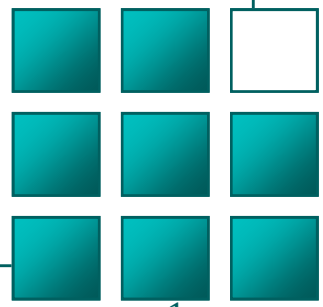
Programa de Pós-graduação em Informática

Tópicos em Sistemas de Computação – Computação em Nuvem

Aletéia Patrícia Favacho de Araújo

Aula 6 – Sistemas de Arquivos Distribuídos e Armazenamento em Nuvem*



*Material adaptado de Carnegie Mellon University in Qatar

Sistemas de Arquivos Distribuídos

- O sistema de arquivo é um componente fundamental de qualquer sistema distribuído.
- Um Sistema de Arquivo Distribuído – DFS é uma implementação distribuída do modelo clássico de tempo compartilhado de um sistema de arquivos, no qual vários usuários compartilham arquivos e recursos de armazenamento.
- O objetivo de um DFS é dar suporte ao compartilhamento de arquivos, mesmo quando eles estão fisicamente dispersos entre as instalações de um sistema distribuído.

Sistemas de Arquivos Distribuídos

- O DFS permite acessar arquivos remotos exatamente como se fossem locais, possibilitando que os usuários acessem arquivos a partir de qualquer ponto da rede.
- Um DFS bem projetado dá acesso aos arquivos armazenados em um servidor com desempenho e confiabilidade semelhantes aos arquivos armazenados em discos locais.

Um Sistema de Arquivos deve...

- Habilitar programas para armazenar e acessar arquivos remotos, como se fossem locais (transparência).
- Permitir a concentração de armazenamento persistente em alguns servidores.
- Fornecer uma interface de programação que caracteriza a abstração de arquivos, liberando os programadores da preocupação com os detalhes.
- Parecer aos seus clientes um sistema de arquivos convencional, centralizado.

Assim...

- A multiplicidade e a distribuição de seus servidores e dispositivos de armazenamento devem se tornar invisível, ou seja, a interface de cliente de um DFS não deverá distinguir entre arquivos locais e remotos.
- O DFS deve localizar os arquivos e realizar o transporte dos dados.
- Em resumo, as responsabilidades de um DFS são:
  - Organização, armazenamento, recuperação, nomeação, compartilhamento e proteção dos arquivos.

Características de um SAD

- Transparência
- Atualizações concorrentes de arquivos
- Replicação de arquivos
- Heterogeneidade do hardware e do SO
- Tolerância a falhas
- Consistência
- Segurança
- Eficiência

Características de um SAD

- Transparência:
  - **Transparência de Acesso:** um único conjunto de operações é fornecido para acesso a arquivos locais e remotos.
  - **Transparência de Localização:** os arquivos podem ser deslocados de um servidor a outro sem alteração de seus nomes de caminho.
  - **Transparência de Mobilidade:** não precisar alterar nada nos programas quando os arquivos forem movidos.

Características de um SAD

- Transparência:
  - **Transparência de Desempenho:** os programas clientes devem trabalhar satisfatoriamente, mesmo que a carga sobre o serviço mude.
  - **Transparência de Mudança de Escala:** o serviço pode ser expandido para tratar com uma ampla variedade de cargas e tamanhos de rede.

Características de um SAD

- Atualizações concorrentes de arquivos
  - As alterações feitas em um arquivo por um único cliente não devem interferir na operação de outros clientes que estejam acessando, ou alterando o mesmo arquivo simultaneamente.
- Replicação de arquivos
  - Em um DFS que suporta replicação, um arquivo pode ser representado por várias cópias em diferentes locais.

Características de um SAD

- Heterogeneidade do hardware e do SO
  - As interfaces de serviços devem ser definidas de modo que os softwares cliente e servidor possam ser implementados para diferentes SOs e computadores.
- Tolerância a falhas
  - Por ser parte fundamental de um SD, é essencial que o serviço de arquivo distribuído continue a funcionar diante de falhas de clientes e servidores.

Características de um SAD

- Consistência
  - Importante mesmo sem replicação, pois é comum que os DFS implementem cache no servidor e/ou no cliente.
- Segurança
  - Deve implementar, no mínimo, mecanismos de controle de acesso.
- Eficiência
  - Um DFS deve obter um nível de desempenho comparável aos sistemas convencionais.

Benefícios de um DFS

- Os Sistemas de Arquivos Distribuídos fornecem:

1. Compartilhamento de arquivos através de uma rede: sem um DFS, teríamos que trocar arquivos, por exemplo por aplicativos de e-mail ou FTP.
2. Acesso transparente de arquivos: programas de um usuário podem acessar arquivos remotos como se eles fossem locais. Os arquivos remotos não precisam de APIs especiais, eles são acessados como os locais.
3. Fácil gerenciamento de arquivos: gerenciar um DFS é mais fácil do que gerenciar vários sistemas de arquivos locais.

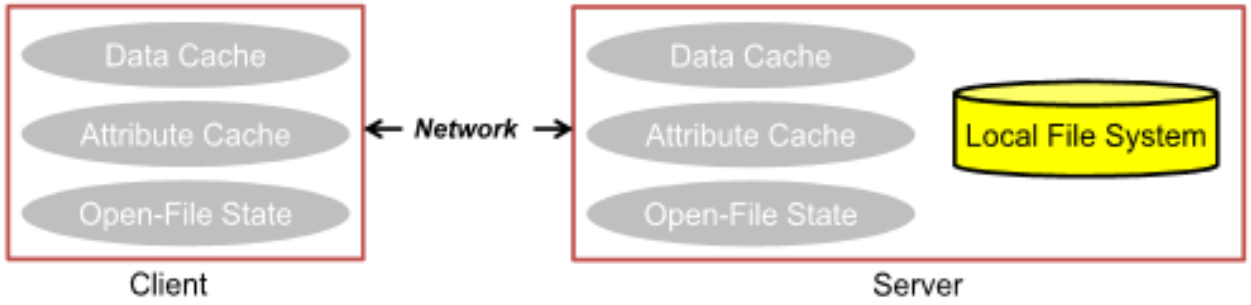
Componentes de um DFS

• Os componentes de um DFS podem ser categorizada como:

1. O estado do dado: este é o contexto do arquivo;
2. O estado do atributo (metadados): esta é a informação sobre cada arquivo (por exemplo, tamanho e lista de controle de acesso de arquivo);
3. O estado de arquivos abertos: isso inclui os arquivos que estão abertos e em uso, e descreve a forma como os arquivos são bloqueados.

Componentes de um DFS

- Os estados do dado e dos atributos residem permanentemente no sistema de arquivos local do servidor, mas as informações recentemente acessadas ou modificadas podem residir no servidor e/ou caches do cliente.
- O estado de arquivos abertos é transitório. Ele muda conforme os processos abrem e fecham os arquivos.



Localização dos Componentes de um DFS

• Três conceitos básicos comandam a estratégia de localização dos componentes de um DFS:

1. Velocidade de Acesso: caches no cliente melhoram consideravelmente o desempenho.

2. Consistência: se há cache de informações nos clientes, todos compartilham a mesma visão?

3. Recuperação: se um ou mais computador quebrar, quais outros são afetados? Quanta informação é perdida?

Aspectos de um DFS

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?
Naming	How is naming often handled in DFSs?
Synchronization	What are the file sharing semantics adopted by DFSs?
Consistency and Replication	What are the various features of client-side caching as well as server-side replication?
Fault Tolerance	How is fault tolerance handled in DFSs?

Arquitetura

Aspect	Description
Architecture	How are DFSs generally organized?

Arquitetura

- Sistema de Arquivos Distribuídos Cliente-Servidor
- Sistema de Arquivos Distribuídos Baseado em *Cluster*
- Sistema de Arquivos Distribuídos Simétrico

DFS Cliente-Servidor

• Os Sistemas de Arquivos Distribuídos baseados na arquitetura Cliente-Servidor trabalham, principalmente, com dois Modelos de Acesso:

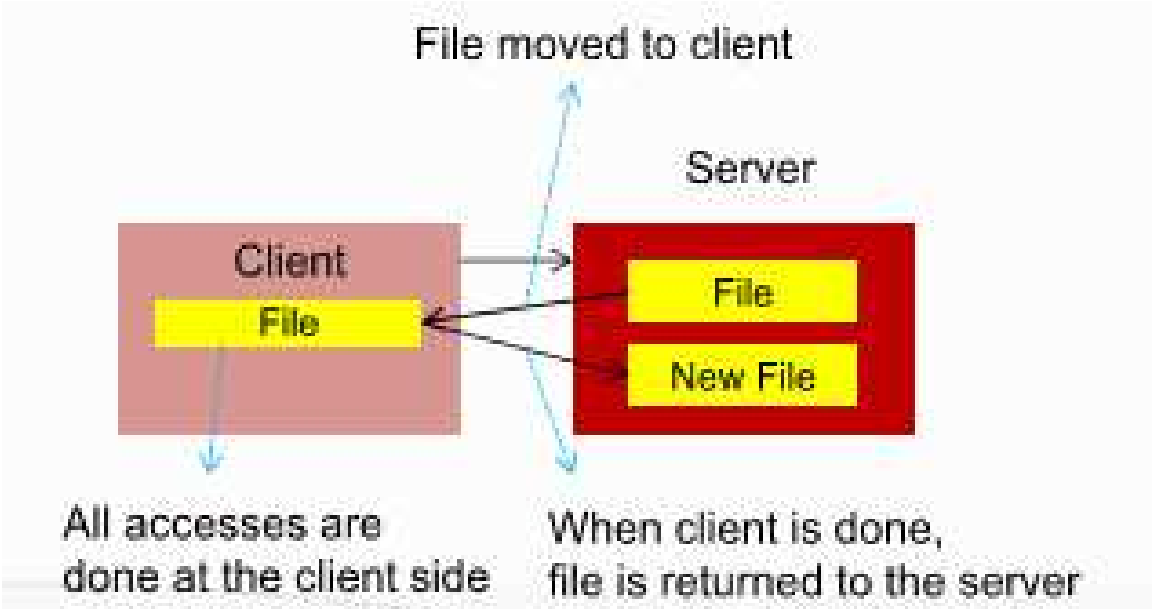
- Modelo de Carga/Atualização (*Upload/Download*)
- Modelo de Acesso Remoto

Modelo de Acesso

- Modelo de Carga/Atualização (*Upload/Download)*
  - O cliente acessa um arquivo localmente, após tê-lo descarregado do servidor.
  - Quando o cliente conclui o uso do arquivo, este é carregado de volta para o servidor, de modo que possa ser usado por um outro cliente novamente.
- Vantagens
  - Eficiência na execução do cliente (acesso local).
- Desvantagens
  - Espaço suficiente nos clientes.
- Exemplo: *Andrew File System* (AFS)

