Em um sistema distribuído os arquivos podem ser armazenados e acessados não necessariamente de forma local. Assim os arquivos precisam estar adaptados por estarem sujeitos a falhas.

Requisitos do SAD

- Transparência
  - Acesso – aplicações não devem conhecer a distribuição dos arquivos. Acesso local e remoto semelhante.
    O acesso deve ser semelhante de forma local e remota.
  - Localização - a nomeação dos arquivos deve se manter independente da localização.
    Os nomes devem se manter os mesmos independente da localização física.
  - Mobilidade – arquivos podem se mover sem afetar as aplicações.
    A movimentação não deve refletir na aplicação.
  - Desempenho - a variação na carga do sistema não deve refletir tanto no funcionamento das aplicações.
    A carga no sistema não deve interferir muito no desempenho.
  - Mudança de escala – deve permitir a expansão do serviço para lidar com as variações de carga.
    O aumento do número do cliente não deve interferir. A escalabilidade é importante.
- Atualizações concorrentes de arquivos
  - Deve suportar acessos simultâneos a arquivos, garantindo consistência através de travas(locks)
    Como são acessados remotamente pode ocorrer acessos concorrentes, deve usar travas para evitar condições de corridas.

Requisitos do SAD

- Replicação de arquivos - permite balanceamento de carga e tolerância a falhas
  É necessário que se use mecanismos de réplicas para lidar com falhas. E

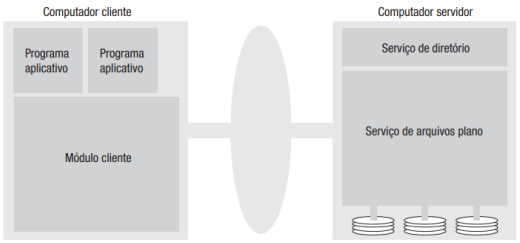
balanceamento de carga para escalabilidade.

- Heterogeneidade do hardware e do SO
  Devem ser acessados em qualquer plataforma.
- Tolerância a falhas
  Deve-se implementar as semânticas de "mais de uma vez", "apenas uma vez"
- Consistência
  Uma mesma versão do arquivo seja vista por todos os processos do sistema.
- Segurança – Controle de acesso via ACL
  Mecanismos eficientes baseado em listas de controle de acesso (com quem pode acessar e o que se pode fazer).
- Eficiência – desempenho deve ser comparável ao dos sistemas locais
  Depende da estrutura de conexão de rede.

Estudos de caso

- O NFS – Network File System
  - Padrão Internet – RFC 1813.
    Padronizar o sistema de arquivos e garantir alguns requisitos como transparência de acesso.
  - Fornece acesso transparente a arquivos remotos
  - Cada computador pode ser cliente e/ou servidor.
  - Suporte para heterogeneidade de hardware e sistema operacional
    máquinas diferentes acessam NFS de maneira transparente. Serve para UNIX e Windows.
- O GFS – Google File System
  Sistema de arquivos do Google para suporte das aplicações que disponibilizam. O sistema é extremamente escalável.
  - Sistema de arquivos escalável para aplicações de distribuição intensiva de dados.
  - Utilizado pela Google para organizar e manipular grandes arquivos e permitir que aplicações consigam usar os recursos necessários.
    - Exemplo: Gmail, Youtube e Google Maps

Arquitetura do serviço de arquivos



Servidor:

- **Serviço de arquivos plano:** Permite acesso aos arquivos armazenados em meios persistentes.
- **Serviço de diretório:** organiza os arquivos com identificadores e disponibiliza uma estrutura hierárquica.

Cliente:

- **Módulo cliente:** Garante transparência ao cliente, disponibiliza uma interface comum de acesso ao servidor de arquivos.

