Google File System - sistema de arquivos usado pelo Google para implementação de diversas aplicações disponibilizadas no seu portfólio de sistemas de arquivos.

GFS

- Milhares de PCs em um cluster.
  Os computadores são reunidos em um cluster. Normalmente computadores de pequeno porte com uma quantidade razoável de memória e processamento, para facilitar substituição em caso de falha.
- Cada PC é montado a partir de componentes comuns e baratos.
- Acesso contínuo a dados por centenas de clientes.
  Característica principal.
- Falhas de componentes são a norma, não a exceção
  O sistema se torna mais tolerante à falha justamente por sua arquitetura suscetível a falhas.
- Tipos de falhas
  - Bugs nas aplicações
  - Bugs no SO
  - Falha humana
  - Falhas de hardware
- Monitoramento, tolerância a falhas e recuperação automática são elementos essenciais!

Padrões de acesso

Características particulares das aplicações onde ele é utilizado

- Maioria dos arquivos são grandes(GB)
  Arquivos de mídia, como vídeo e áudios.
- Rápido crescimento do conjunto de dados.
  O volume de escrita é grande e cresce rapidamente.
- A maioria das modificações são por anexação concorrente.
  Normalmente o que é adicionado são arquivos ao final do arquivo, de maneira concorrente com outros clientes.
- Escritas aleatórias são raras.
  Normalmente a escrita é ao final do arquivo.

- Arquivos são lidos de forma sequencial.
  Dificilmente o acesso é aleatório.
- Evita o uso de cache
  Não há cache para dados, apenas para metadados.
- Foco: Operação de append e garantias de atomicidade com o mínimo de custo de sincronização.
  append: anexar ao final. Garante atomicidade.

Requisitos

- Monitoramento, detecção e recuperação eficiente de falhas.
  As falhas são regra e não exceção.
- Leituras
  - Grandes blocos de dados
  - Em um mesmo cliente geralmente são sequenciais
  - Seeks são raríssimos.
    Dificilmente ocorrem acessos aleatórios.
- Escritas
  - Sequenciais de grandes blocos de dados
    Blocos grandes de escrita e leitura.
  - Geralmente são appends(escritas no final)
- Necessidade de tratar acessos massivamente concorrentes.
  Evita a inconsistência nos arquivos e usa estratégias de atomicidade.
  - Garantir atomicidade com o mínimo de overhead

Requisitos

- Maior largura de banda é mais importante que baixa latência de acesso.
  Deve-se priorizar mais a largura de banda quando se comparado com a redução de latência no acesso aos arquivos.
- Necessidade de utilizar abstrações convencionais (diretórios, path).

- Prover API estilo UNIX(create, delete, open, close)
  As operações do UNIX devem estar presentes no GFS.

Arquitetura do GFS

- Um arquivo é formado por um ou mais chunks
  O GFS divide os arquivos em **chunks**.
- Chunk: blocos de 64MB, identificados por um ID de 64bits

- Milhares de chunkservers
  Armazenados em **chunkservers,** que são aplicações rodando em diversas máquinas distribuídas em datacenters do Google. Dentro de cada máquina física existe um linux que realiza a função do chunk server.
  - Distribuídos em vários racks
  - São processos Linux

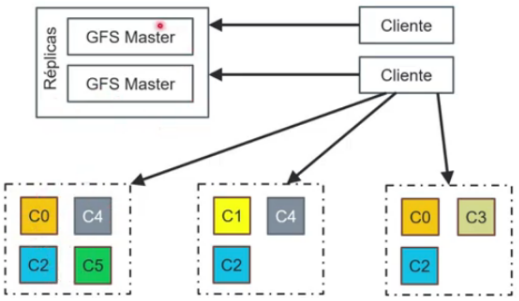
  - Cada chunk é replicado em 3 chunk servers(default)

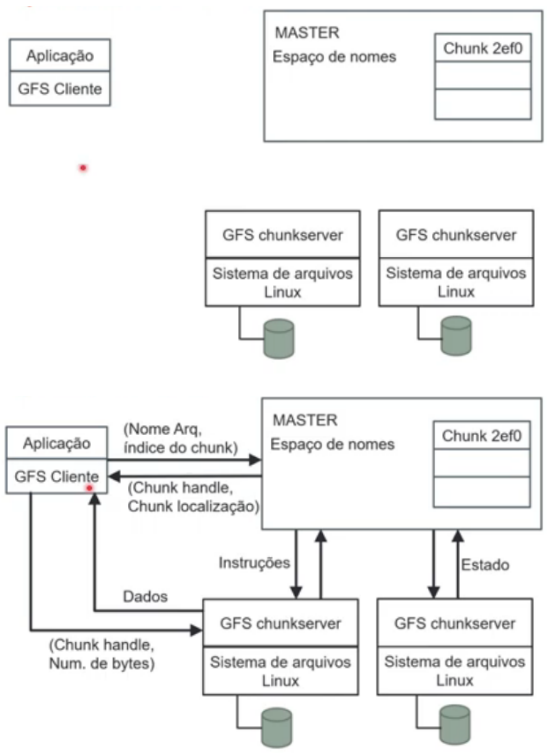
- Master Server (replicado em backups)
  mantém todos os metadados necessários para organização dos chunks e chunkservers.