Modelo de Acesso

- Modelo Carga/Atualização *(Upload/Download)*

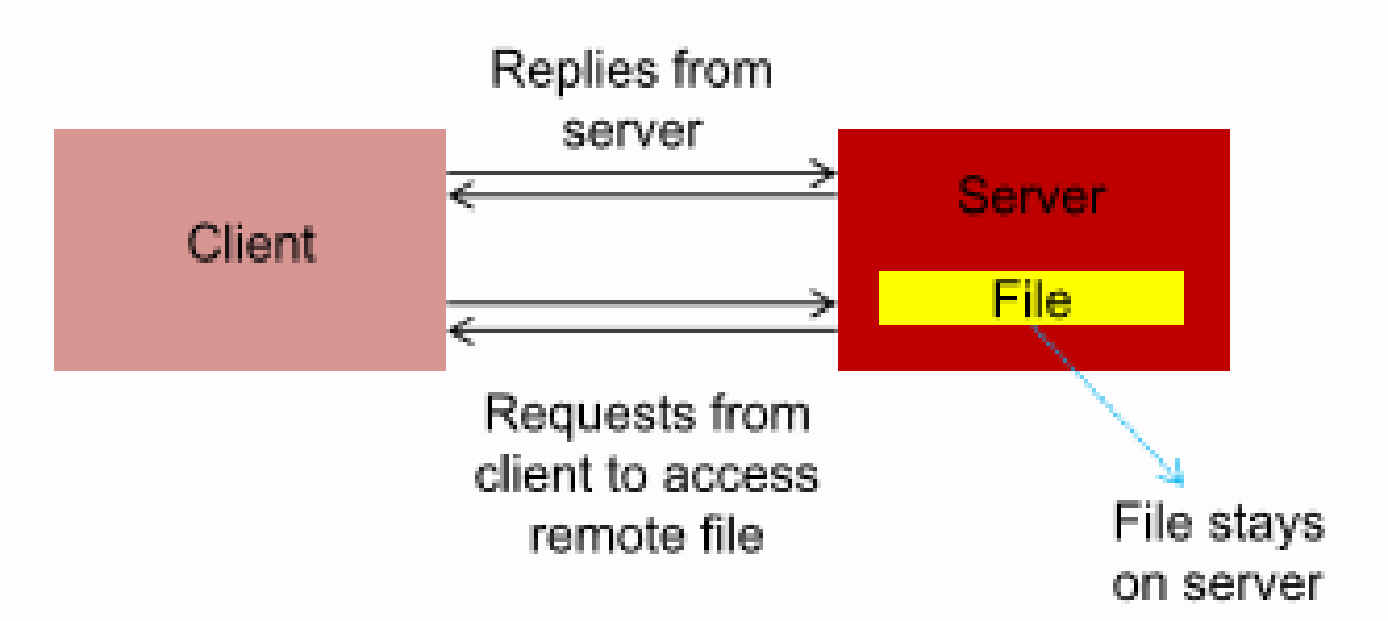


Modelo de Acesso

- Modelo de Acesso Remoto
  - É oferecida ao cliente somente uma interface que contém várias operações sobre arquivos, mas o servidor é responsável por implementar essas operações.
  - Os arquivos permanecem no servidor.
- Vantagem:
  - Requer pouco espaço de armazenamento nos clientes.
  - Pode seguir o modelo cliente/servidor (request/reply) ou usar *Caching* => desempenho melhor
- Exemplos:
  - com *caching:* NFS, Locus, Sprite, Spring
  - sem *caching:* Amoeba

Modelo de Acesso

- Modelo de Acesso Remoto:

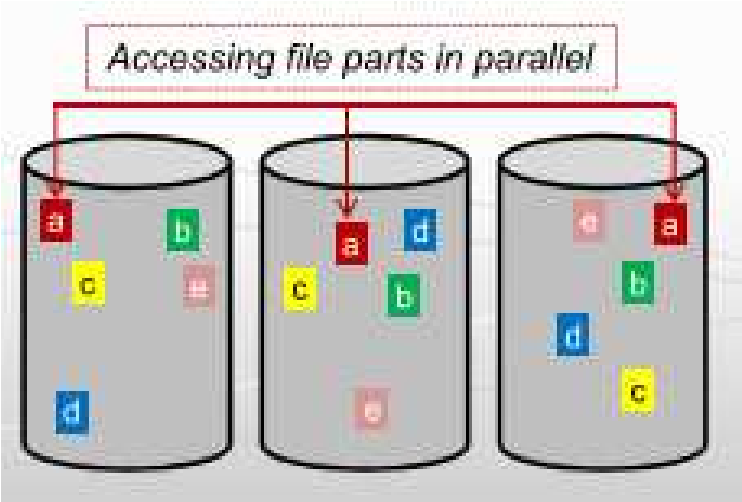


DFS Baseados em *Cluster*

- Aplicações de computação em nuvem e de HPC executam, normalmente, em milhares de nós de computação e manipulam grandes volumes de dados.
- O DFS baseado em *cluster* é um componente essencial para o desempenho escalável dessas aplicações, pois eles dividem e distribuem os grandes arquivos utilizando técnicas de *striping*, para permitir que os dados sejam acessados simultaneamente;
- Exemplos:
  - Google File System (GFS) e S3 são Sistemas de Arquivos Distribuídos para Nuvem;
  - Parallel Virtual File System (PVFS) e General Parallel File System (GPFS) são Sistemas de Arquivos Distribuídos para HPC.

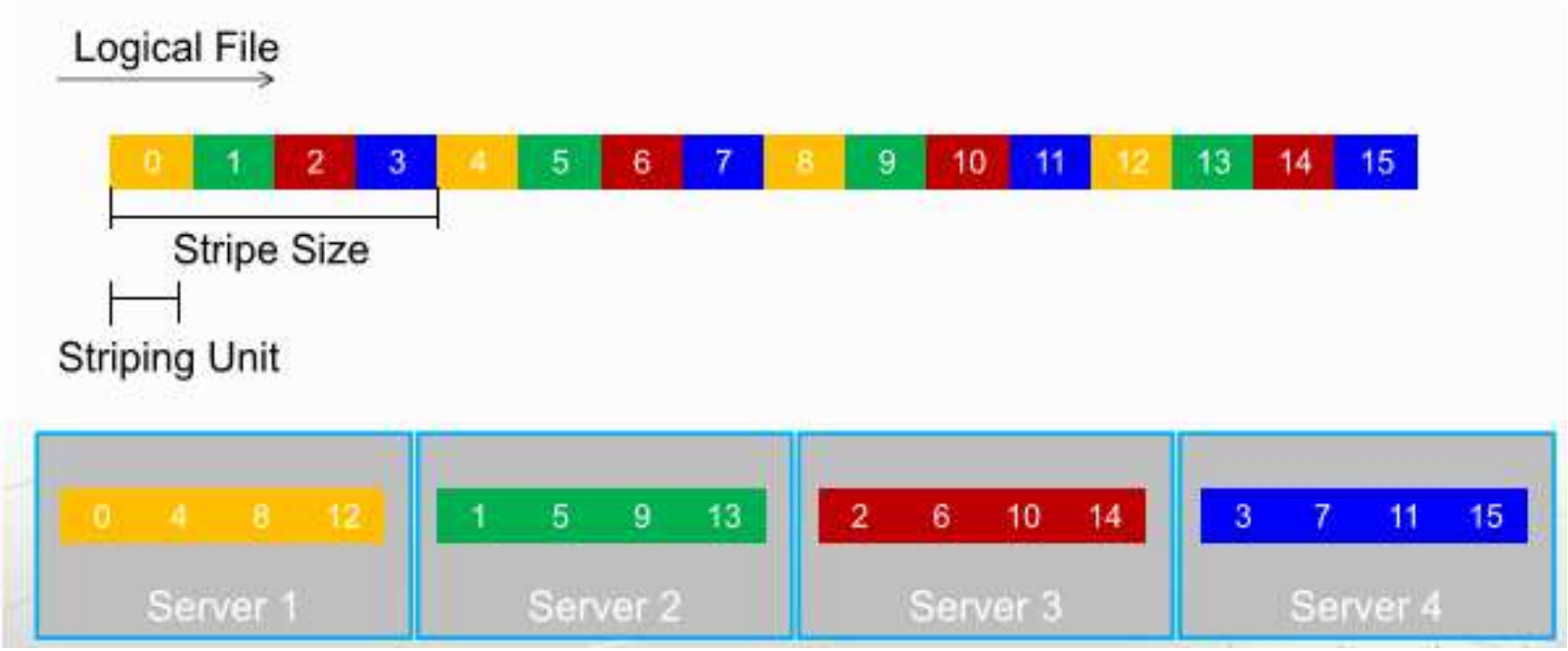
Técnica de *Striping* de Arquivos

- O arquivo é dividido em partes, as quais são distribuídas em diversos servidores.
- Isso torna possível obter as diferentes partes do arquivo em paralelo.



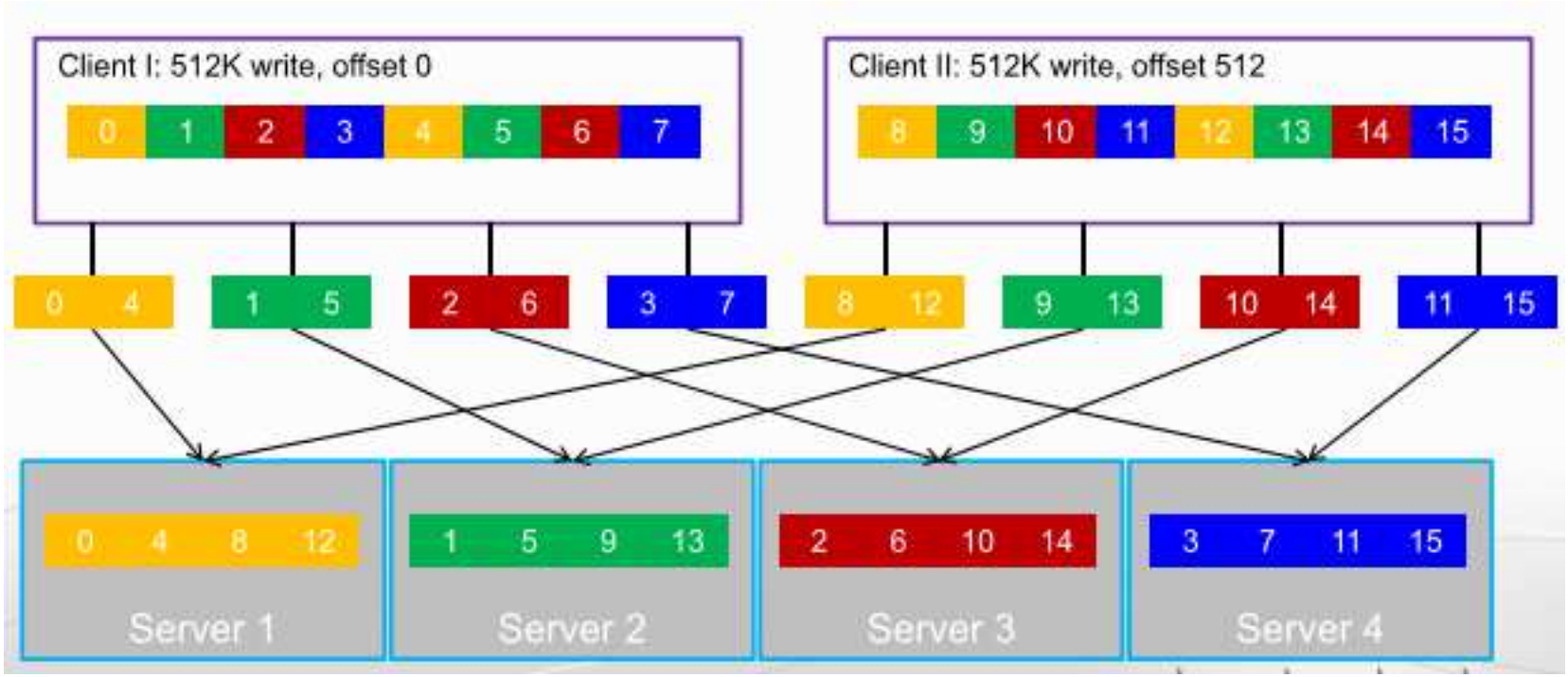
Política de Distribuição
Round Robin

- Como distribuir um arquivo em múltiplas máquinas?
  - *Round-Robin* é tipicamente uma solução razoável.

