Arquitetura BigFS

Wender Vieira Marques Júnior

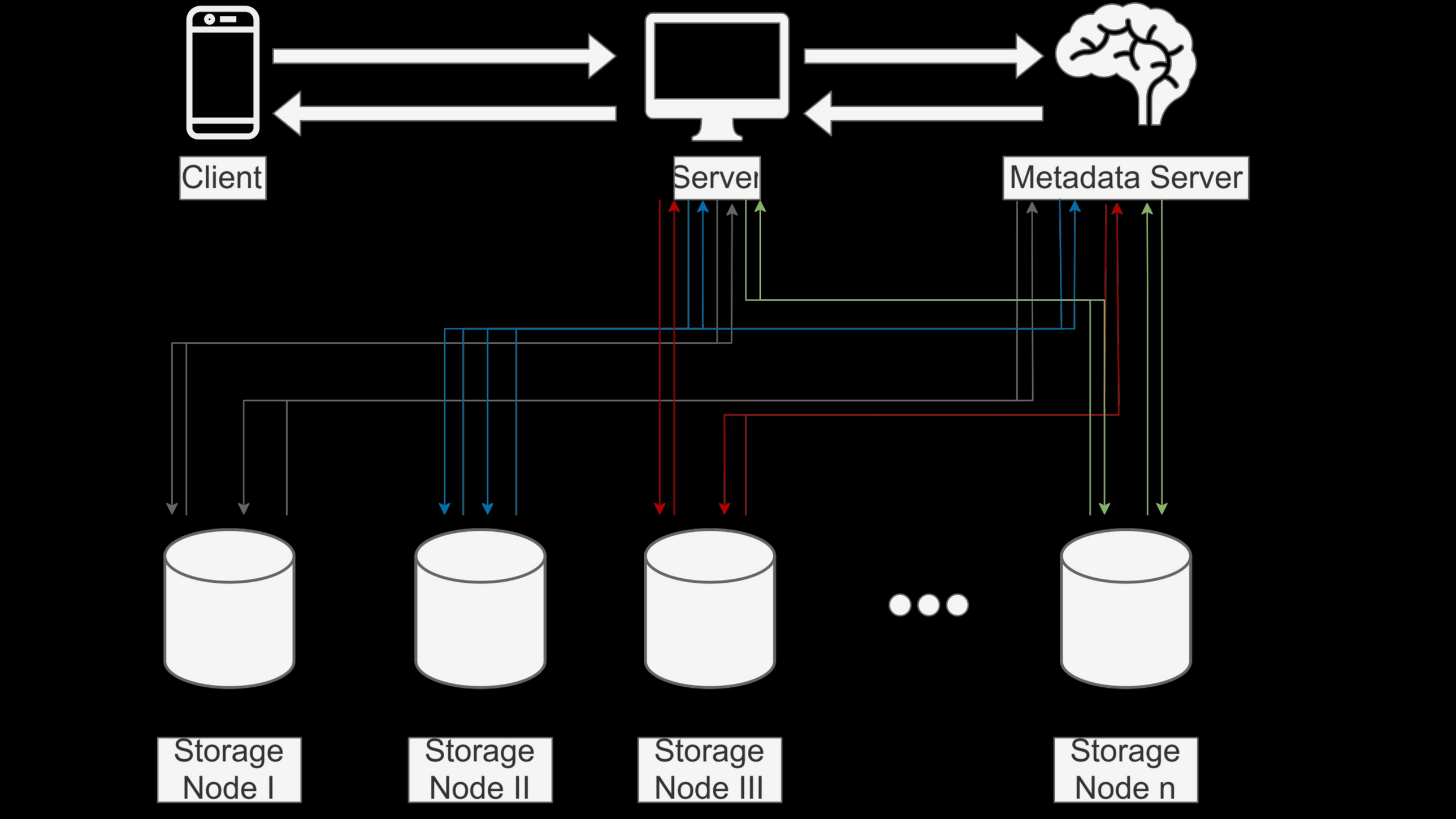
1 - Arquitetura e componentes

2 - Fluxo de Operações

3 - Estratégia de Tolerância a Falhas

4 - Garantia de Integridade na Reconstrução de Arquivos

Arquitetura e componentes


Client
Server
Metadata Server
Storage Node I
Storage Node II
Storage Node III
Storage Node n

Motivação

Escalabilidade Massiva e de Baixo Custo:

- **Problema:** Um único servidor tem um limite físico de armazenamento e seu custo cresce exponencialmente (escalabilidade vertical);
- **Solução:** A arquitetura permite escalabilidade horizontal quase infinita. Adicionar mais capacidade é tão simples e barato quanto adicionar novos Storage Nodes (servidores comuns) ao cluster.

