Documentação Técnica Detalhada
`storage_node.py`

wender13

1 de julho de 2025

Sumário

1 Introdução e Papel na Arquitetura

O arquivo `storage_node.py` implementa o "músculo"do cluster BigFS. Cada instância deste script representa um servidor de armazenamento independente e robusto, cuja única finalidade é armazenar fisicamente os *chunks* (pedaços de arquivos) e se reportar ao `Metadata Server`.

A simplicidade é uma característica fundamental do seu design. Ele não possui conhecimento global do sistema de arquivos; apenas gerencia os blocos de dados que lhe são confiados e executa as ordens de replicação, contribuindo para a escalabilidade e resiliência do sistema como um todo.

2 Análise da Classe `StorageService`

Esta classe implementa a API do serviço `StorageService`, que define as operações de dados que o nó pode executar.

2.1 `__init__(self, storage_dir)`

O construtor prepara o ambiente de armazenamento do nó.

```
def __init__(self, storage_dir):
    self.storage_dir = storage_dir
    if not os.path.exists(storage_dir):
        os.makedirs(storage_dir)
```

Explicação: Ao ser inicializado, o serviço recebe o caminho para seu diretório de armazenamento (ex: `storage_50052`). Ele armazena esse caminho e garante que o diretório exista no disco, criando-o se for necessário.

2.2 `StoreChunk(self, request, context)`

Este é o método central para a escrita de dados e o disparo da replicação.

```
def StoreChunk(self, request, context):
    chunk_path = os.path.join(self.storage_dir, request.chunk_id)
    try:
        with open(chunk_path, 'wb') as f: f.write(request.data)
        if request.replica_node_ids:
            for replica_addr in request.replica_node_ids:
                threading.Thread(target=self._replicate_chunk,
                                 args=(replica_addr, request),
                                 daemon=True).start()
        return bigfs_pb2.SimpleResponse(success=True)
    except IOError: return bigfs_pb2.SimpleResponse(success=False)
```

Explicação do Fluxo:

1. **Armazenamento Local:** O método primeiro constrói o caminho completo para o arquivo do chunk e salva os dados binários (‘request.data‘) recebidos.

2. **Disparo da Replicação:** Em seguida, ele verifica se a requisição contém uma lista de 'replica$_n ode_i ds$'.$Sesim, signi ficaqueeleestatuandocomoon**Primrio**paraestaoperao.$**Execução**

2.3 `_replicate_chunk(self, replica_address, chunk)`

Esta função privada é o trabalhador da replicação.

```
def _replicate_chunk(self, replica_address, chunk):
    try:
        with grpc.insecure_channel(replica_address) as channel:
            stub = bigfs_pb2_grpc.StorageServiceStub(channel)
            # Envia uma mensagem limpa, sem a lista de réplicas
            replica_chunk = bigfs_pb2.Chunk(
                chunk_id=chunk.chunk_id, data=chunk.data
            )
            stub.StoreChunk(replica_chunk, timeout=15)
    except grpc.RpcError: pass
```

Explicação: Executando em sua própria thread, esta função se conecta a outro `Storage Node` (a réplica) e chama o seu método `StoreChunk`. É importante notar que a mensagem enviada à réplica é uma nova instância de `Chunk` que não contém a lista de 'replica$_n ode_i ds$', $paraevitarquearplicatentereplicarosdadosnovamente, oquecausariaumloopin$

2.4 `RetrieveChunk(self, request, context)`

Responsável por servir os dados quando solicitado.

```
def RetrieveChunk(self, request, context):
    chunk_path = os.path.join(self.storage_dir, request.chunk_id)
    try:
        with open(chunk_path, 'rb') as f: data = f.read()
        return bigfs_pb2.Chunk(chunk_id=request.chunk_id, data=data)
    except FileNotFoundError:
        context.set_code(grpc.StatusCode.NOT_FOUND)
        context.set_details("Chunk não encontrado")
        return bigfs_pb2.Chunk()
```

Explicação: Uma função simples de leitura de arquivo. Ela lê os bytes do chunk solicitado do disco e os retorna ao requisitante (geralmente, o `client.py`). Se o arquivo não for encontrado, ela define um código de erro gRPC 'NOT$_F OUND$', $queoclienteutilizaems$

3 Interação com o Cluster

Estas funções gerenciam a existência e o status do nó dentro do ecossistema BigFS.

3.1 `send_heartbeats(...)`

A função que mantém o nó "vivo"aos olhos do Mestre.

```python
def send_heartbeats(node_id, storage_dir, metadata_address):
    while True:
        try:
            # 1. Conta o número de chunks no disco
            chunk_count = len(os.listdir(storage_dir))
            with grpc.insecure_channel(metadata_address) as channel:
                stub = bigfs_pb2_grpc.MetadataServiceStub(channel)
                # 2. Envia seu status (endereço + contagem)
                node_info = bigfs_pb2.NodeInfo(
                    address=node_id, chunk_count=chunk_count
                )
                stub.RegisterNode(node_info)
        except Exception: pass
        time.sleep(5)
```

Explicação: Rodando em uma thread separada, esta função entra em um loop infinito. A cada 5 segundos, ela conta quantos arquivos (chunks) existem em seu diretório de armazenamento e envia essa contagem, junto com seu próprio endereço, para o `Metadata Server`. Esta informação é a base para a estratégia de **balanceamento de carga** do sistema.

3.2 Inicialização do Servidor (`serve` e `main`)

O código que inicia o processo do Storage Node.

```python
def serve(port, metadata_address, my_ip):
    node_id = f"{my_ip}:{port}"; storage_dir = f"storage_{port}"
    # ...
    server = grpc.server(...)
    bigfs_pb2_grpc.add_StorageServiceServicer_to_server(
        StorageService(storage_dir), server
    )
    server.add_insecure_port(f'[::]:{port}'); server.start()
    # Inicia a thread de heartbeat em paralelo
    threading.Thread(target=send_heartbeats, ...).start()
    server.wait_for_termination()

if __name__ == '__main__':
    # ... parseia os argumentos da linha de comando ...
    serve(...)
```

Explicação: O ponto de entrada principal lê os argumentos da linha de comando (seu IP, sua porta e o endereço do mestre), cria uma instância do servidor gRPC, registra a lógica da classe `StorageService` nele, e, crucialmente, inicia a **thread de `send_heartbeats`** para que o nó comece a se anunciar para o cluster assim que for iniciado.