Documentação Detalhada do Sistema de Arquivos BigFS-v2

Esta documentação oferece uma visão detalhada do sistema de arquivos BigFS-v2, abordando seus requisitos funcionais e não-funcionais, a arquitetura e seus componentes principais, e o fluxo de operações para cada funcionalidade.

Requisitos Funcionais do BigFS-v2

O BigFS-v2 oferece as seguintes funcionalidades principais para o gerenciamento de arquivos:

- **Listagem de Conteúdo (Listar):**
  - **Descrição:** Permite que o cliente visualize o conteúdo de um diretório específico no sistema de arquivos remoto, incluindo arquivos e subdiretórios.
  - **Fluxo de Operação:**
    1. O cliente envia uma requisição `Listar` com o `path` do diretório desejado para o servidor gRPC.
    2. O `server.py` recebe a requisição e chama a função `listar_conteudo` no `file_manager.py`, passando o `BASE_DIR` e o `path` da requisição.
    3. O `file_manager.py` normaliza o `path`, verifica se ele está dentro do `BASE_DIR` para segurança e então verifica se o `path` existe e se é um arquivo ou diretório.
    4. Se for um arquivo, retorna o nome do arquivo. Se for um diretório, utiliza `os.listdir()` para obter a lista de itens.
    5. O `file_manager.py` retorna o status (sucesso/falha), mensagem, tipo (arquivo/diretório) e o conteúdo (lista de itens) para o `server.py`.
    6. O `server.py` empacota a resposta em um `ConteudoResponse` e envia de volta ao cliente.

- **Deleção de Arquivos (Deletar):**
  - **Descrição:** Permite a remoção de um arquivo específico do sistema de arquivos remoto.
  - **Fluxo de Operação:**
    1. O cliente envia uma requisição `Deletar` com o `path` do arquivo a ser removido para o servidor gRPC.
    2. O `server.py` recebe a requisição e chama a função `deletar_arquivo` no `file_manager.py`, passando o `BASE_DIR` e o `path`.
    3. O `file_manager.py` normaliza o `path`, verifica se o arquivo existe e se não é um diretório (diretórios não podem ser removidos por esta função).
    4. Utiliza `os.remove()` para deletar o arquivo.
    5. O `file_manager.py` retorna o status (sucesso/falha) e uma mensagem para o `server.py`.
    6. O `server.py` empacota a resposta em um `OperacaoResponse` e envia de volta ao cliente.
- **Upload de Arquivos (Upload):**
  - **Descrição:** Permite que o cliente envie dados para criar ou sobrescrever um arquivo no sistema de arquivos remoto.
  - **Fluxo de Operação:**
    1. O cliente envia uma requisição `Upload` com o `path` desejado e os `dados` do arquivo para o servidor gRPC.
    2. O `server.py` recebe a requisição e chama a função `salvar_arquivo` no `file_manager.py`, passando o `BASE_DIR`, o `path` e os `dados`.
    3. O `file_manager.py` normaliza o `path`, garante que os diretórios intermediários existam usando `os.makedirs(exist_ok=True)`.

4. Escreve os `dados` no `path` especificado em modo binário (`wb`).
5. O `file_manager.py` retorna o status (sucesso/falha) e uma mensagem para o `server.py` .
6. O `server.py` empacota a resposta em um `OperacaoResponse` e envia de volta ao cliente.

- **Download de Arquivos (Download):**
  - **Descrição:** Permite que o cliente baixe o conteúdo de um arquivo específico do sistema de arquivos remoto.
  - **Fluxo de Operação:**
    1. O cliente envia uma requisição `Download` com o `path` do arquivo a ser baixado para o servidor gRPC.
    2. O `server.py` recebe a requisição e chama a função `ler_arquivo` no `file_manager.py` , passando o `BASE_DIR` e o `path` .
    3. O `file_manager.py` normaliza o `path` , verifica se o `path` existe e se é um arquivo.
    4. Lê o conteúdo do arquivo em modo binário (`rb`).
    5. O `file_manager.py` retorna o status (sucesso/falha), mensagem e os `dados` do arquivo para o `server.py` .
    6. O `server.py` empacota a resposta em um `FileDownloadResponse` e envia de volta ao cliente.
- **Cópia Interna de Arquivos (CopiarInterno):**
  - **Descrição:** Permite a cópia de um arquivo de um local de origem para um local de destino dentro do sistema de arquivos remoto.
  - **Fluxo de Operação:**
    1. O cliente envia uma requisição `CopiarInterno` com o `origem` e `destino` dos arquivos para o servidor gRPC.