  A partir dele se localiza quais chunks compõem um arquivo. Associa diretórios e arquivos para seus chunks.
  - Mantém os metadados – mapeamento da árvore de diretórios para os chunks

- Milhares de clientes acessam arquivos hospedados em diferentes nós.

Arquitetura do GFS





Master Server

- Armazena os metadados em memória

  metadados são:

  - Árvore de diretórios
  - Informações de controle de acesso a arquivos

    quem pode acessar que arquivo.
  - Mapeamento: arquivos -> conjunto de chunks
  - Localização de cada chunk nos chunkservers

Master Server garante:

- Acesso rápido em função dos metadados em memória

  em função dos metadados estarem armazenados em memória.
- Coleta de lixo de chunks órfãos

  Quando se apaga um arquivo, os chunks daquele arquivo se tornam órfãos e são apagados.
- Controla a migração de chunks entre chunkservers

  Se um chunk server falhar, o master server realiza a migração de um chunk server para outro.
- Comunica-se periodicamente com os chunkservers (heartbeat) para enviar comandos e receber status

  heartbeat - muito usado em sistemas de tolerância a falhas e monitoramento de sistemas redundantes.

- Master centralizado simplifica o gerenciamento dos chunks
  Um master server faz controle de diversos chunkservers, de maneira centralizada. Os master servers são replicados para garantir tolerância a falhas.

Cliente GFS

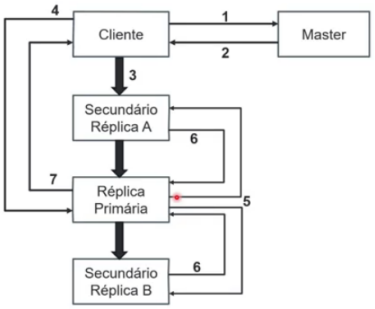
- A API do GFS é ligada à aplicação.
  Garante que a aplicação consiga fazer requisições ao GFS.
- Consulta o Master Server para obter os metadados
  - Ex: Endereço dos chunkservers que hospedam os chunks solicitados pela aplicação.
- Interage com os chunkservers para leituras e escritas nos chunks
  Cliente GFS interage com o chunk server.
- Não faz cache de chunks
  O custo é inviável pois a garantia de coerência se torna cara.
  - O custo de manter a coerência da cache não vale a pena.
- Faz cache dos metadados (tempo limitado).
  A cache é usada para os metadados no lado do cliente para se obter mais rápida localização. O tempo é limitado para evitar incoerência de cache.
  - Ex: localização dos chunks nos chunkservers
- Cliente consulta o master server por vários chunks de uma só vez
  Se junta uma quantidade grande de requisições e manda ao master server para se obter eficiência de uso na banda de rede.

Interações do sistema

- Mutações – Operações que alteram os metadados ou o conteúdo de um chunk. Ex: write e append

- Uma mutação deve ser feita em todas as réplicas.
  Normalmente existe uma réplica primária e uma secundária. A ordem de alteração segue a ordem de réplica.
- Concessões – Garante uma ordem consistente das mutações em todas as réplicas.
  Eleição de quem é a réplica primária.
- Uma das réplicas é eleita primária pelo master.
- O primário determina a ordem das mutações em um chunk nas outras réplicas.
- Uma concessão dura 60 segundos, mas pode ser renovada pelo master sempre que necessário.
  Normalmente limitada.

Interação de mutação



Chunk Server

- Responsável por armazenar os chunks de 64MB
- Utiliza arquivos convencionais do sistema de arquivos Linux
  O processo usa um sistema de arquivos convencional linux para armazenar os arquivos.
- Atende requisições do Master
  Requisições:
  - Quais chunks hospeda
  - Status
- Cria novos chunks
- Remove chunks

Interação Master – Chunkservers

- Utiliza o heartbeat para consultas regulares sobre qual chunk server hospeda uma réplica de um chunk.
  Faz consultas regulares e verifica se os chunkservers estão bem.
- Tipo de consultas aos chunkservers
  Perguntas que o master server faz periodicamente aos chunkservers.