Alto Desempenho para Grandes Volumes:

- **Problema:** A velocidade de leitura/escrita de um único servidor é limitada por seu disco e sua rede, criando um gargalo para arquivos de gigabytes ou terabytes;
- **Solução:** Através do particionamento (sharding) e do processamento paralelo, o BigFS lê e escreve um único arquivo a partir de múltiplos nós simultaneamente, multiplicando a velocidade de transferência de dados.

Alto Desempenho para Grandes Volumes:

- **Problema:** Um sistema centralizado é um ponto único de falha. Se o servidor falha, os dados são perdidos e o serviço fica indisponível.
- **Solução:** A replicação automática de cada chunk de dados em múltiplos nós garante que o sistema seja projetado para falhas. A queda de um ou mais nós não resulta em perda de dados nem interrompe a disponibilidade do sistema.

Inspiração

Google File System (GFS):

- O sistema pioneiro criado pelo Google para gerenciar a quantidade colossal de dados gerada por seus serviços. O artigo original do GFS definiu o padrão Mestre/Escravo para este tipo de aplicação.

Hadoop Distributed File System (HDFS):

- O BigFS implementa a robusta arquitetura de backend Mestre/Escravo do HDFS, usando um Metadata Server central para gerenciar Storage Nodes que armazenam os dados replicados, garantindo escalabilidade e tolerância a falhas.

Minio (Inspiração da Interface):

- A interação com o BigFS é inspirada na usabilidade do Minio Client (mc), com uma interface de linha de comando clara e baseada em ações (ls, cp, get), focada na experiência do desenvolvedor.

Composição

Cliente:

- **Ponto de Acesso Único:** Atua como a interface de usuário, comunicando-se exclusivamente com o Gateway Server. Ele não tem nenhum conhecimento sobre a topologia do backend (Metadata ou Storage Nodes);

- **Simplicidade nas Operações:** Sua lógica foi simplificada:

- **Upload (cp):** Envia o arquivo inteiro em um fluxo contínuo para o Gateway, sem se preocupar com particionamento (sharding) ou replicação;
- **Download (get):** Pede o "mapa" do arquivo ao Gateway e, com o mapa em mãos, baixa os chunks em paralelo diretamente dos Storage Nodes;
- **Inteligência de Failover:** A lógica para lidar com falhas durante o download (tentar uma réplica se um Storage Node cair ou se um chunk estiver corrompido) permanece no cliente, garantindo a resiliência da leitura.

Composição

Gateway Server:

- **Ponto de Ingestão Centralizado:** Recebe os arquivos completos enviados pelo cliente, atuando como o único ponto de entrada para escritas no sistema;
- **Motor de Particionamento (Sharding):** Sua função mais crítica. Após receber um arquivo, ele consulta o Metadata Server para obter um plano de escrita e então executa a separação do arquivo em múltiplos chunks;
- **Distribuidor de Carga Inteligente:** É responsável por enviar cada chunk para o Storage Node primário correto, seguindo o plano de alocação (que prioriza os nós mais vazios) fornecido pelo Metadata Server;
- **Proxy de Metadados:** Para operações como ls e rm, ele simplesmente atua como um intermediário, repassando a requisição do cliente para o Metadata Server e devolvendo a resposta.

Composição

Metadata Server:

- **Cérebro Central:** Atua como o Mestre do cluster e a fonte única da verdade para a estrutura do sistema;
- **Gerenciador de Metadados:** Mantém em memória o mapa completo que associa cada nome de arquivo à localização de todos os seus chunks;
- **Arquiteto de Dados:** Elabora planos de escrita para novos arquivos, decidindo como particioná-los e em quais nós a cópia primária e as réplicas serão salvas;
- **Monitor do Cluster:** Recebe e processa os Heartbeats de todos os Storage Nodes para saber quem está ativo;
- **Detector de Falhas:** Identifica quando um Storage Node falha ao notar a ausência de seus heartbeats;

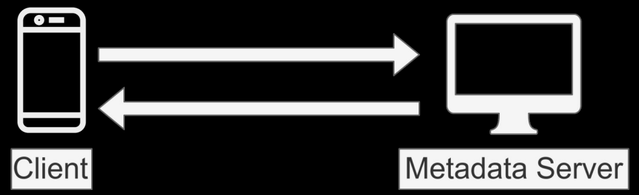
Composição

Storage Node:

- **Músculo do Cluster:** Atua como o "disco rígido" distribuído do sistema, sendo o componente que fisicamente armazena os dados;
- **Foco em Armazenamento:** Sua única responsabilidade é armazenar e servir chunks de dados brutos quando solicitado;
- **Reporte de Status:** Envia Heartbeats periódicos ao Metadata Server para comunicar que está online e saudável, permitindo a detecção de falhas;
- **Comunicação Direta:** Responde diretamente às requisições de leitura e escrita do Cliente, formando o "Plano de Dados" do sistema;
- **Conhecimento Local:** Opera com conhecimento zero sobre o arquivo completo, sua estrutura ou a existência de outros nós. Ele apenas gerencia os chunks que lhe foram designados;
- **Duplo Papel na Replicação:** Funciona tanto como nó Primário (recebendo dados originais do cliente) quanto como Réplica (recebendo cópias de outro Storage Node para redundância).

Fluxo de Operações

Gateway Server x Metadata Server



Propósito: Servir como o ponto de entrada único para todas as operações do usuário, tanto para transferência de dados (upload) quanto para comandos de gerenciamento, abstraindo completamente a complexidade do cluster de backend.

Fluxo de Comunicação:

Cliente → Gateway Server: O cliente dá ordens de alto nível.

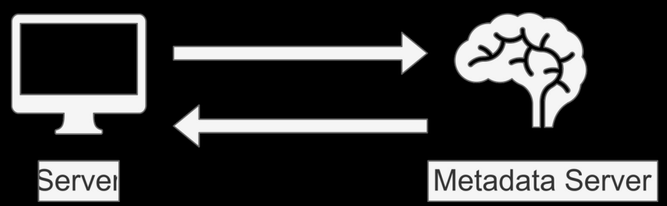
UploadFile: "Por favor, armazene este arquivo completo no sistema." (Enviado como um fluxo contínuo de dados).
GetDownloadMap: "Preciso do 'mapa' para baixar este arquivo."
ListFiles: "Liste para mim o conteúdo do sistema."
DeleteFile: "Apague este arquivo do sistema."

Gateway Server → Cliente: O Gateway retorna os resultados finais.

Retorna uma resposta simples de sucesso ou falha para operações como UploadFile e DeleteFile.
Retorna o mapa de chunks completo (FileLocationResponse) para que o cliente possa realizar o download.
Retorna a lista de arquivos (FileListResponse).

Plano Arquitetural: Esta comunicação opera em ambos os planos. É Plano de Dados durante o UploadFile e Plano de Controle para todas as outras requisições de metadados e comandos.

Gateway Server x Metadata Server



- **Natureza:** Server para Gerente. O server é o requisitante, e o metadata server é a autoridade que fornece inteligência e permissão;
- **Propósito:** Unicamente para troca de metadados e planejamento. Nenhum dado de arquivo trafega neste canal;

Fluxo de Comunicação:

Server → Metadata Server: O server perguntas.
GetWritePlan: "Como e onde eu salvo este novo arquivo?"
GetFileLocation: "Onde estão os pedaços deste arquivo existente?"
ListFiles: "Quais arquivos existem no sistema?"

Metadata Server → Cliente: O servidor dá respostas e "mapas".
Retorna a localização do nó Primário e das Réplicas para cada chunk.
Retorna a lista ordenada de chunks e seus endereços para leitura.

- **Plano Arquitetural:** Esta comunicação pertence exclusivamente ao Plano de Controle.