2. O `server.py` recebe a requisição e chama a função `copiar_arquivo` no `file_manager.py`, passando o `BASE_DIR`, `origem` e `destino`.
3. O `file_manager.py` normaliza os caminhos, verifica se ambos estão dentro do `BASE_DIR` e se o arquivo de `origem` existe e é um arquivo.
4. Garante que o diretório de destino exista usando `os.makedirs(exist_ok=True)`.
5. Utiliza `shutil.copy2()` para copiar o arquivo, preservando metadados.
6. O `file_manager.py` retorna o status (sucesso/falha) e uma mensagem para o `server.py`.
7. O `server.py` empacota a resposta em um `OperacaoResponse` e envia de volta ao cliente.

- **Tratamento de Erros:**
  - **Descrição:** O sistema é projetado para capturar e retornar mensagens de erro claras e informativas em caso de falhas nas operações de arquivo.
  - **Fluxo de Operação:** Todas as funções no `file_manager.py` e os métodos no `server.py` utilizam blocos `try-except` para capturar exceções. Em caso de erro, uma mensagem descritiva é gerada e retornada ao cliente, indicando a natureza do problema (ex: "Caminho não encontrado", "Acesso negado", "Erro ao remover").
- **Segurança de Caminho:**
  - **Descrição:** Garante que todas as operações de arquivo sejam restritas a um diretório base pré-definido no servidor, prevenindo acesso ou manipulação de arquivos fora da área permitida.
  - **Fluxo de Operação:** Antes de qualquer operação de arquivo, o `file_manager.py` verifica se o caminho absoluto resultante da requisição (`caminho_absoluto`) começa com o caminho absoluto do diretório base (`os.path.abspath(caminho_base)`). Se não, a operação é negada com uma mensagem de "Acesso negado: fora da área exportada".
- **Suporte a Grandes Arquivos:**

- **Descrição:** O servidor gRPC é configurado para lidar eficientemente com a transferência de arquivos de grande porte.
- **Fluxo de Operação:** No `server.py`, as opções `grpc.max_send_message_length` e `grpc.max_receive_message_length` são definidas para `1024 * 1024 * 1024` bytes (1 GB). Isso permite que o gRPC gerencie a transmissão de dados em chunks para arquivos que excedam os limites de mensagem padrão, otimizando o uso de memória e a performance para grandes transferências.

Requisitos Não-Funcionais do BigFS-v2

Os requisitos não-funcionais descrevem como o sistema deve se comportar e as restrições sob as quais ele deve operar:

- **Performance:**
  - **Concorrência:** O servidor gRPC é configurado com um `ThreadPoolExecutor` que permite o processamento de até 10 requisições simultâneas, garantindo que o sistema possa lidar com múltiplos clientes e operações concorrentes sem degradação significativa de desempenho.
  - **Escalabilidade:** A arquitetura baseada em gRPC facilita a escalabilidade horizontal. Novos servidores podem ser adicionados para distribuir a carga de trabalho, permitindo que o sistema cresça para atender a um número crescente de usuários e volume de dados.
  - **Suporte a Grandes Mensagens:** A configuração explícita dos limites de mensagem (`grpc.max_send_message_length` e `grpc.max_receive_message_length` para 1 GB) garante que o sistema possa transferir arquivos de grande porte de forma eficiente, otimizando o uso de recursos de rede e memória.
- **Segurança:**
  - **Validação de Caminho:** Uma funcionalidade crítica de segurança é a validação de caminho implementada no `file_manager.py`. Esta validação impede ataques de travessia de diretório, assegurando que todas as operações de arquivo ocorram

estritamente dentro do diretório base (`BASE_DIR`) exportado pelo servidor, protegendo o sistema de arquivos subjacente de acessos não autorizados.

    - **Tratamento de Exceções:** A robusta implementação de blocos `try-except` em todas as operações de arquivo e métodos do servidor gRPC garante que o sistema possa se recuperar graciosamente de erros inesperados, como problemas de I/O, permissões ou caminhos inválidos, evitando falhas completas do servidor e fornecendo feedback adequado ao cliente.