Arquitetura do serviço de arquivos

- Serviço de arquivos plano – implementa as operações sobre o conteúdo dos arquivos.
    - UFID – Unique File Identifiers
      Identificador único para cada arquivo. Normalmente são longas sequências de bits.
- Serviço de diretório – fornece o **mapeamento entre uma UFID e o nome do arquivo**
  Garante assim um sistema de arquivos hierárquico.
    - Cliente do serviço de arquivos planos
- Módulo Cliente – fornece para as aplicações uma interface única com operações comuns sobre arquivos.
  Abrir, fechar, ler, escrever.
    - Pode utilizar uma cache para melhorar o desempenho.
      Garante acesso mais rápido.

Read(FileId, i, n) → Data — gera *BadPosition*	Se *1 ≤ i ≤ Length(File)*: lê uma sequência de até *n* elementos de um arquivo, começando no elemento *i*, e a retorna em *Data*. Gera uma exceção se o valor *i* é inválido.
Write(FileId, i, Data) — gera *BadPosition*	Se *1 ≤ i ≤ Length(File)+1*: grava uma sequência de *Data* em um arquivo, começando no elemento *i*, ampliando o arquivo, se necessário. Gera uma exceção se o valor *i* é inválido.
create( ) → FileId	Cria um novo arquivo de tamanho zero e gera um UFID para ele.
Delete(FileId)	Remove o arquivo.
GetAttributes(FileId) → Attr	Retorna os atributos do arquivo.
SetAttributes(FileId, Attr)	Configura os atributos do arquivo (somente os atributos que não estão sombreados na Figura 12.3).

field: identificador;
i: posição da leitura;
n: quantidade de elementos a ser lido no arquivo;

Data: dado lido.

Comparação com o UNIX

NFS x UNIX

- Semelhanças
  - As primitivas do UNIX e do NFS são equivalentes.
    As mesmas operações de abrir, fechar, ler, escrever.
- Diferenças do NFS
  O que muda é como as operações são implementadas.
  - **Não há operações open e close** (o acesso aos arquivos é imediato)
  - As operações read e write tem um parâmetro que indica o ponto de partida para a operação
  - Operações podem ser repetidas (idempotentes) – Exceção: create

  Pode-se realizar a mesma operação de leitura N vezes, porém o "create" não, pois não pode-se criar o mesmo arquivo mais de uma vez.
  - Servidores são sem estado(stateless)
    Não é guardada por exemplo a posição inicial de leitura.

Controle de acesso

- No UNIX, o UID é utilizado para verificar os direitos de acesso ao arquivo.
  A verificação é feita no momento do login. O UID define se o usuário tem direito de acesso.
- Nos NFS, a cada requisição do cliente, o UID do usuário é enviado.
  Pois não se mantém histórico de operações em arquivos.

Interface do serviço de diretório

Lookup(Dir, Name) → *FileId* — gera *NotFound*	Localiza o nome textual no diretório e retorna o UFID relevante. Se *Name* não estiver no diretório, gera uma exceção.
AddName(Dir, Name, FileId) — gera *NameDuplicate*	Se *Name* não estiver no diretório, adiciona (*Name, File*) no diretório e atualiza o registro de atributos do arquivo. Se *Name* já estiver no diretório, gera uma exceção.
UnName(Dir, Name) — gera *NotFound*	Se *Name* estiver no diretório, a entrada contendo *Name* é removida do diretório. Se *Name* não estiver no diretório, gera uma exceção.
GetNames(Dir, Pattern) → *NameSeq*	Retorna todos os nomes textuais presentes no diretório que correspondam à expressão regular *Pattern*.

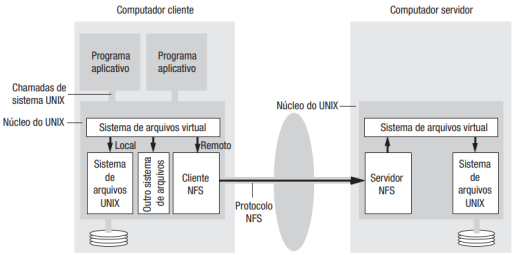
Estudo de caso: Sun NFS

NFS: Padrão aberto usado na internet para implementação de sistemas de arquivos distribuídos.