Gateway Server x Storage Nodes

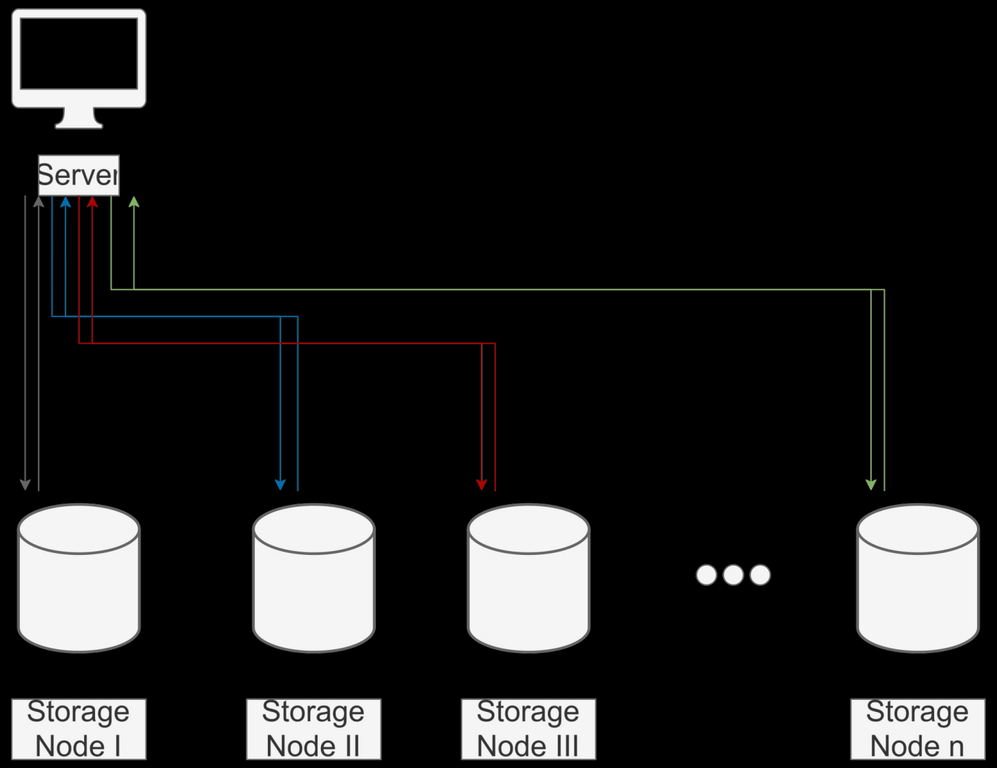
- **Natureza:** Server para Depósito. O server tem as instruções e comanda os trabalhadores do depósito (Storage Nodes) a moverem caixas (chunks);
- **Propósito:** Transferência de dados brutos em alta velocidade. É aqui que o trabalho pesado acontece.

Fluxo de Comunicação:

Server → Storage Node: O cliente dá ordens diretas.
StoreChunk: "Armazene este bloco de dados."
RetrieveChunk: "Me entregue este bloco de dados."

Storage Node → Server: O nó executa e responde.
Retorna uma confirmação simples de sucesso.
Retorna o bloco de dados solicitado.

- **Plano Arquitetural:** Esta comunicação pertence exclusivamente ao Plano de Dados.



Metadata Server x Storage Nodes

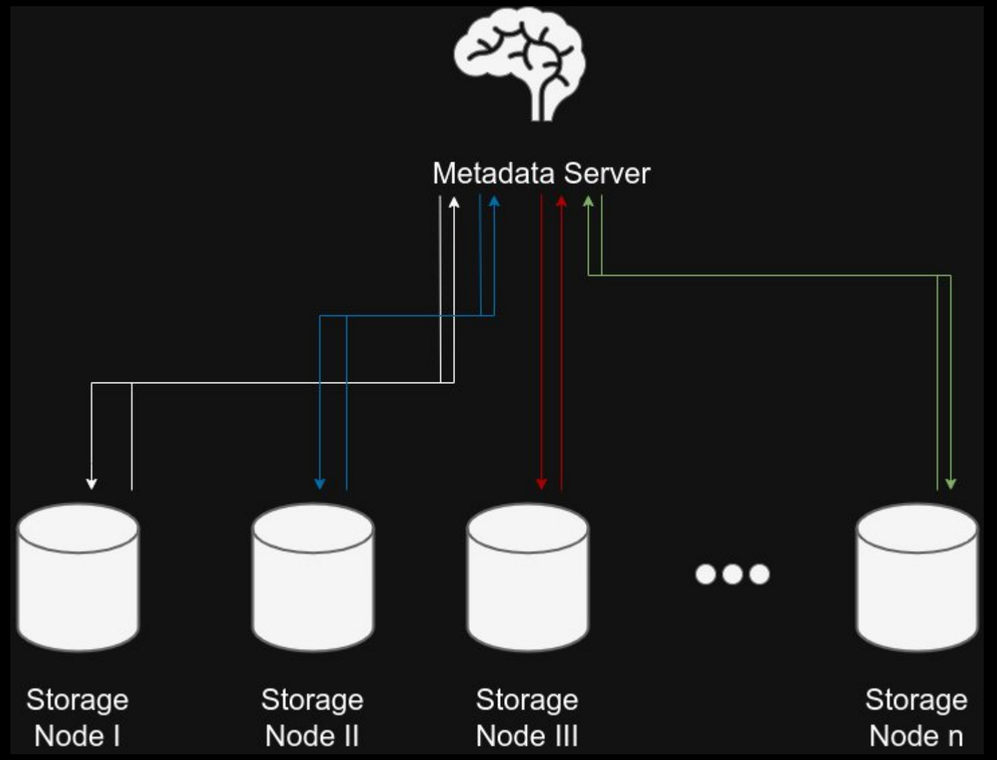
- **Natureza:** Gerente para Funcionário. Focada em status, saúde e comandos administrativos;
- **Propósito:** Gerenciamento do estado e da saúde do cluster.