- **Disponibilidade:**
    - **Servidor Contínuo:** O servidor gRPC é projetado para operar continuamente, permanecendo ativo e respondendo a requisições até que seja explicitamente encerrado. Isso assegura alta disponibilidade do serviço de arquivos, minimizando o tempo de inatividade.

- **Usabilidade:**
    - **Mensagens de Erro Claras:** O sistema retorna mensagens de erro detalhadas e informativas para o cliente. Isso facilita a depuração para desenvolvedores e ajuda os usuários a entenderem a causa de uma falha na operação, melhorando a experiência geral de uso.

- **Manutenibilidade:**
    - **Modularidade:** A arquitetura do BigFS-v2 é altamente modular, com uma clara separação entre a lógica de comunicação do servidor (`server.py`) e as operações de gerenciamento de arquivos (`file_manager.py`). Isso simplifica a manutenção, o desenvolvimento de novas funcionalidades e a correção de bugs, pois as alterações em uma camada não afetam diretamente a outra.
    - **Uso de gRPC:** A adoção do gRPC como protocolo de comunicação padroniza a interface entre cliente e servidor, facilitando a integração com outros sistemas e a manutenção do código, além de permitir a geração automática de código para diversas linguagens.

- **Portabilidade:**

- **Tecnologias Padrão:** Implementado em Python e utilizando gRPC, o sistema se beneficia da portabilidade dessas tecnologias. Isso significa que o BigFS-v2 pode ser executado em uma ampla gama de sistemas operacionais e ambientes que suportam Python e gRPC, sem a necessidade de grandes modificações.

- **Robustez:**

  - **Tratamento Abrangente de Exceções:** A estratégia de tratamento de exceções em todo o sistema, desde as operações de baixo nível no `file_manager.py` até os métodos do servidor gRPC, garante que o sistema seja resiliente a falhas. Isso previne que erros individuais causem a interrupção completa do serviço, contribuindo para a estabilidade e confiabilidade do BigFS-v2.

Arquitetura e Componentes Principais do BigFS-v2

O BigFS-v2 é projetado com uma arquitetura cliente-servidor, utilizando gRPC para uma comunicação eficiente e de alto desempenho. A modularidade é um princípio fundamental, separando a lógica de negócios do sistema de arquivos da camada de comunicação do servidor.

Componentes Principais:

1. **Servidor gRPC (`server.py`):**

   - **Função:** Atua como o ponto central de comunicação, recebendo e processando todas as requisições dos clientes.

   - **Detalhes:** Implementa o serviço `FileSystemServiceServicer` definido nos arquivos `.proto`. Utiliza um `ThreadPoolExecutor` para gerenciar um pool de threads, permitindo o processamento concorrente de requisições. A configuração para suportar mensagens de até 1 GB é crucial para a funcionalidade de transferência de arquivos grandes. O servidor é iniciado na porta 50051 e permanece ativo para servir os clientes.

2. **Gerenciador de Arquivos (`file_manager.py`):**

- **Função:** Contém toda a lógica de negócios para as operações de arquivo, como listar, deletar, salvar, ler e copiar.
- **Detalhes:** Este módulo é o coração do sistema de arquivos. Ele realiza validações de segurança para garantir que as operações de arquivo estejam confinadas ao `BASE_DIR`, prevenindo acessos não autorizados. Lida com a criação de diretórios intermediários, o que simplifica as operações de upload para o cliente. As funções neste módulo são chamadas pelo servidor gRPC para executar as ações solicitadas.

3. **Definições de Protocolo (`.proto` files):**
   - **Função:** Define a interface de serviço (`FileSystemService`) e as estruturas de mensagens (`ConteudoResponse`, `OperacaoResponse`, etc.) que formam o contrato de comunicação entre o cliente e o servidor.
   - **Detalhes:** Utilizando a sintaxe do Protocol Buffers, esses arquivos são compilados para gerar o código Python (`_pb2.py` e `_pb2_grpc.py`) que é usado para serializar e desserializar os dados, garantindo uma comunicação eficiente e tipada. A definição explícita dos serviços e mensagens torna a API do sistema clara e fácil de entender.
4. **Diretório de Armazenamento (`storage`):**
   - **Função:** É o diretório no sistema de arquivos do servidor onde os dados são fisicamente armazenados.
   - **Detalhes:** O `BASE_DIR` (definido em `server.py`) aponta para este diretório. Todas as operações de leitura, escrita, deleção e cópia são restritas a este local, garantindo a segurança e o isolamento dos dados gerenciados pelo BigFS-v2.
5. **Cliente (Ex: `client/intelligent_client.py`):**
   - **Função:** Representa a aplicação do lado do usuário que interage com o sistema de arquivos remoto.
   - **Detalhes:** O cliente utiliza o código gerado a partir dos arquivos `.proto` para criar um *stub* que permite invocar os métodos do serviço `FileSystemService` como se fossem chamadas de função locais. O cliente é responsável por iniciar as operações de arquivo, como solicitar uma listagem de diretório ou enviar um arquivo para upload.