    - Chunkserver está operacional?
    - Houve falhas no disco?
    - Existem réplicas de chunks corrompidas?
    - Quais réplicas de chunk hospeda?

- Comandos enviados aos chunkservers

  O master server envia comandos quando necessário:
    - Remover chunks
    - Criar chunks

- Facilita o gerenciamento do ambiente dinâmico dos chunkservers

  Master Server determina que chunkservers armazenam que chunks.

Escolha das réplicas primária e secundárias

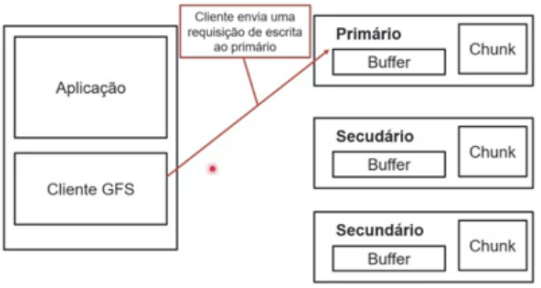
- O master fornece leases (concessões) a uma das réplicas

  Indica dentro de um conjunto de um conjunto de chunkservers.

- O primário define a ordem das operações e todos os secundários adotam esta ordem.
- Um lease expira a cada 60 segundos, podendo ser aumentado.
    - Pode ser revogado em caso de suspeita de falha no primário

---

Exemplo operação de escrita



Record Append

- Operação comum e fundamental para o GFS
- Anexa o conteúdo de um cliente ao final do arquivo.
- O arquivo é tratado com uma fila com múltiplos produtores e um único consumidor.

  O arquivo é alimentado por vários processos produtores que escrevem ao final do

arquivo e um único processo consumidor (quem irá ler).
- O bloco de dados do cliente é anexado de maneira atômica

Record Append - Algoritmo

- Aplicação requisita append
  Solicita que um dado seja armazenado ao fim de um arquivo.
- Cliente GFS traduz a requisição e envia ao master server.
- Master server responde com o chunk handle (manipulador) e localização dos chunkservers (primário e secundários).
- Cliente envia os dados a todos os chunkservers.
- Primário verifica se o dado cabe no chunk
  - Se couber,
    - dado é colocado no chunk(padding)
    - Primário solicita aos secundários fazerem o mesmo e espera confirmação.
    - Informa o cliente
  - Se não couber
    - Coloca o que couber no chunk, notifica os secundários para fazerem o mesmo
    - Notifica o cliente que não coube
    - Cliente deverá refazer o record append no próximo chunk

---

Modelo de consistência

- A consistência de nomes de arquivos nos chunkservers é mantida exclusivamente pelo master.
  O master server centraliza todo o processo de garantia de consistência.
- O estado das atualizações dos chunks depende da execução bem sucedida de appends concorrentes.

- Estados de uma região de um arquivo:
  - Consistente – Dados idênticos em todas os chunkservers
    Os clientes vêem sempre os mesmos dados armazenados em cada chunk server do sistema.
  - Definida – consistente após uma alteração e clientes enxergam todas as alterações não concorrentes realizadas.
    Os clientes vêem o que cada mutação escreveu.
  - Indefinida – consistente após uma alteração e todos os clientes enxergam o mesmo dado, mas não reflete uma sequência coerente das escritas concorrentes.
    Os clientes vêem o dado correto mas não sabem quem fez cada mutação.

- O GFS permite que aplicações diferenciam regiões definidas e indefinidas.
  Fica a critério do usuário resolver as incosistências.

inconsistente: diferentes clientes vendo diferentes dados em um momento.

Modelo de consistência

- GFS garante que o estado de uma região seja definida através:
  Para garantir que a região de um arquivo seja considerada definida:
  - Confirmação das alterações do chunk em todas as réplicas
  - Detecção e exclusão de réplicas obsoletas.
- A localização dos chunks nas caches dos clientes é mantida por tempo limitado para evitar acesso a réplicas obsoletas.

- Uso de checksum para detectar a corrupção de dados.

- Envio de handshake periódico do master para os chunkservers para detectar falhas.

Conclusões

- Permite o uso de hardware comum em ambientes com processamento em larga escala
  grande escalabilidade.
- Trata falhas como normais, ao invés de como exceção.
- Otimizado para grandes arquivos e escrita concorrente (operações de append).
- Tolerante a falhas
  Usa heartbeat para monitoramento contínuo.
  - Monitoramento contínuo
  - Replicação
  - Checksum
- Garante alta vazão, desacoplando o fluxo de controle do fluxo de dados.
  fluxo de controle: master servers.
  fluxo de dados: chunkservers.