Fluxo de Comunicação:

Storage Node → Metadata Server: O funcionário reporta seu status.
RegisterNode (Heartbeat): "Continuo vivo e operando neste endereço." Este é um sinal periódico e proativo.

Metadata Server → Storage Node: O gerente monitora e comanda.
Monitoramento (Passivo): A principal ação do servidor é interna. Ele atualiza sua lista de nós ativos com base nos heartbeats recebidos. Se um heartbeat para, o nó é considerado morto.
Comandos (Ativo, em implementação completa): O servidor comandaria um nó para executar ações de recuperação, como "Você (novo primário), crie uma cópia deste chunk em outro nó".

- **Plano Arquitetural:** Esta comunicação pertence ao Plano de Gerenciamento do Cluster.



Estratégia de Tolerância a Falhas

Detecção: Protocolo Heartbeat ❤

- **Sinal de Vida:** Storage Nodes enviam um "heartbeat" para o Metadata Server a cada 5 segundos.
- **Diagnóstico de Falha:** Se o mestre não recebe um sinal após um timeout (15s), ele considera o nó como **"morto"**.
- **Ação Imediata:** O nó é removido da lista de ativos e os mecanismos de recuperação são acionados.

Durabilidade e Disponibilidade via Replicação

Estratégia Fundamental: Redundância

- O princípio central para tolerar falhas é a redundância de dados. Cada chunk é replicado N vezes pelo cluster.

Mecanismo de Implementação: Orquestração Mestre/Escravo

- O sistema utiliza um fluxo orquestrado para garantir a replicação e a descoberta das cópias:
- Planejamento (feito pelo Metadata Server): Atua como o cérebro, definindo um plano de replicação para cada chunk (1 Primário + N-1 Réplicas) com base nos nós ativos.
- Execução (feita pelo Nó Primário): Após receber os dados do cliente, o nó primário executa a replicação de forma assíncrona para as réplicas designadas.
- Descoberta (garantida pelo Metadata Server): Ele centraliza o "mapa" autoritativo de todas as cópias. O cliente confia neste mapa para encontrar a localização primária e todas as alternativas (réplicas) durante a leitura ou em caso de falha.

Garantias Oferecidas:

- **Durabilidade:** Assegura que a perda de N-1 nós contendo o mesmo dado não resulte em perda de informação.
- **Alta Disponibilidade:** Permite que as operações de leitura continuem funcionando sem interrupção através do failover transparente do cliente para uma réplica saudável.

Recuperação: Failover e Self-Healing 🛡️

- **Failover (Lado do Cliente): Se um cliente falha ao ler um chunk de um nó, ele automaticamente tenta a leitura a partir de uma réplica. Para o usuário, o sistema continua funcionando sem interrupções.**

- **Self-Healing (Lado do Cluster): Após detectar a morte de um nó, o mestre "cura" o sistema: promove uma réplica para ser a nova primária e comanda a criação de uma nova cópia em outro nó saudável, restaurando o nível de redundância automaticamente.**

Garantia de Integridade na Reconstrução de Arquivos

Garantia da Ordem: Metadados com Índices

- **O Problema: Durante o download paralelo, os "pedaços" (chunks) do arquivo podem chegar fora de ordem.**
- **A Solução: O Metadata Server fornece ao cliente um mapa ordenado de chunks.**
- **O Mecanismo: Cada chunk possui um índice único (0, 1, 2, ...). O cliente usa esses índices para remontar o arquivo na sequência correta, como um quebra-cabeça.**

Garantia da Qualidade: Checksums (Soma de Verificação) ✓

- O Problema: Proteger contra a corrupção silenciosa de dados, onde os bits em disco se alteram sem aviso (bit rot).

- A Solução: Uma "assinatura" matemática (checksum) verifica a integridade de cada chunk.

- Na Escrita:
  Um checksum é calculado para o chunk.
  Essa assinatura é armazenada junto com os dados no Storage Node.

- Na Leitura:
  O Storage Node recalcula o checksum dos dados lidos do disco.
  Ele compara o resultado com o checksum original armazenado.
  Se forem diferentes, o dado está corrompido. O nó retorna um erro.
  O cliente, ao receber o erro, solicita o mesmo chunk de outra réplica saudável.

Fim