Benefícios da Arquitetura BigFS-v2

O BigFS-v2 apresenta diversos benefícios decorrentes de sua arquitetura e das tecnologias empregadas:

1. **Modularidade e Separação de Responsabilidades:** A distinção clara entre a lógica do servidor gRPC (`server.py`) e o gerenciamento de arquivos (`file_manager.py`) resulta em um código mais organizado, fácil de entender, manter e testar. Essa separação permite que a lógica de negócios do sistema de arquivos seja desenvolvida e testada independentemente da camada de comunicação, facilitando a colaboração e a evolução do projeto.

2. **Performance e Escalabilidade com gRPC:**

   - **Comunicação Eficiente:** A utilização de gRPC, que se baseia em HTTP/2 para transporte e Protocol Buffers para serialização, proporciona uma comunicação de alta performance e baixa latência. Isso é particularmente vantajoso para a transferência de grandes volumes de dados ou mensagens complexas, como é o caso de operações de arquivo.

   - **Suporte a Streaming:** gRPC oferece suporte nativo a streaming bidirecional, o que é ideal para operações de upload e download de arquivos grandes. Essa capacidade permite a transferência de dados em *chunks*, otimizando o uso de recursos e melhorando a experiência do usuário para arquivos de grande porte.

   - **Concorrência Eficiência:** O uso de um `ThreadPoolExecutor` no servidor gRPC permite que ele processe múltiplas requisições de clientes simultaneamente. Isso aumenta a capacidade de resposta do sistema e seu *throughput*, garantindo que um grande número de operações possa ser tratado de forma eficiente.

   - **Escalabilidade Horizontal Simplificada:** A natureza distribuída do gRPC facilita a escalabilidade horizontal do sistema. É possível adicionar novos servidores para distribuir a carga de trabalho, permitindo que o BigFS-v2 expanda sua capacidade para atender a um número crescente de usuários e volume de dados sem a necessidade de reengenharia complexa.

3. **Segurança Básica Implementada:** Uma medida de segurança fundamental implementada no `file_manager.py` é a validação de caminho. Essa validação impede ataques de travessia de diretório (`path traversal`), garantindo que todas as operações de arquivo estejam restritas ao diretório base (`BASE_DIR`) exportado pelo servidor. Isso protege o sistema de arquivos subjacente contra acessos e manipulações não autorizadas.

4. **Suporte Robusto a Grandes Arquivos:** A configuração explícita de `grpc.max_send_message_length` e `grpc.max_receive_message_length` para 1 GB no servidor gRPC demonstra um design intencional para lidar com arquivos de grande porte. Essa capacidade é essencial para um sistema de arquivos moderno, que frequentemente precisa gerenciar mídias, backups e outros arquivos volumosos.

5. **Interoperabilidade:** gRPC é uma tecnologia agnóstica à linguagem. Isso significa que clientes desenvolvidos em diversas linguagens de programação (como Java, Go, Node.js, C#, etc.) podem interagir facilmente com o servidor BigFS-v2, desde que possuam as definições do Protocol Buffer. Essa interoperabilidade facilita a integração do BigFS-v2 em ecossistemas de software heterogêneos.

6. **Robustez Através do Tratamento de Exceções:** A implementação abrangente de blocos `try-except` em todas as operações de arquivo e métodos do servidor gRPC confere ao sistema uma alta robustez. Isso garante que o BigFS-v2 possa se recuperar graciosamente de erros inesperados (como problemas de I/O, permissões ou caminhos inválidos), evitando falhas completas do servidor e mantendo a disponibilidade do serviço.