Documentação Técnica do Contrato de API `bigfs.proto`

wender13

1 de julho de 2025

Sumário

1 Introdução

Este documento fornece uma análise técnica detalhada do arquivo `bigfs.proto`, que serve como o contrato de API e a "fonte única da verdade"para toda a comunicação no sistema de arquivos distribuído BigFS. Utilizando Protocol Buffers (Protobuf) e gRPC, este arquivo define de forma agnóstica de linguagem todos os serviços, funções remotas (RPCs) e estruturas de dados (mensagens) que os componentes do sistema usam para interagir.

2 Visão Geral dos Serviços (`service`)

Um serviço em gRPC é uma coleção de funções remotas relacionadas que um componente expõe. A arquitetura de 3 camadas do BigFS é claramente definida pelos três serviços a seguir.

2.1 GatewayService

Esta é a **API pública** do sistema, o único ponto de contato para o `client.py`. Ele abstrai a complexidade do backend.

- **`UploadFile(stream ChunkUploadRequest) returns (SimpleResponse)`**: O RPC para upload de arquivos. Utiliza **streaming do cliente**, onde o cliente envia o arquivo em um fluxo contínuo de pedaços, permitindo a transferência eficiente de arquivos grandes sem sobrecarregar a memória.
- **`GetDownloadMap(FileRequest) returns (FileLocationResponse)`**: RPC para obter o "mapa"de um arquivo. O cliente chama esta função para descobrir onde estão os chunks de um arquivo antes de iniciar o download direto dos Storage Nodes.
- **`ListFiles(PathRequest) returns (FileListResponse)`**: RPC para listar os arquivos. Atua como um proxy para o Metadata Server.

2.2 MetadataService

Esta é a **API de controle interno do backend**, usada exclusivamente pelo Gateway para gerenciar o estado do cluster.

- **`RegisterNode(NodeInfo) returns (SimpleResponse)`**: O RPC de **heartbeat**. É como um Storage Node informa ao Mestre que está vivo e qual sua carga atual.
- **`GetFileLocation(FileRequest) returns (FileLocationResponse)`**: A implementação real da lógica de busca de metadados de um arquivo.
- **`GetWritePlan(FileRequest) returns (FileLocationResponse)`**: O RPC central para a lógica de negócio. É aqui que o Mestre decide como um arquivo será particionado e em quais nós será replicado, aplicando a estratégia de balanceamento de carga.

- **`ListFiles(PathRequest) returns (FileListResponse)`**: A implementação real da listagem de arquivos.

2.3 StorageService

Esta é a **API de dados do backend**, implementada por cada `Storage Node`. É a interface mais simples, focada no armazenamento bruto.

- **`StoreChunk(Chunk) returns (SimpleResponse)`**: Recebe um chunk de dados e o salva no disco local. É chamado pelo Gateway (na escrita inicial) e por outros Storage Nodes (na replicação).
- **`RetrieveChunk(ChunkRequest) returns (Chunk)`**: Lê um chunk do disco local e o retorna. É chamado pelo Cliente durante o download.

3 Análise Detalhada das Mensagens (`message`)

As mensagens definem a estrutura dos dados trocados entre os serviços. Os números (‘= 1‘, ‘= 2‘, etc.) são tags de campo para a serialização binária e não devem ser alterados.

3.1 Estruturas Chave

3.1.1 NodeInfo

```
message NodeInfo {
    string address = 1;
    int32 chunk_count = 2;
}
```

A mensagem do heartbeat. O campo `chunk_count` é essencial para a estratégia de alocação inteligente, permitindo que o Metadata Server saiba qual nó está menos ocupado.

3.1.2 ChunkUploadRequest

```
message ChunkUploadRequest {
    oneof content {
        FileMetadata metadata = 1;
        bytes data = 2;
    }
}
```

Mensagem usada no stream de upload do Cliente para o Gateway. O `oneof` garante que cada pacote do stream contenha ou os metadados do arquivo (a primeira mensagem) ou os dados brutos (as mensagens subsequentes).

3.1.3 FileLocationResponse e ChunkLocation

```
message FileLocationResponse {
    bool is_sharded = 1;
    repeated ChunkLocation locations = 2;
}

message ChunkLocation {
    int32 chunk_index = 1;
    string chunk_id = 2;
    string primary_node_id = 3;
    repeated string replica_node_ids = 4;
}
```

Juntas, formam o "mapa"de um arquivo. `FileLocationResponse` é o contêiner, e a lista (‘repeated‘) de `ChunkLocation` descreve cada chunk:

- **`chunk_index`**: Garante a ordem correta para a reconstrução do arquivo.
- **`primary_node_id`**: O nó principal para este chunk.
- **`replica_node_ids`**: A lista de nós de backup, crucial para a tolerância a falhas.

3.2 Outras Mensagens

- **`FileRequest`**: Um pedido genérico para uma operação em um arquivo, contendo seu nome e, opcionalmente, seu tamanho.
- **`Chunk`**: A representação de um pedaço de arquivo no backend. Crucialmente, contém o campo `replica_node_ids` para que o nó primário saiba para quem deve replicar os dados.
- **`SimpleResponse`**: Uma resposta genérica de sucesso/falha para operações que não retornam dados complexos.

A Código-Fonte Completo do `bigfs.proto`

```
syntax = "proto3";

package bigfs;

service GatewayService {
    rpc UploadFile(stream ChunkUploadRequest) returns (SimpleResponse);
    rpc GetDownloadMap(FileRequest) returns (FileLocationResponse);
    rpc ListFiles(PathRequest) returns (FileListResponse);
}

service MetadataService {
```

```
    rpc RegisterNode(NodeInfo) returns (SimpleResponse) {}
    rpc GetFileLocation(FileRequest) returns (FileLocationResponse) {}
    rpc GetWritePlan(FileRequest) returns (FileLocationResponse) {}
    rpc ListFiles(PathRequest) returns (FileListResponse) {}
}

service StorageService {
    rpc StoreChunk(Chunk) returns (SimpleResponse) {}
    rpc RetrieveChunk(ChunkRequest) returns (Chunk) {}
}

message NodeInfo {
    string address = 1;
    int32 chunk_count = 2;
}

message ChunkUploadRequest {
    oneof content {
        FileMetadata metadata = 1;
        bytes data = 2;
    }
}

message FileMetadata {
    string remote_path = 1;
}

message PathRequest {
    string path = 1;
}

message FileRequest {
    string filename = 1;
    int64 size = 2;
}

message FileListResponse {
    message FileInfo {
        string filename = 1;
        int64 size = 2;
    }
    repeated FileInfo files = 1;
}

message FileLocationResponse {
    bool is_sharded = 1;
    repeated ChunkLocation locations = 2;
}

```

```
message ChunkLocation {
    int32 chunk_index = 1;
    string chunk_id = 2;
    string primary_node_id = 3;
    repeated string replica_node_ids = 4;
}

message ChunkRequest {
    string chunk_id = 1;
}

message Chunk {
    string chunk_id = 1;
    bytes data = 2;
    repeated string replica_node_ids = 3;
}

message SimpleResponse {
    bool success = 1;
    string message = 2;
}
```

Listing 1: Contrato gRPC Completo