- Protocolo NFS
  Permite chamadas remotas de procedimentos RPC para clientes e realizar operações de armazenamento remoto.

  - Conjunto de chamadas de procedimento remoto para clientes realizarem operações em armazenamento remoto
  - É independente de SO, mas originalmente desenvolvido para UNIX.
  - Composto por um módulo servidor e um módulo cliente.
    Servidor disponibiliza acesso ao sistema de arquivos local e o cliente que acessa esse sistema.
  - Os módulos comunicam-se usando RPC
    A comunicação é feita usando RPC.
  - O RPC foi criado para ser usado no desenvolvimento do NFS
  - Pode ser configurado para usar TCP ou UDP
    TCP possui confiabilidade e ordenação.
- Programas de usuário podem executar operações sobre arquivos locais ou remotos sem distinção.
  É transparente as operações.
- A integração é feita através do VFS
  O VFS está acima de todos os sistemas de arquivos e garante a transparência.

Arquitetura do NFS



O servidor NFS recebe a requisição e traduz para o VFS que busca a resposta no sistema de arquivos.

O cliente NFS entende os padrões porque o VFS garante isso.

Sistema de arquivos virtual (VFS)

- Manipulador de arquivo: identificador de arquivo do NFS.
  Identificador único de cada arquivo.
  - Contém informações para distinguir um arquivo

Identificador do sistema de arquivos	*i-node* do arquivo	número de geração do *i-node*

ID do sistema de arquivos: onde o arquivo está armazenado.
I-node do arquivo: estrutura que mantém os atributos do arquivo e endereços de bloco do arquivo.

Número de geração do I-node: Incrementado toda vez que o I-node for utilizado.

- O VFS mantém uma estrutura para cada tipo de sistema de arquivos montado e um v-node para cada arquivo aberto.
- Número de geração – incrementado a cada uso/reuso de um inode.
  Garante que se saiba se um I-node foi ou não utilizado, garantindo a unicidade do manipulador de arquivos.

O servidor - Características

- Implementa a política write-through (escrita direta)
  Toda requisição de alteração de dados no servidor, a alteração deve ser feita no disco antes de confirmar ao cliente.
  - Qualquer bloco modificado por uma escrita deve ser atualizado no disco antes de estar completo.
- É sem estado
  Não se mantém o estado das requisições anteriores. O endereço deve ser mandado a cada requisição.
  - Todas as requisições são independentes e completas.
- Independente de hardware e SO.
  Heterogeneidade.
- Recuperação rápida de falhas – principal razão para estar sem estado.
  é sem estado pois ao enviar uma requisição se houver falha, o cliente irá enviar a requisição várias vezes.

  No lado do servidor as falhas se tornariam mais complexas.
- Acesso transparente.
  Escrita e leitura como em arquivo local.
- A forma de acesso do UNIX é mantida no cliente.
- Desempenho "razoável".
  Os dados estão armazenados remotamente, então não se perde muito desempenho. Apenas a escrita se torna mais custosa por usar write-through, mas é um atraso tolerável pois não é muito realizada.

O protocolo NFS

- Utiliza RPC e XDR para representação de dados.
  RPC como mecanismos de invocação remota de

procedimentos. XDR pare representação e garantia de interoperabilidade.

- Definido como um conjunto de procedimentos remotos.
  Cada chamada é um procedimento remoto.
- Cada chamada de procedimento possui todas as informações necessárias para sua conclusão.
  O servidor não mantém estado.
- Em caso de falha do servidor, o cliente deve retransmitir a requisição até obter resposta.
  Questão da recuperação rápida de falhas.
- O cliente não tem como diferenciar se o servidor falhou e se recuperou ou se é uma lentidão.
  Não há manutenção do lado do servidor. Se o servidor perde conexão o cliente não consegue identificar.

A operação lookup()

- Retorna o manipulador do arquivo
- lookup(dirfh, nome) retorna (fh,attr)
  Recebe o endereço do diretório e um texto que se quer pesquisar. O resultado é o manipulador do arquivo e seus atributos.