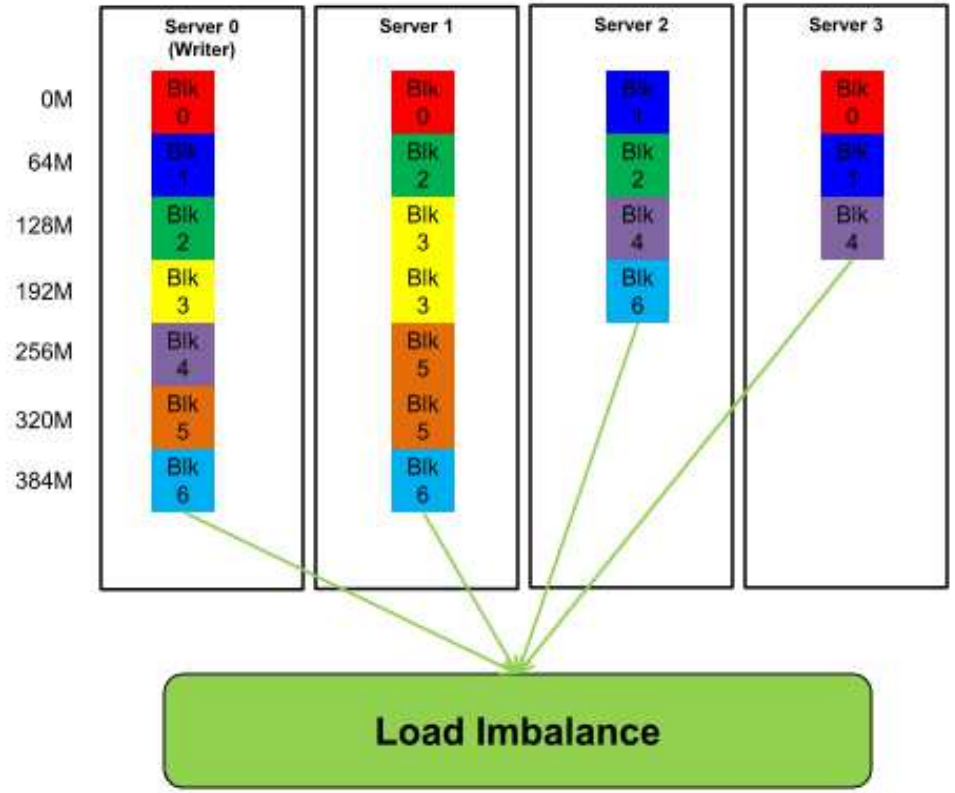


Política de Distribuição *Round Robin*

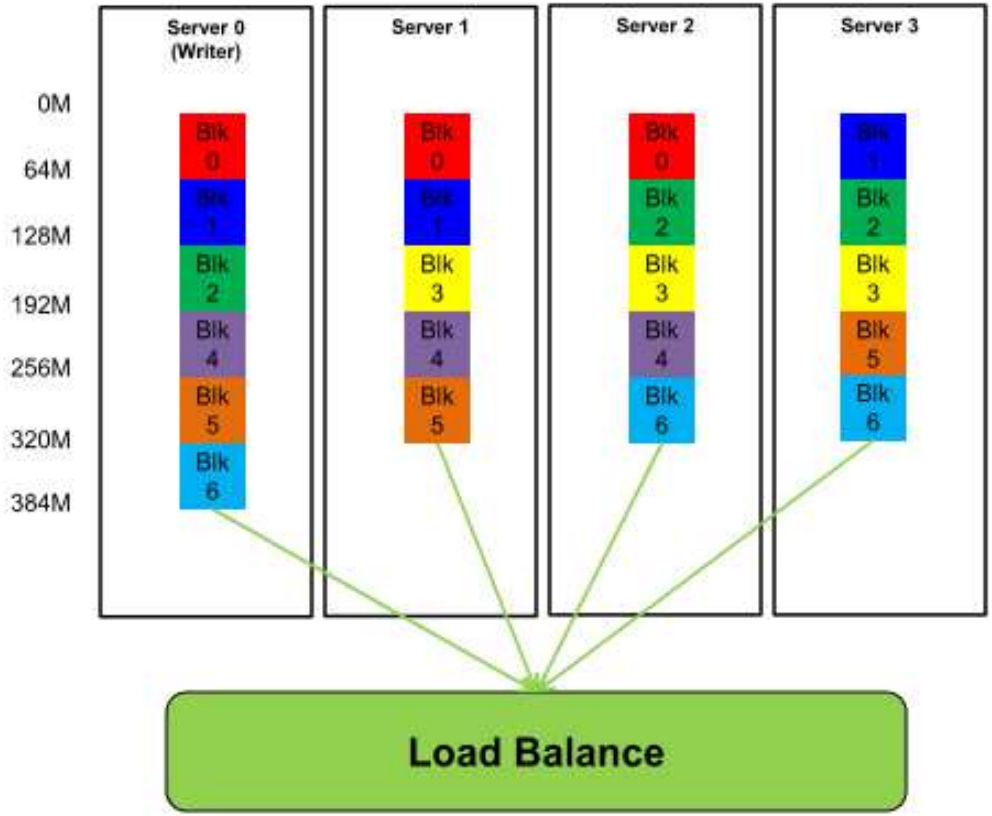
- Clientes realizam escritas/leituras em várias regiões:



Política de Distribuição Randômica - Desbalanceada



Política de Distribuição Randômica - Balanceada



DFS Simétrico

- Os Sistemas de Arquivos Distribuídos Simétricos se baseiam na tecnologia *peer-to-peer.*
- Assim, todas as máquinas do ambiente distribuído podem armazenar os arquivos que serão acessados.
- As implementações atuais, como por exemplo o sistema Ivy, baseiam-se em tabelas *hash* para a distribuição de dados, e combinadas com uma chave baseada em mecanismo de pesquisa.

Aspectos de um DFS

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?

Processos

• O aspecto mais importante a respeito de processos DFS é se eles devem ser *stateless* ou *stateful*.

1. Abordagem *Stateless*:
   - Não requer que os servidores mantenham qualquer estado do cliente;
   - Quando um servidor falha, não há necessidade de entrar em uma fase de recuperação para colocar o servidor para um estado anterior;
   - Bloquear um arquivo não pode ser feito facilmente;
   - Por exemplo, NFSv3 e PVFS (sem cache do lado do cliente).

Servidores *Stateless*

- Quando um cliente envia uma requisição a um servidor, esse processa a requisição, envia a resposta, e então remove de suas tabelas internas todas as informações relativas à requisição que acabou de ser processada.
- Assim, entre duas requisições, o servidor não mantém armazenado nenhum tipo de informação específica de nenhum cliente.
- Cada requisição deve trazer todas as informações necessárias ao seu processamento no servidor. Isso aumenta o tamanho da mensagem requerida.

Servidores *Stateless*

- Vantagens:
  - Simplicidade
  - Tolerância a falhas
  - Chamadas *open e close desnecessárias*
  - Espaço do servidor não desperdiçado com tabelas de estados
  - Número de arquivos em uso (abertos) ilimitado

Processos

2. Abordagem *Stateful*:
   - Requer que o servidor mantenha algum estado do cliente;
   - Os clientes podem fazer uso efetivo de caches, mas isso implica no uso de um protocolo de coerência de cache eficiente;
   - Fornece um servidor com a capacidade de suportar chamadas de retorno (ou seja, a capacidade de fazer RPC para um cliente), a fim manter o controle de seus clientes;
   - Exemplos: NFSv4 e HDFS.

Servidores *Stateful*

- Neste caso, os servidores devem manter informações de estado dos clientes no intervalo de tempo entre duas requisições.
- Depois de abrir um arquivo, o cliente recebe um descritor de arquivo, para ser usado em todas as chamadas subseqüentes para identificar esse particular arquivo.
- Ao receber uma requisição, o servidor usa o descritor de arquivos para determinar qual dos arquivos está sendo solicitado.

Servidores *Stateful*

- Vantagens:
  - Mensagens de requisição menores
  - Melhor desempenho no controle dos arquivos armazenados
  - Facilidade de fazer bloqueio em um arquivo

Aspectos de um DFS

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?

Comunicação

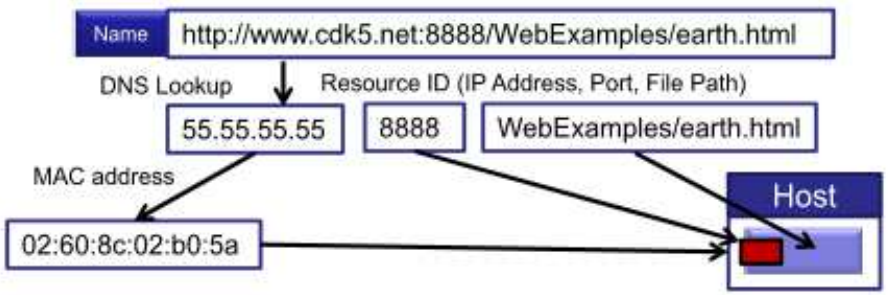
- Comunicação em DFSs é baseada, principalmente, em chamadas de procedimento remoto (RPC).
- A principal razão para a escolha do RPC é fazer com que o sistema seja independente das camadas subjacentes, tais como SO, redes e protocolos de transporte.
- O GFS (*Google File System*), por exemplo, usa RPC e pode quebrar um arquivo em várias chamadas de procedimento remoto para aumentar o paralelismo.

Aspectos de um DFS

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?
Naming	How is naming often handled in DFSs?

Nomeação

- Os nomes são usados exclusivamente para identificar entidades em sistemas distribuídos.
  - As entidades podem ser processos, objetos remotos, etc.
- Nomes são mapeados para localização de uma entidade usando uma resolução de nomes.
- Um exemplo de resolução de nomes:



Aspectos de um DFS

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?
Naming	How is naming often handled in DFSs?
Synchronization	What are the file sharing semantics adopted by DFSs?

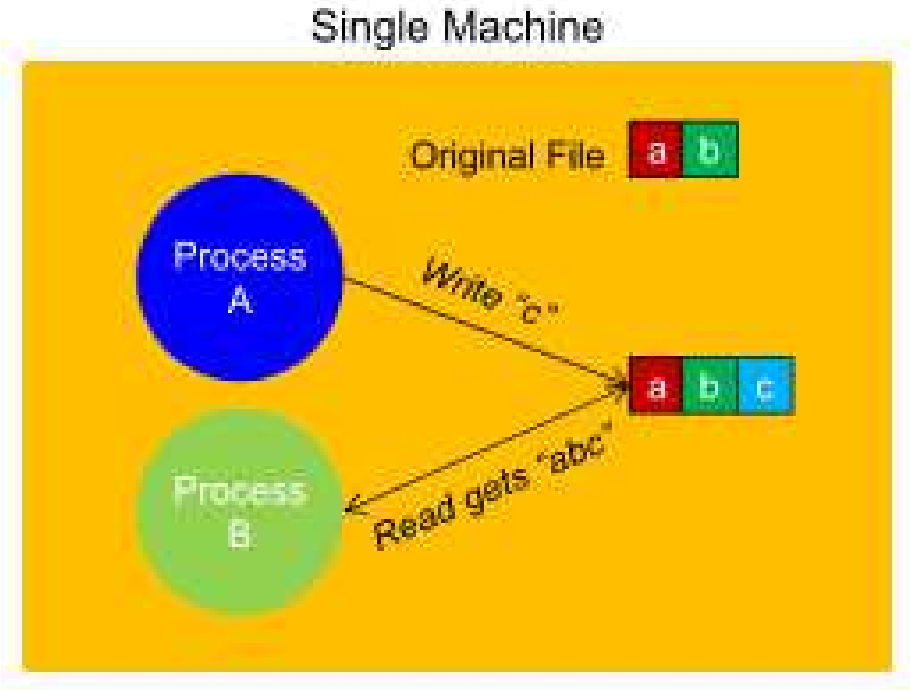
Sincronização em DFSs

- Semânticas de Compartilhamento de Arquivos
- Gerenciamento de *Locks*

Semântica Unix em um Sistema de Processador Único

- Sincronização de sistemas de arquivos não seria um problema se os arquivos não fossem compartilhados.
- Quando dois ou mais usuários compartilham o mesmo arquivo ao mesmo tempo, é necessário definir a semântica de leitura e escrita.
- Em um sistema com um processor apenas, uma operação de leitura após uma gravação retornará o valor que terminou de ser escrito.
- Esse modelo é chamado de Semântica Unix.

