e servidores réplica, foi definido um protocolo simples baseado em troca de mensagens de controle seguidas, quando necessário, pela transmissão dos bytes do arquivo. O protocolo contempla três operações principais: *upload*, *list* e *replicação*.

1. **Operação de Upload**
   O cliente inicia o envio de um arquivo transmitindo um cabeçalho no formato:

   `UPLOAD <client_id> <filename> <filesize>`

   O servidor primário responde com `SEND_FILE`, autorizando o cliente a enviar os bytes do arquivo. Após receber todo o conteúdo, o servidor armazena o arquivo localmente e inicia o processo de replicação. Ao final, o cliente recebe:

   - `UPLOAD_SUCCESS`, caso o arquivo tenha sido salvo localmente e replicado com sucesso em todas as réplicas;
   - `UPLOAD_FAILURE`, caso não consiga salvar o arquivo localmente ou caso alguma réplica apresente erro.

2. **Operação de Listagem**
   Para consultar os arquivos já enviados, o cliente envia:

   `LIST <client_id>`

   O servidor primário recupera os arquivos presentes no diretório associado ao cliente e devolve uma string contendo a lista de nomes. Caso o diretório esteja vazio ou não exista, uma mensagem informativa é retornada.

3. **Operação de Replicação**
   Após receber um arquivo, o servidor primário atua como cliente e estabelece uma conexão independente com cada réplica. Para cada replicação, envia o cabeçalho:

   `REPLICA <client_id> <filename> <filesize>`

   A réplica responde com `SEND_FILE`, autorizando a transferência dos dados. Quando o recebimento termina, a réplica tenta salvar o arquivo localmente e responde:

   - `REPLICA_SUCCESS`, caso o arquivo seja salvo corretamente;
   - `REPLICA_FAILURE`, caso algum erro ocorra.

   O servidor primário utiliza essas respostas para decidir se envia ao cliente final um `UPLOAD_SUCCESS` ou `UPLOAD_FAILURE`.

4 Limitações da Implementação

Apesar de o sistema cumprir as funcionalidades principais propostas, algumas limitações surgem devido ao modelo simplificado adotado. A seguir são apresentadas as principais restrições identificadas durante o desenvolvimento.

4.1 Ausência de Tratamento Robusto de Erros

O sistema não possui um mecanismo estruturado de tratamento de falhas. Caso ocorra qualquer erro de conexão — seja entre cliente e servidor primário ou entre o servidor primário e as réplicas — a operação é imediatamente interrompida. Não há tentativas de reconexão, detecção refinada de erros ou estratégias de recuperação.

4.2 Processo de Replicação Não Independente

A replicação é executada de forma sequencial e dependente entre as réplicas. Se uma réplica falha durante o processo (por exemplo, a primeira da lista), o servidor primário interrompe toda a operação e retorna um `UPLOAD_FAILURE` ao cliente, sem tentar replicar o arquivo para as demais réplicas.

4.3 Inconsistência Após Falhas e Ausência de Rollback

Não existe tratamento pós-falha para restaurar o estado do sistema. Quando ocorre um erro durante um upload, o cliente recebe um `UPLOAD_FAILURE`, mas o arquivo pode já ter sido salvo no servidor primário antes da falha. Assim, comandos como `list` podem exibir arquivos que o usuário acredita não terem sido armazenados. O mesmo vale para as réplicas: algumas podem ter recebido o arquivo enquanto outras não, resultando em estados inconsistentes.