  É útil quando se desenvolve aplicação e solicita abertura de arquivos.
  - Retorna em fh o manipulador e em attr os atributos do arquivo no diretório dirfh.

Ex:

```
fd = open("/etc/conf/passfile.conf")
```

• Resulta em:

```
lookup(rootfh, "etc") retorna (fh0, attr)
lookup(fh0, "conf") retorna (fh1, attr)
lookup(fh1, "passfile.conf") retorna (fh2, attr)
```

A operação lookup()

- Por que todos estes passos?
  É necessário o "file render" de cada um dos diretórios.
  - Qualquer parte de letc/conf/passfile.conf pode residir em sistemas de arquivos em diferentes servidores.

O cliente

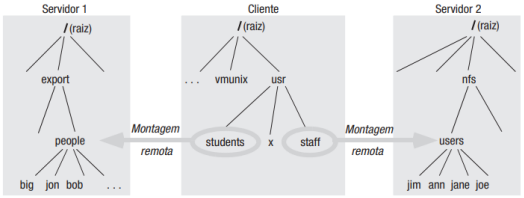
- Provê transparência no acesso.
  Para as aplicações o tratamento de arquivos locais ou remotos devem

ser feitos da mesma maneira.

- Trata todos os arquivos (locais e remotos) do mesmo modo.

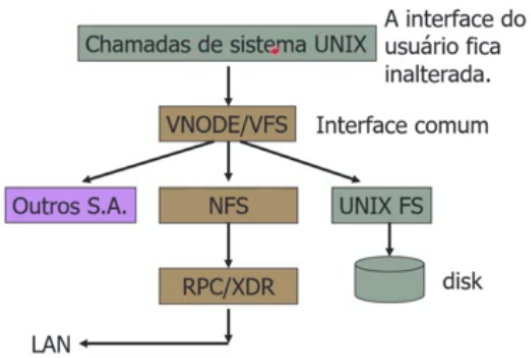
- Os nomes dos caminhos a serem montados são especificados em uma tabela de montagem (/etc/fstab)

- A subárvore remota aparece como parte da árvore de diretórios local.
  Para transparência, se associa um diretório remoto a um diretório local.



Nota: O sistema de arquivos montado em */usr/students* no cliente é, na verdade, a subárvore localizada em */export/people* no Servidor 1; o sistema de arquivos montado em */usr/staff* no cliente é, na verdade, a subárvore localizada em */nfs/users* no Servidor 2.

Cliente



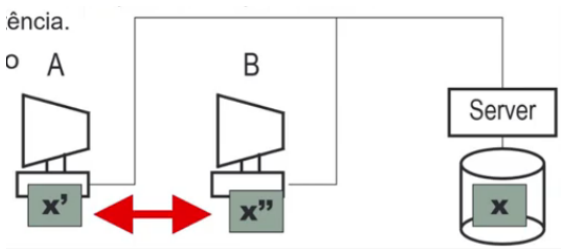
As chamadas de sistemas fazem requisições ao VFS, que ao fazer manipulação de arquivo cria um v-node (estrutura que mantém informações para operações necessárias de manipulações de arquivos locais e remotos). O VFS com v-node faz o acesso a sistemas de arquivos locais e ao NFS. No caso do NFS usa-se RPC para mecanismo de invocação remota e XDR para representação externa de dados.

Consistência

O NFS deve verificar a consistência principalmente se utilizar cache do lado do servidor.

- A necessidade de uma cache no cliente para garantir eficiência.
  Evita que o cliente busque sempre o arquivo no servidor.
  - Não é possível enviar todas as operações de read e write para o servidor.
  - Cache no cliente pode trazer problemas de consistência.

- Exemplo



Solução do NFS para consistência

- O cliente
    - Envia frequentemente os blocos modificados para o servidor.
    - Pergunta frequentemente ao servidor para revalidar os blocos que possui em cache.