Semântica Unix em um Sistema de Processador Único

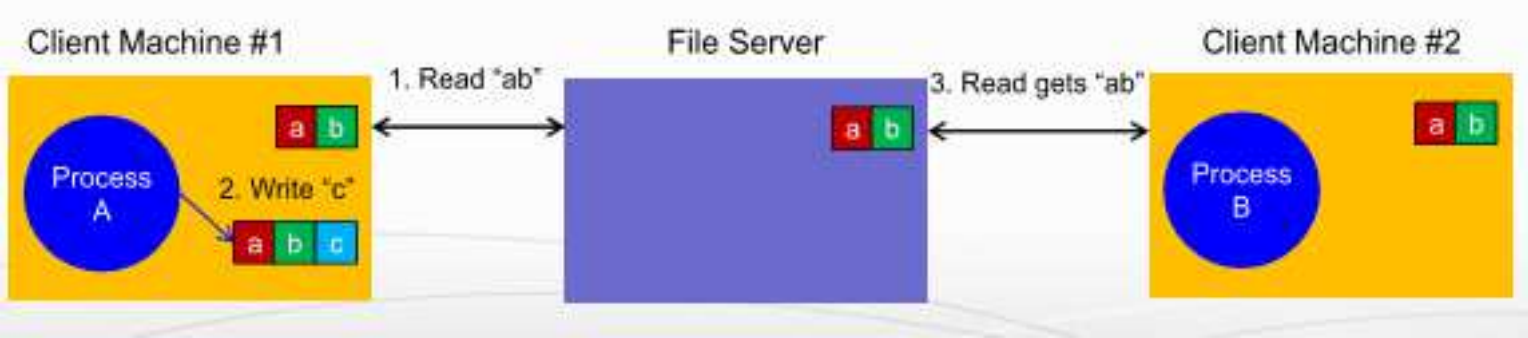


Sincronização em DFS

- Em um DFS, a semântica Unix pode ser facilmente alcançada se houver apenas um servidor de arquivos e clientes sem cache.
- Assim, todos vão ler e escreve diretamente a partir do servidor de arquivos, por meio de processos estritamente sequenciais.
- Essa regra garante a semântica UNIX, mas o desempenho facilmente degrada, pois todos as solicitações vão para o mesmo servidor.

Semântica Unix e Cache

- O desempenho de um DFS com um único servidor de arquivos pode ser melhorada por meio do uso de cache.
- Se um cliente, no entanto, localmente modifica um arquivo de cache, e logo em seguida um outro cliente lê o arquivo a partir do servidor, ele irá obter um arquivo obsoleto.



Semântica de Sessão

- Uma solução alternativa é mudar a regra do que deve ser considerado consistente, relaxando a regra de semântica de compartilhamento.
- Assim, a regra diz “mudanças em um arquivo aberto são visíveis apenas p/ o processo que modificou o arquivo. Só quando o arquivo for fechado é que as mudanças serão visíveis para os demais processos”.
- Essa regra é conhecida como **semântica de sessão.**
  - Ela não garante a leitura dos valores reais (atuais).

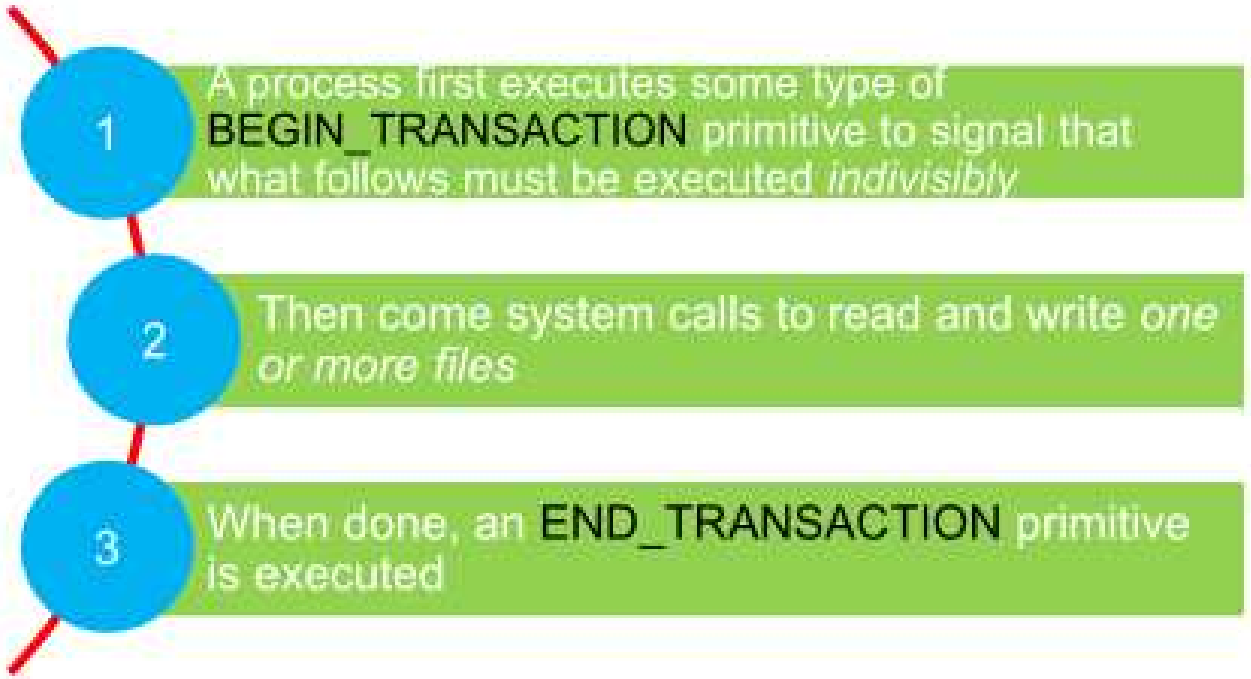
Semântica Imutável

- Uma proposta diferente é não permitir alterações nos arquivos já criados.
- Essa é a **semântica dos arquivos imutáveis.**
  - Nesse caso, as únicas operações possíveis nos arquivos são *create e read.*

Transações Atômicas

- Outra possibilidade de tratar arquivos compartilhados é usando as **transações atômicas.**
  - Enquanto um processo estiver alterando um arquivo, ninguém poderá manipular esse arquivo, só após a transação ser finalizada.
- A propriedade fundamental é que o sistema garante que todas as chamadas contidas na transação serão executadas em ordem, sem nenhuma interferência de outra transação concorrente.

Transações Atômicas

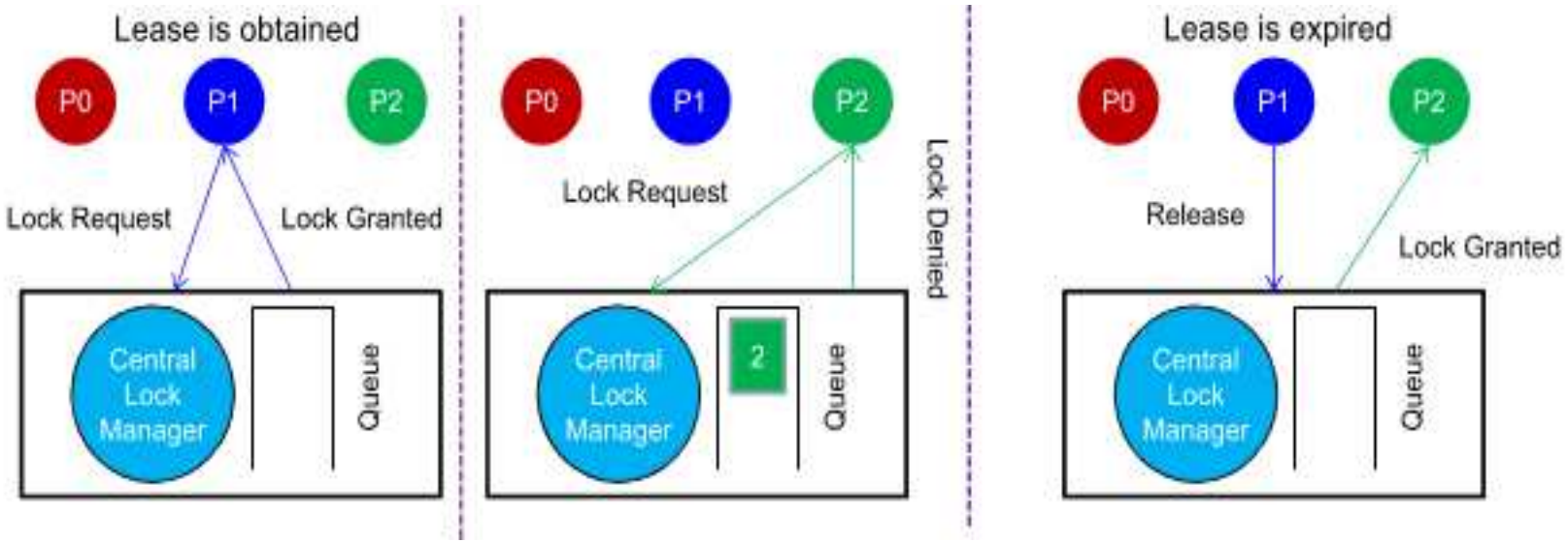


Semânticas de Compartilhamento de Arquivos

Abordagem	Comentário
Semântica do tipo UNIX	Cada operação em um arquivo é visível a todos os processos
Semântica de Sessão	Nenhuma mudança é vista por outros processos até a execução de um CLOSE
Arquivos Imutáveis	Nenhuma atualização é permitida
Transações Atômicas	Operações do tipo "tudo-ou-nada"

Gerenciador de *Lock* Central

- Um gerenciador central de bloqueio pode ser implantado, onde o acesso a um recurso compartilhado é sincronizado através de concessão.



