Documentação Completa do Sistema de Arquivos Distribuído BigFS

wender13

30 de junho de 2025

Sumário

1 Introdução

O **BigFS** é uma simulação funcional de um sistema de arquivos distribuído, projetado para demonstrar na prática os princípios de escalabilidade, tolerância a falhas, replicação e gerenciamento de dados em larga escala. A arquitetura final adota um modelo de **3 camadas** (Cliente-Gateway-Backend), que separa claramente a interface do usuário, a lógica de negócio e o armazenamento físico, inspirando-se em conceitos de sistemas robustos como HDFS e na usabilidade de ferramentas modernas como o Minio Client.

Este documento detalha a estrutura, o funcionamento e a implementação de cada componente do sistema.

2 Estrutura do Projeto

A organização do projeto foi projetada para separar o código-fonte de outros artefatos, seguindo as melhores práticas de desenvolvimento.

```
/BigFS/
 project/
    bigfs.proto
    bigfs_pb2.py
    bigfs_pb2_grpc.py
    client.py
    gateway_server.py
    metadata_server.py
    storage_node.py

 gateway_temp/

 storage_XXXX/ (ex: storage_50052)
```

3 Guia de Instalação e Configuração

Este guia é compatível com Linux, macOS e Windows (usando PowerShell ou WSL).

3.1 Pré-requisitos

- Python 3.8+
- Git

3.2 Passos de Instalação

1. **Obter o Código-Fonte:** Clone o repositório do projeto.

   ```
   git clone https://github.com/wender13/BigFS.git
   cd BigFS/
   ```

2. **Criar o Ambiente Virtual:** Na raiz do projeto (`/BigFS/`).

No Linux ou macOS: `python3 -m venv venv` e depois `source venv/bin/activate`

No Windows (PowerShell): `python -m venv venv` e depois `.\venv\Scripts\Activate.ps1`

3. **Instalar as Dependências:** Com o ambiente virtual ativado:

```
pip install grpcio grpcio-tools
```

4. **Compilar o Contrato gRPC:** Este passo traduz o `.proto` em código Python e é crucial.

```
python3 -m grpc_tools.protoc -Iproject --python_out=project
--grpc_python_out=project project/bigfs.proto
```

3.3 Análise do Comando de Compilação

O comando `protoc` é a ponte entre o plano e o código.

```
python3 -m grpc_tools.protoc -Iproject --python_out=project
--grpc_python_out=project project/bigfs.proto
```

`-m grpc_tools.protoc` Invoca o compilador do ambiente virtual.

`-Iproject` Define `project/` como o diretório de busca para arquivos `.proto`.

`-python_out=project` Gera o `bigfs_pb2.py` (as mensagens) dentro de `project/`.

`-grpc_python_out=project` Gera o `bigfs_pb2_grpc.py` (os serviços) dentro de `project/`.

`project/bigfs.proto` O arquivo-fonte a ser compilado.

4 Arquitetura e Componentes

O BigFS opera em um modelo de 3 camadas para máxima flexibilidade.

4.1 Cliente (`client.py`)

A interface inteligente e o orquestrador ativo das operações. Oferece um shell interativo (estilo Minio `mc`) para o usuário, abstraindo toda a complexidade do sistema. Sua função é traduzir comandos simples (`cp`, `ls`) em chamadas de API para o Gateway.

4.2 Gateway Server (`gateway_server.py`)

A camada intermediária e o ponto de entrada único para o cliente. Ele recebe os arquivos completos, assume o trabalho pesado de **particioná-los em chunks**, e orquestra a comunicação com o backend para armazená-los de forma distribuída e replicada.

4.3 Metadata Server (`metadata_server.py`)

O cérebro do backend. Gerencia a estrutura de diretórios em uma **árvore de Inodes** e mantém o mapa que associa cada arquivo aos seus chunks e suas respectivas localizações (primário e réplicas). Ele monitora a saúde dos Storage Nodes via **heartbeats**.

4.4 Storage Node (`storage_node.py`)

O músculo do backend. Sua única função é armazenar e servir chunks de dados. Ele se reporta ao Metadata Server e, quando atua como nó primário em uma escrita, é responsável por executar a **replicação assíncrona** dos dados para outros nós.

5 Funcionalidades e Fluxos de Operação

5.1 Fluxo de Escrita (Upload)

1. **Cliente → Gateway:** O cliente envia o arquivo inteiro para o Gateway via um *stream* gRPC.
2. **Gateway → Metadata Server:** O Gateway, após receber o arquivo, pede um plano de escrita ao Mestre. O Mestre responde com um mapa (Primário + Réplicas para cada chunk).
3. **Gateway → Storage Nodes:** O Gateway particiona o arquivo e envia cada chunk para seu nó Primário designado, junto com a lista de suas réplicas.
4. **Storage Node Primário → Réplicas:** O nó primário salva o chunk e inicia threads para replicar os dados para os outros nós.

5.2 Tolerância a Falhas e Replicação

A estratégia de sobrevivência do sistema se baseia em três pilares:

- **Redundância:** Cada *chunk* é replicado ‘N‘ vezes em nós distintos, garantindo a **durabilidade** dos dados contra falhas de hardware.
- **Detecção:** O ‘Metadata Server‘ detecta falhas de ‘Storage Nodes‘ pela ausência de **heartbeats**.
- **Recuperação (Failover):** Durante uma leitura, se o cliente não consegue contatar um nó primário, ele automaticamente tenta uma das réplicas, garantindo a **alta disponibilidade** do sistema.

6 Guia de Uso

6.1 Iniciando o Cluster

Execute cada comando em um terminal separado, a partir da raiz do projeto `/BigFS/`.

Terminal 1 (Mestre): `python3 project/metadata_server.py`

Terminal 2 (Escravo 1): `python3 project/storage_node.py localhost 50052 localhost:5005`

Terminal 3 (Escravo 2): `python3 project/storage_node.py localhost 50053 localhost:5005`

Terminal 4 (Gateway): `python3 project/gateway_server.py`

6.2 Utilizando o Cliente

Terminal 5 (Cliente):
```
# Conecte-se ao GATEWAY na porta 50050
python3 project/client.py localhost:50050
```

Sessão de Exemplo:
```
bigfs > mkdir bfs://documentos
bigfs > cp meu_arquivo.txt bfs://documentos/remoto.txt
bigfs > ls bfs://documentos/
bigfs > get bfs://documentos/remoto.txt copia_local.txt
bigfs > quit
```