Aspectos de um DFSs

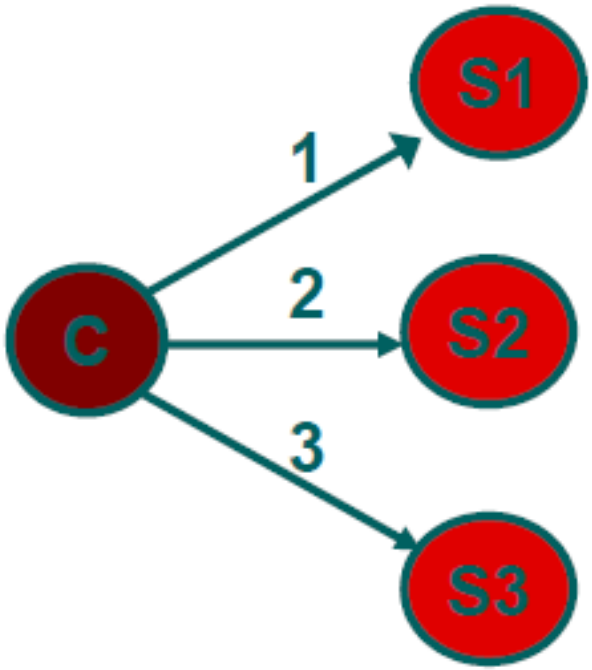
Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?
Naming	How is naming often handled in DFSs?
Synchronization	What are the file sharing semantics adopted by DFSs?
Consistency and Replication	What are the various features of client-side caching as well as server-side replication?

Replicação de Arquivos

- A replicação é um dos segredos da eficácia dos sistemas distribuídos, pois ela pode fornecer um melhor desempenho, alta disponibilidade e tolerância a falhas.
- Assim, há duas razões primárias para replicar dados:
  - Confiabilidade
    - Aumento da confiabilidade nos dados.
    - Aumento da disponibilidade dos servidores.
  - Desempenho
    - Aumento do desempenho por meio da divisão da carga de trabalho.

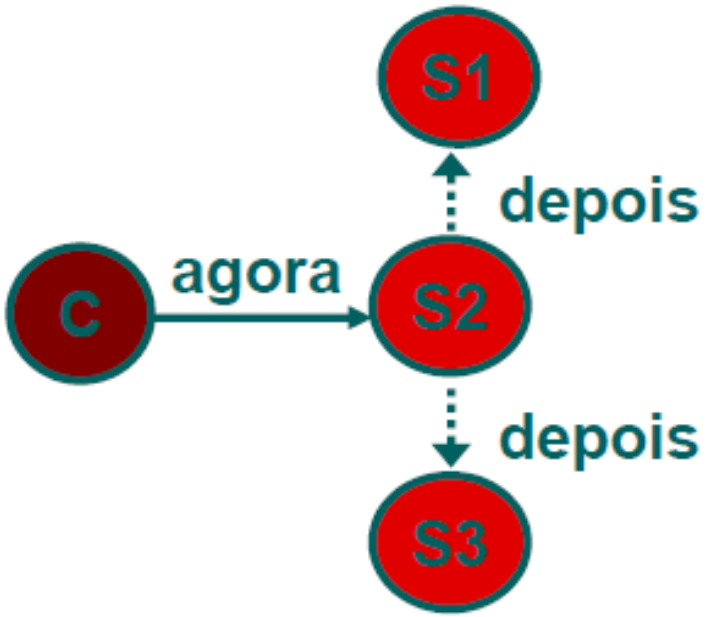
Replicação Explícita

- O programador controla todo o processo;
- O processo cria arquivos em servidores específicos;
- Um sistema de arquivos distribuído deve proporcionar replicação transparente.



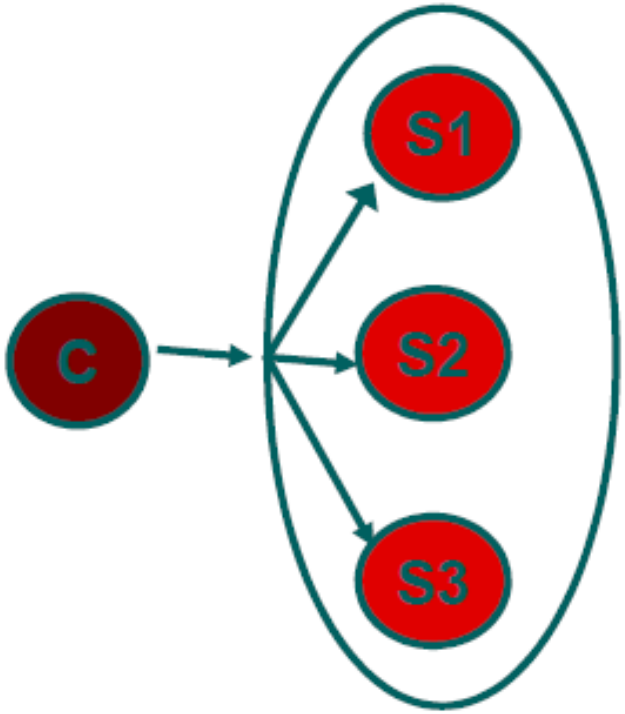
Replicação Lenta ou Preguiçosa (*Lazy file replication*)

- Somente uma cópia de cada arquivo é criada em um nó;
- Mais tarde, o servidor replica o arquivo automaticamente, sem conhecimento do programador;
- Sistema deve prover mecanismos para recuperar as cópias.



Replicação Usando Grupos

- Utiliza comunicação para grupo;
- Todos os comandos de escrita são enviados simultâneamente para todos os servidores;
- Cópias extras são feitas ao mesmo tempo em que a original está sendo criada;
- Cliente envia arquivo para um endereço de grupo.



Aspectos de um DFSs

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?
Naming	How is naming often handled in DFSs?
Synchronization	What are the file sharing semantics adopted by DFSs?
Consistency and Replication	What are the various features of client-side caching as well as server-side replication?

Consistência da Cache

- O uso de cache introduz o problema de inconsistência no sistema.
- Métodos para reduzir a inconsistência:
  - Escrita Direta
  - Escrita Atrasada
  - Escrita no Fechamento

Algoritmos para Consistência da Cache

- **Escrita Direta (*Write-through*):**
  - Nesse caso, quando um arquivo da cache (ou um bloco) for modificado, o novo valor é mantido na cache, mas também é enviado imediatamente ao servidor.
- **Escrita Retardada (*Write-back*):**
  - Nesse caso, o cliente simplesmente faz uma nota indicando que o arquivo foi atualizado.
  - Uma vez a cada 30 segundos (por exemplo), todos os arquivos modificados são identificados e enviados juntos ao servidor de uma vez só.
  - Em geral, uma única escrita grande é mais eficiente do que muitas escritas pequenas.

Algoritmos para Consistência da Cache

- Algoritmo **Escrita no Fechamento (*Write-on-close)*:**
  - Este algoritmo é idêntico a semântica de sessão.
  - O arquivo só será atualizado na memória principal quando ele for fechado.
  - Assim, todas as atualizações feitas durante o período de abertura do arquivo acontecem somente na memória cache.

Aspectos de um DFS

Aspect	Description
Architecture	How are DFSs generally organized?
Processes	• Who are the cooperating processes? • Are processes *stateful* or *stateless*?
Communication	• What is the typical communication paradigm followed by DFSs? • How do processes in DFSs communicate?
Naming	How is naming often handled in DFSs?
Synchronization	What are the file sharing semantics adopted by DFSs?
Consistency and Replication	What are the various features of client-side caching as well as server-side replication?
Fault Tolerance	How is fault tolerance handled in DFSs?

Falhas

- As falhas podem acontecer devido a uma variedade de razões:
  - Falhas de hardware;
  - *Bugs* de Software;
  - Interrupções de Rede.
- Um sistema é dito falho quando não puder cumprir as suas expectativas.
- Assim, um sistema é dito tolerante a falhas se ele for capaz de continuar operando mesmo diante de falhas.

Exemplos de DFSs

- GFS (Google File System)
- HDFS (Hadoop Distributed File System)
- NFS (Network File System)
- AFS (Andrew File System)
- PVFS (Parallel Virtual File System)
- GMAILFS
- CODA (COnstant Data Availability)
- GlusterFS

Google File System - GFS

- O que é?
  - Sistema de Arquivos Distribuído proprietário projetado e implementado pela Google em ~2000;
  - O projeto do GFS foi publicado como artigo em congresso em 2003;
  - Implementação *open-source*: HDFS (*Hadoop Distributed File System*).
- Por que é importante?
  - Por que ao contrário dos DFSs anteriores, ele faz novas escolhas de arquitetura (baseada em *cluster*) e implementação que privilegiam a escala atual de dados.

Motivação Arquitetural

- *Clusters* de hardware de prateleira
  - 100x ou 1.000x máquinas;
  - Falhas são a regra e não a exceção;
  - Necessidade de monitoramento constante, e detecção e recuperação de falhas automática.
- Muitos arquivos enormes -Terabytes (TB), Petabytes (PB).
- Maioria dos arquivos são *append-only,* e uma vez escritos, são lidos inúmeras vezes, de forma sequencial.
  - Em geral, eles são atualizados mais por anexar dados a um arquivo do que por atualizar partes de um arquivo.
- Exemplos: e-mails enviados/recebidos, fóruns e *web crawl data*.

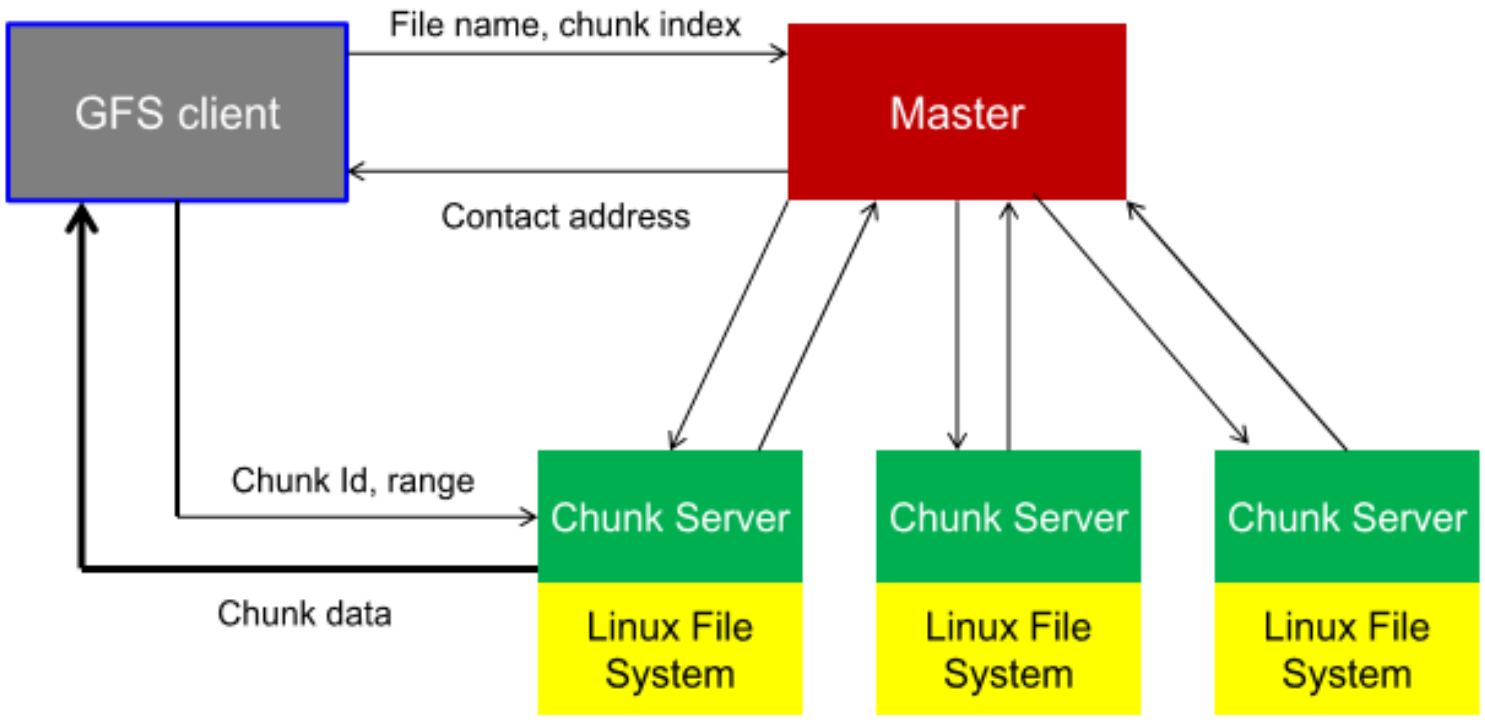
Arquitetura

- Um *cluster* GFS é dividido em dois tipos de servidores:
  - Master
    - Somente um por *cluster*
    - Gerencia os metadados
  - Chunckserver
    - Vários por *cluster*
    - Gerencia os blocos de dados
- Metadados: árvore de diretórios, *chunkservers*, permissões de acesso, mapeamento arquivo <--> bloco, identificação das máquinas onde se encontram cada bloco, etc.

Arquitetura

- Cada arquivo no GFS é dividido em blocos (*chunks*) de tamanho fixo (64MB ou 128MB), e armazenado no sistema de arquivos do SO. O NFS, por exemplo, trabalha com blocos de 8 Kbytes, e o AFS com blocos de 64 Kbytes.
- Para garantir a tolerância a falhas, cada bloco é replicado entre várias máquinas.
- O Master conhece todos os *Chunkservers* e se comunica periodicamente com eles.
- Uma aplicação cliente se comunica através de uma API com ambos os tipos de servidores (embora transparente).

Arquitetura do GFS



GFS – Google File System

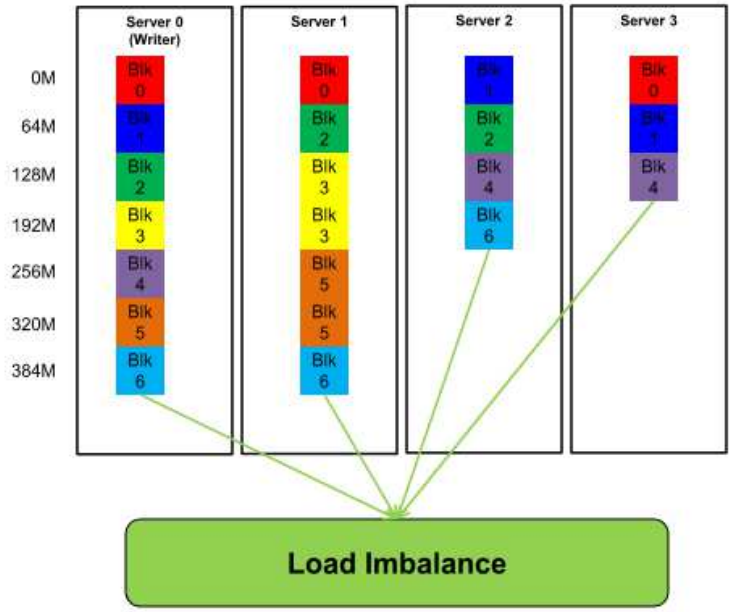
- Um cliente GFS passa ao mestre um nome de arquivo e um índice de *chunk*, esperando um endereço de contato para o mesmo.
- O endereço de contato contém todas as informações para acessar o *chunkserver* correto para obter o bloco do arquivo requisitado.
- Clientes GFS ficam sabendo apenas quais *chunkservers* o mestre informou, pois neles estão armazenados os dados solicitados.
- Além disso, os *chunkservers* mantêm uma contabilidade do que tem armazenado.

GFS – Google File System

- Quanto a escalabilidade, duas decisões tratam essa questão:
  - O trabalho árduo é executado pelos *chunkservers*;
  - O espaço de nomes para arquivos é implementado com a utilização de uma tabela simples, de um só nível, localizada em memória RAM, na qual nomes de caminho são mapeados para metadados.
- Essa organização permite que um único mestre controle algumas centenas de *chunkservers*, o que é um tamanho considerável para um único *cluster.*

Política de Distribuição de Dados GFS

• O GFS distribui esses *chunks* usando uma política de distribuição randômica, sem se preocupar com balanceamento de carga entre os servidores.

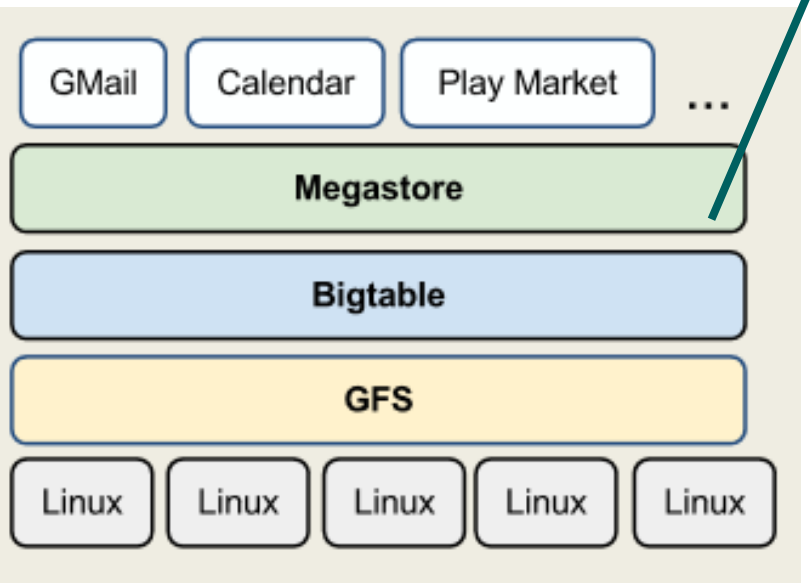


Limitações do GFS

- Um único mestre é SPoF (*Single Point of Failure*)
  - Solução: Mestre secundário (em *stand by*)

- Não adequado para aplicações interativas ou com muitos arquivos pequenos.

Atualmente no Mundo Google



Camada de software que adiciona algumas características (linguagem de consulta tipo SQL, transações atômicas multi-registro, índices secundários, etc) acima do BigTable. A Google fez isso porque para algumas aplicações ela precisava garantir as propriedades ACID.

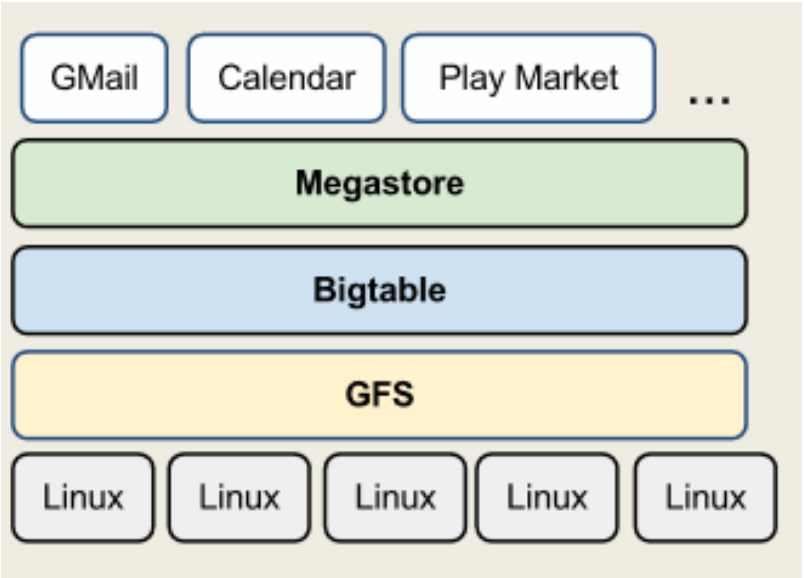
Spanner (BigTable 2)

Colossus (GFS 2.0)

Perspectivas

- Colossus - GFS 2.0
  - Vários servidores mestres
  - Particionamento automática de metadados
  - Replicação comandada pelo cliente
  - Suporte a aplicações interativas

No Mundo do Software Livre



HBase

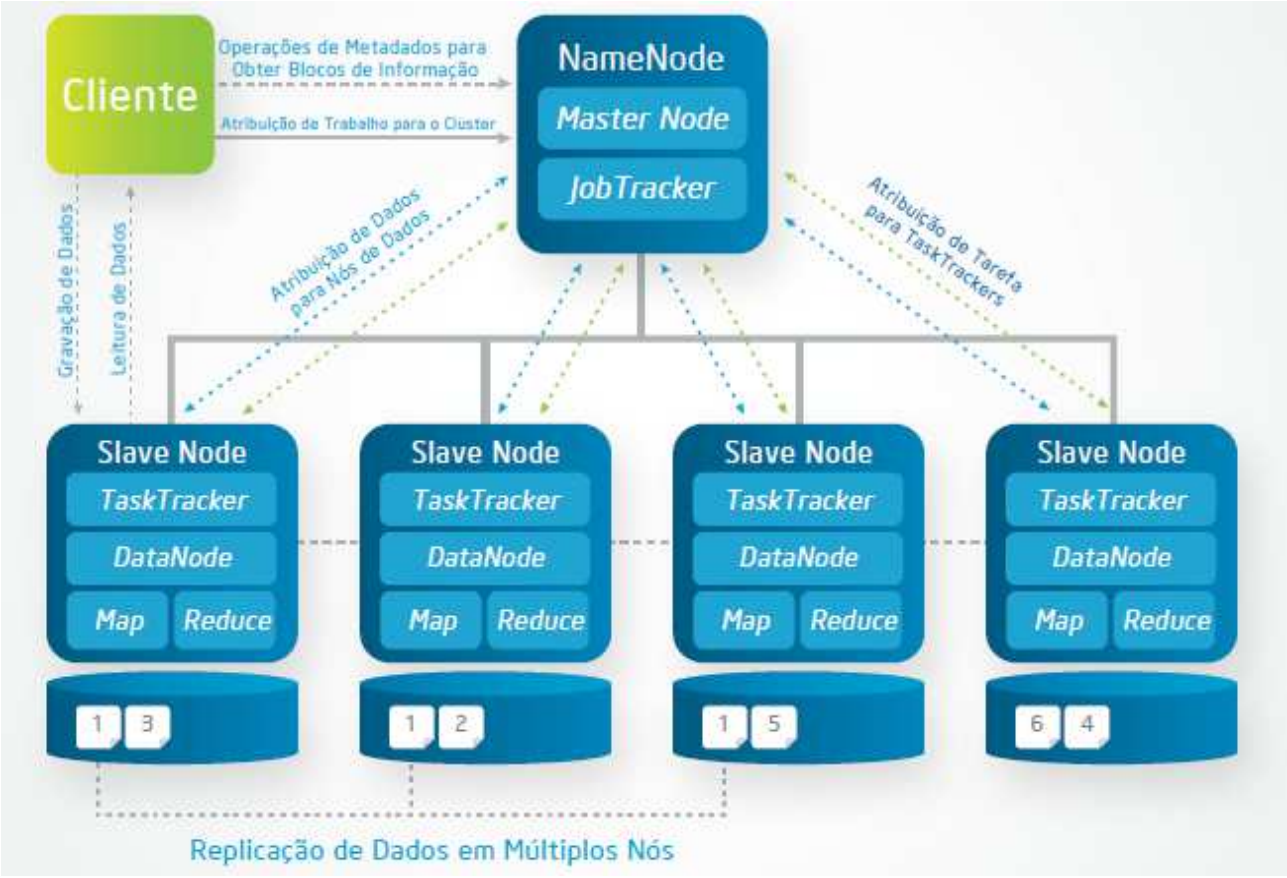
HDFS

Hadoop Distributed File System – HDFS

- O HDFS é um sistema de arquivos distribuído integrado ao arcabouço Hadoop.
- Esse sistema teve forte inspiração no GFS, entretanto, uma das diferenças deve-se ao fato de que o HDFS é um arcabouço de código aberto, e foi implementado 100% na linguagem Java.
- Ele permite o armazenamento e a transmissão de grandes conjuntos de dados em máquinas de baixo custo.
- E possui mecanismos que o caracteriza como um sistema altamente tolerante a falhas.
- Sua arquitetura...

Master	→	NameNode
Chunkserver	→	DataNode (ou SlaveNode)

Arquitetura do HDFS



Arquitetura do HDFS

- Um *cluster* HDFS consiste em um único NameNode (o mestre) e vários DataNodes/SlaveNodes (os escravos).
- O NameNode é o elemento central do HDFS, e por isso é recomendável ser implantado em um nó exclusivo e, preferencialmente, o nó com melhor desempenho do *cluster*.
- Ainda por questões de desempenho, o NameNode mantém todas suas informações em memória RAM.
- O NameNode gerencia o espaço de nomes do HDFS e regulariza o acesso dos clientes aos arquivos:
  - É um árbitro e um repositório para todos os metadados do HDFS, e determina o mapeamento de blocos para os DataNodes.

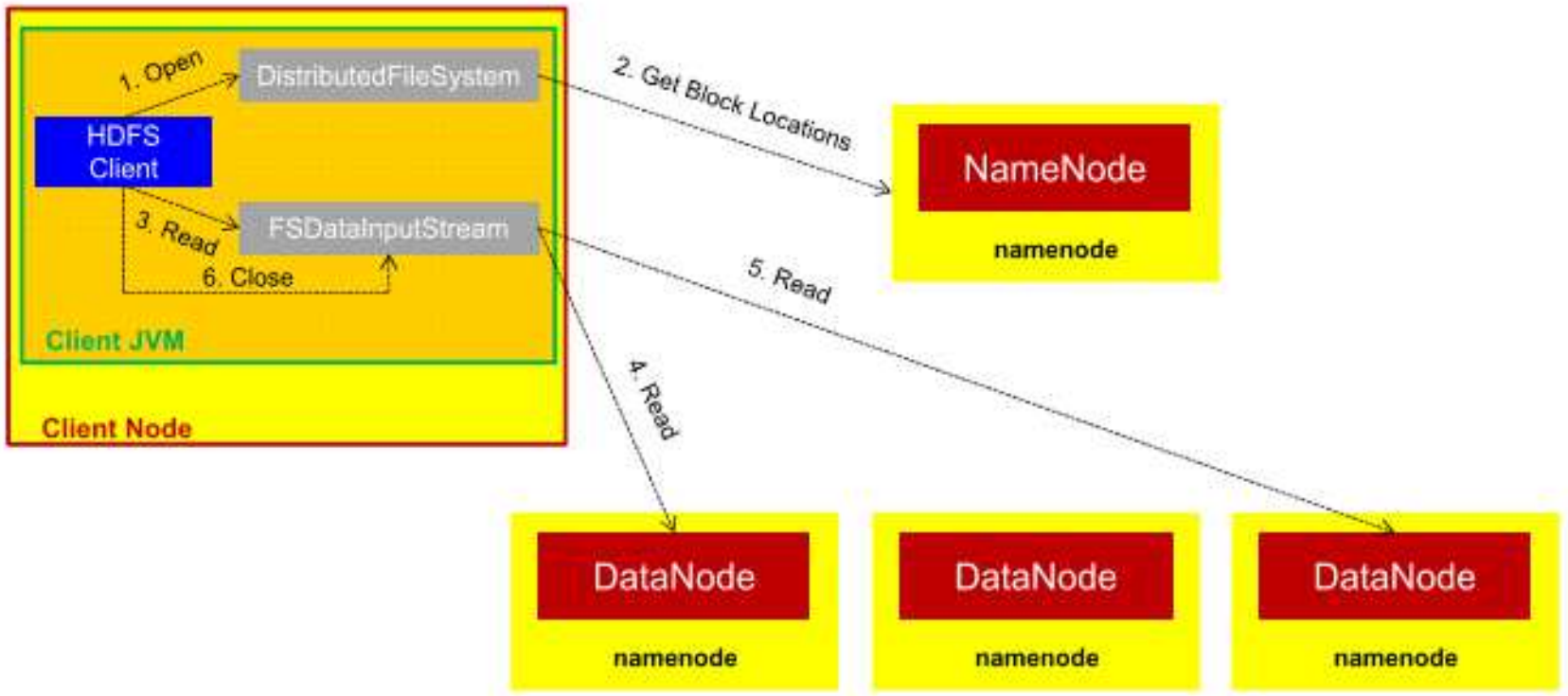
Arquitetura do HDFS

- Para desempenhar seu papel de gerenciar todos os blocos de arquivos, o NameNode possui duas estruturas de dados importantes: o FsImage e o EditLog.
- O primeiro arquivo (FsImage) é o responsável por armazenar informações estruturais dos blocos, como o mapeamento e os *namespaces* dos diretórios e arquivos, e a localização das réplicas desses arquivos.
- O segundo, EditLog, é um arquivo de *log* responsável por armazenar todas as alterações ocorridas nos metadados dos arquivos.

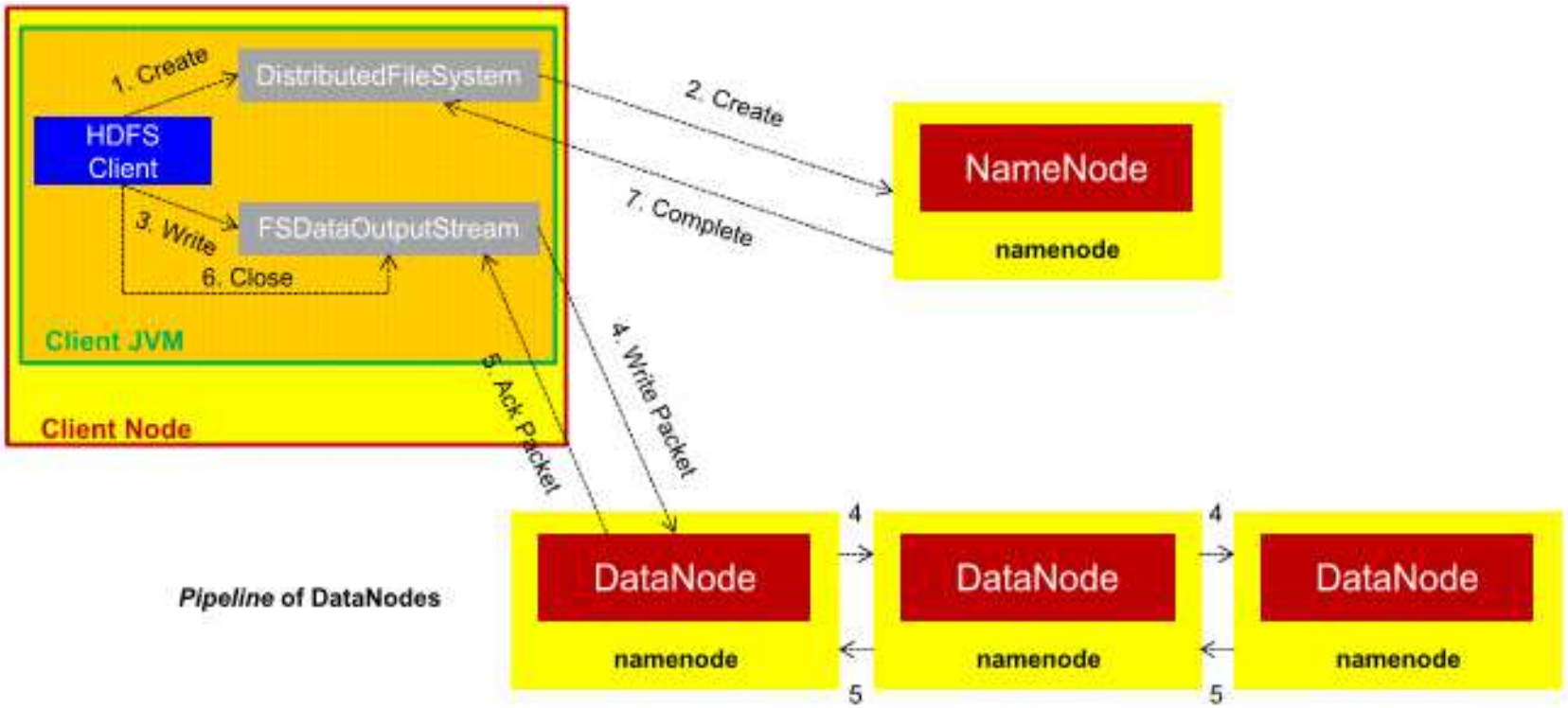
Arquitetura do HDFS

- Os DataNodes são responsáveis por manter os arquivos de dados:
  - Eles são responsáveis por servir os pedidos de leitura e escrita dos clientes;
  - Eles executam a criação, a exclusão e a replicação de blocos de acordo com as instruções do NameNode.

Operação de Leitura a Partir de um Cliente



Operação de Escrita a Partir de um Cliente

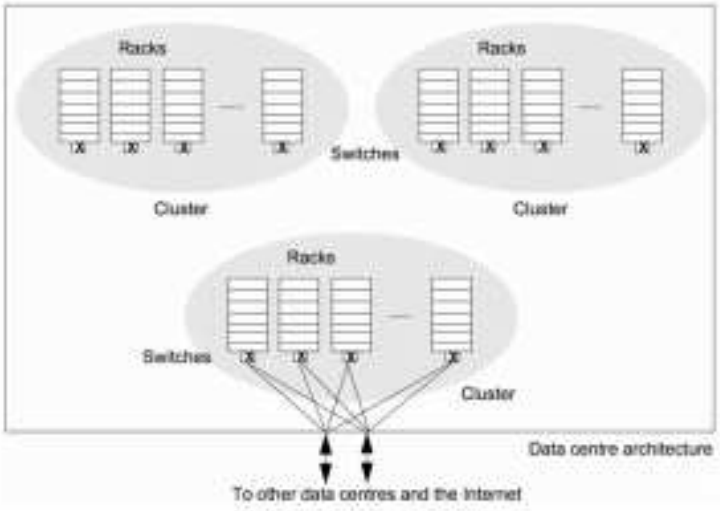


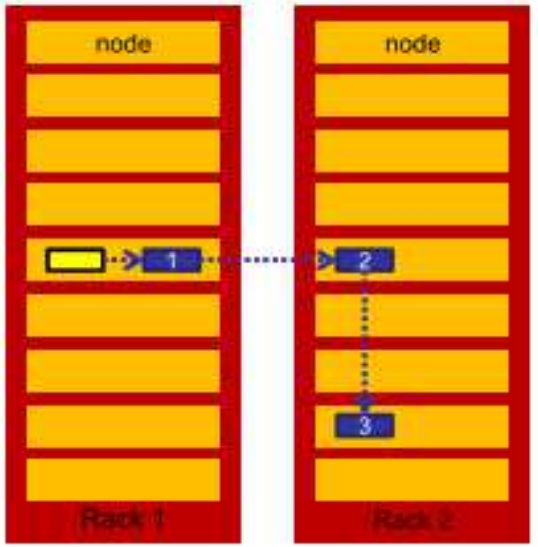
Replicaçãlo do HDFS

- Por questões estratégicas de segurança, o HDFS organiza a armazenagem dos blocos dos arquivos, e suas réplicas, em diferentes máquinas e armários.
- Assim, mesmo ocorrendo uma falha em um armário inteiro, o dado pode ser recuperado e a aplicação não precisará ser interrompida.
- Para cada bloco foi definido um fator de replicação padrão de 3 (três) unidades.

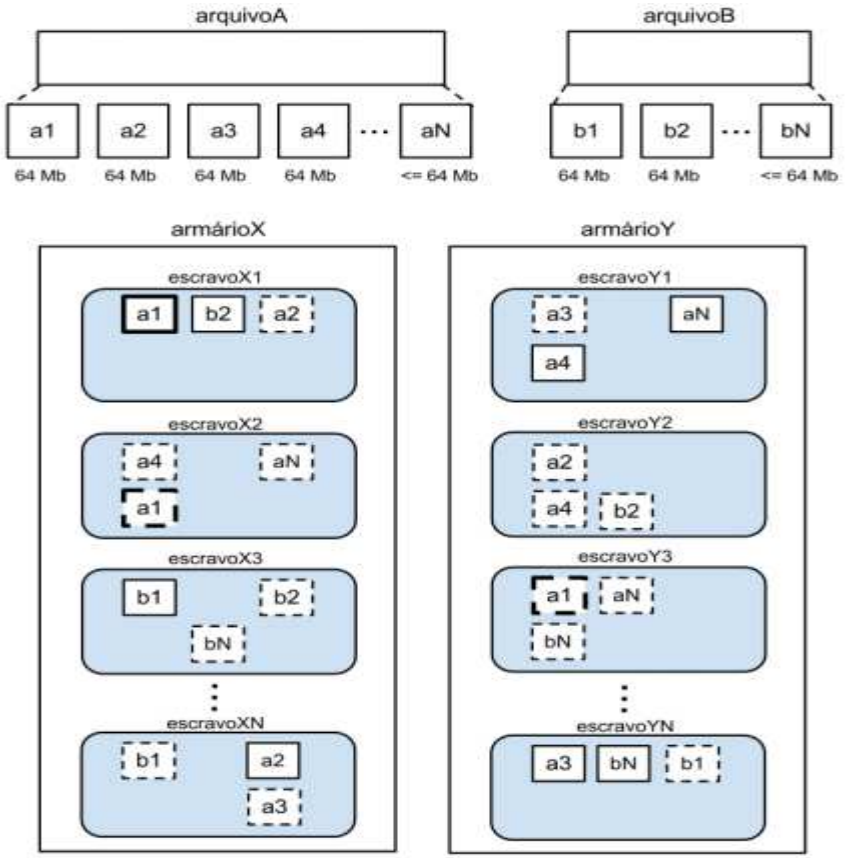
Tolerância a Falhas no HDFS

- As falhas são tratadas no HDFS por meio da replicação:
  - O lugar de armazenamento da replica é escolhido de maneira que garanta proteção contra falhas de nós e *rack.*





Replicação do HDFS



Falhas no DataNode

- Falhas no DataNode são detectadas por meio de um mecanismo de *heartbeat* (batimento cardíaco):
  - DataNodes enviam periodicamente mensagens para o NameNode para indicar que eles estão vivos;
  - A desconexão de rede pode causar um subconjunto de DataNodes sem conectividade com o NameNode;
  - O NameNode marca os DataNodes sem mensagens recentes como mortos e não transmite quaisquer novos pedidos de IO para eles;
  - Os DataNodes mortos podem fazer com que o fator de replicação de alguns blocos fique abaixo de seu valor especificado;
  - Assim, o NameNode constantemente rastreia quais blocos precisam ser replicados e inicia a replicação sempre que necessário.

Falhas no NameNode

- Falha no NameNode provocam a queda completa do HDFS, pois os arquivos não podem ser mapeados para os respectivos blocos em DataNodes:
  - O HDFS usa um mecanismo de *checkpointing* denotado como o NameNode Secundária;
  - O NameNode secundário não é um NameNode *backup* (por isso, ele não pode assumir a função do NameNode primária em cima de um falha NameNode);
  - O NameNode secundário detém uma cópia *out-of-date* de estado persistente do primário, que, em casos extremos, pode ser usado para recuperar o estado de metadados do HDFS;
  - Na versão 2 há um verdadeiro NameNode de *backup.*

Semântica do HDFS

- O HDFS segue a semântica dos arquivos imutáveis:
  - Ele não permiti que os clientes modifiquem um arquivo existente.
  - Todas as operações de gravação devem ser feitas para novos arquivos.

HDFS versão 2

- A principal diferença é que no HDFS versão 1 era um software monolítico, *stand alone*, assim como o Map-Reduce e outros do Hadoop.
- No Hadoop 2 foi introduzido o YARN, um escalonador de recursos em *cluster* que virou uma espécie de núcleo das aplicações Hadoop.
- Assim, muita funcionalidade que estava replicada foi integrada no YARN e tanto o HDFS quanto o Map-Reduce passaram a ser aplicações que usam o YARN.

HDFS versão 2

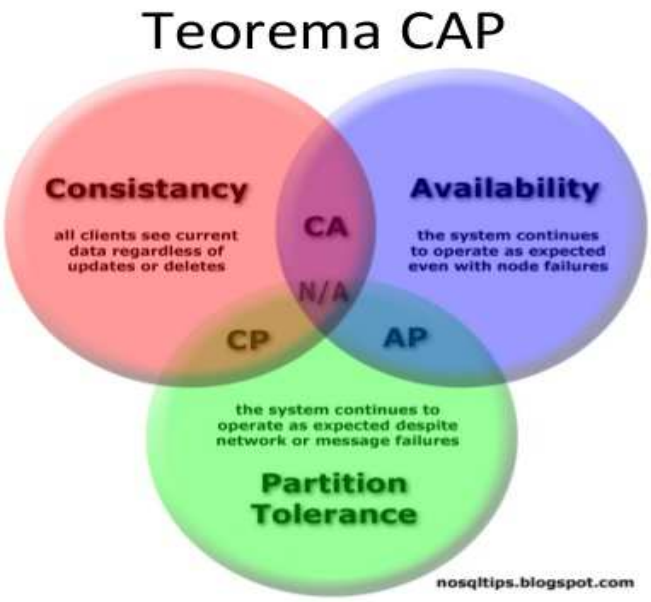
- Além disso, o HDFS versão 2 tem:
  1. *Failover* automático com *hot standby*;
     - Isto significa que é possível haver dois servidores funcionando simultaneamente: o servidor **principal** e o servidor **reserva**, este último como contingência do primeiro. Caso o servidor principal falhe, o servidor reserva entra em ação imediatamente.
  2. Interface NFS para acesso de leitura/escrita ao HDFS;
  3. Mecanismos de recuperação através de snapshots;
  4. Segurança, criptografia e autenticação.

Teorema CAP

- O teorema CAP apresenta três requisitos que influenciam mutuamente o desenho e a instalação de sistemas de arquivos distribuídos: consistência, disponibilidade e tolerância a partição (**C**onsistency, **A**vailability, and **P**artition tolerance).
- O Teorema CAP foi descoberto por Eric Brewer e foi apresentado pela primeira vez em 19 de Julho de 2000 no simpósio da ACM intitulado “Principles of Distributed Computing”.
- A grande revelação de Brewer é que não é possível que um sistema de dados distribuído atender esses 3 requisitos em simultâneo, apenas 2.

Teorema CAP

- O Teorema CAP revolucionou a arquitetura dos sistemas distribuídos de dados em grande escala, por exemplo a arquitetura dos sistemas da Amazon, eBay e Twitter.



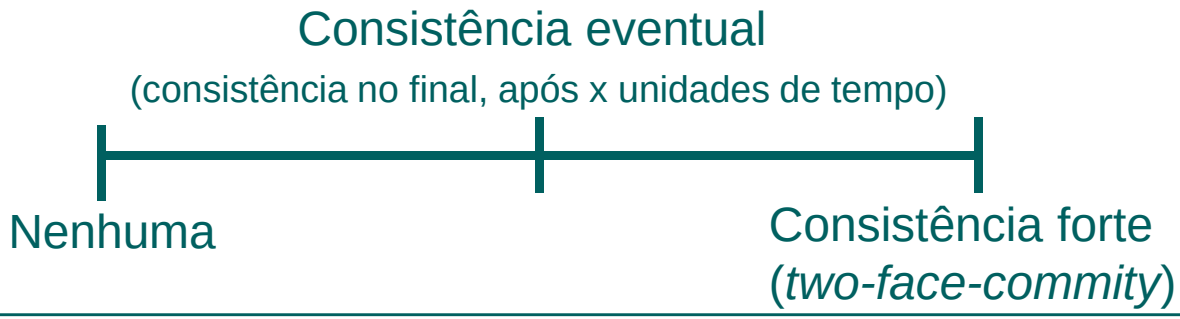
Fonte: http://blog.nosqltips.com/2011/04/cap-diagram-for-distribution.html

Consistência

- Um sistema de dados distribuído consistente é classificado como fortemente consistente ou como tendo alguma forma fraca de consistência.
- O exemplo mais conhecido de um sistema fortemente consistente são os sistemas que implementam as características ACID (**A**tomicidade,**C**onsistência, **I**solamento e **D**urabilidade), implementado pela maioria das bases relacionais.
- Na outra extremidade tem-se as bases de dados BASE (***B**asic **A**vailability, **S**oft-state, **E**ventual consistency*).

Consistência

- Na maioria das vezes, a consistência fraca é fornecida sob a forma de **consistência eventual**, o que significa que a base de dados pode atingir um estado consistente.
- Apenas existe a garantia de que a versão mais recente de um registro está consistente num nó, sendo que versões antigas deste registro podem ainda existir em outros nós, eventualmente todos os nós têm a última versão.
- Ver a consistência como uma janela...



Disponibilidade

- Estar disponível 24 x 7;
- Assim a falha/atualização de um nó não deve implicar na indisponibilidade do sistema.
- Note que a disponibilidade não é só uma proteção contra falhas de hardware/software, mas também uma forma de obter balanceamento de carga e operações concorrentes.
- Um sistema pode ser mais ou menos disponível num período de tempo. No entanto, em um determinado momento T está ou não está disponível.

Tolerante a Partição

- Tolerância a partição refere-se à capacidade de um sistema continuar a operar na presença de uma falha de rede.
- Por exemplo, se tivermos uma base de dados a funcionar em 80 nós em cima de 2 *racks* e a ligação entre os dois *racks* falhar, a base de dados fica particionada.
- Se o sistema é tolerante a partição, então a base de dados é capaz ainda de fornecer operações de leitura/escrita enquanto particionada.
- Assim, mesmo que não haja consistência, o ambiente deve manter as 2 partições funcionando.

Teorema CAP na Prática

- Há uma corrente forte que diz que não é adequado abrir mão (em Computação em Nuvem) da tolerância a partição.
- Logo, é válido ser CP (Consistência & Tolerante a Partição) ou AP (Disponibilidade & Tolerante a Partição).
- O CA (Consistência & Disponibilidade) não é válido para a indústria de nuvem.
- Atualmente, a maioria dos sistemas NoSQL é AP.

Mensagem...

Reconhecer quais dos requisitos do teorema CAP são importante para o nosso sistema é o primeiro passo para construir com sucesso um sistema distribuído, escalável e com alta disponibilidade.