TIAGO HENRIQUE CONTE

UM SISTEMA DE BOOT REMOTO COM SISTEMA DE ARQUIVOS DISTRIBUÍDO

(*versão pré-defesa, compilada em 4 de julho de 2023*)

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Luis Carlos Erpen de Bona.

CURITIBA PR

2023

RESUMO

O boot remoto é um procedimento no qual um computador realiza o boot através de um servidor de rede. Em um ambiente de boot remoto, o método mais comum utilizado para a obtenção de uma raiz de sistema operacional inicializado remotamente é através de um sistema de arquivos remoto, como o NFS (*Network Filesystem*). Entretanto, esse método apresenta alguns problemas como falta de confiabilidade e escalabilidade. Nesse contexto, este trabalho propõe um sistema de boot remoto onde a raiz é obtida por meio de um sistema de arquivos distribuído, o *CephFS*. Além disso, o trabalho busca avaliar se o sistema proposto consegue superar as limitações do boot através de um sistema de arquivos remoto. O sistema proposto mostrou ser melhor no que diz respeito a escalabilidade, confiabilidade e velocidade de acesso a metadados, mas não conseguiu equiparar-se ao desempenho do NFS em velocidade de acesso à arquivos.

Palavras-chave: Boot remoto. Ceph. CephFS. NFS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

DINF	Departamento de Informática
BCC	Bacheraldo em Ciência da Computação
UFPR	Universidade Federal do Paraná
C3SL	Centro de Computação Científica e Software Livre
NFS	*Network Filesystem*
POSIX	*Portable Operating System Interface*
OSD	*Object Storage Device* - Dispositivo de armazenamento de objetos
MDS	*Metadata Server* - Servidor de metadados
PXE	*Preboot Execution Environment*
DHCP	*Dynamic Host Configuration Protocol*
TFTP	*Trivial File Transfer Protocol*

SUMÁRIO

1 INTRODUÇÃO

A inicialização de sistemas operacionais pela rede é um método bem estabelecido para simplificar a administração do sistema em ambientes de computadores de médio a grande porte. O boot remoto do Linux para implantações de sistema em larga escala são amplamente adotados por muitos administradores de sistema (Avramov e Portolani, 2015, p. 43).

Além do carregamento do kernel, o processo de inicialização deve carregar uma raiz de sistema operacional. Para realizar essa operação, um dos métodos mais comuns é o uso do NFS (*Network Filesystem*), um sistema de arquivos remoto bem conhecido e popular. Entretanto, esse tipo de sistema de arquivos apresenta alguns problemas como a falta de confiabilidade e falta de escalabilidade. Tanto o problema da confiabilidade como o da escalabilidade são consequência de o serviço de sistema de arquivos remoto ser oferecido por um único servidor.

Em caso de falha ou indisponibilidade do servidor NFS, o cliente não será capaz de obter a raiz para a inicialização do sistema operacional, causando o problema da confiabilidade. Por outro lado, quando há vários clientes realizando tentativas de acesso ao NFS, o servidor tende ficar lento, o que gera o problema da escalabilidade.

Neste trabalho é proposto um sistema de boot remoto em que a obtenção da raiz do sistema operacional é feita através de um sistema de arquivos distribuído. A ideia é aumentar a confiabilidade e escalabilidade, já que um sistema de arquivos distribuído é servido por um *cluster* de servidores em vez de apenas uma máquina.

Para o desenvolvimento do trabalho, o sistema proposto utiliza o sistema de arquivos *CephFS* para a obtenção da raiz de sistema operacional. O *CephFS* é um sistema de arquivos proposto em 2006 e é largamente utilizado por administradores de sistema no mundo todo.

Este texto também avalia se o sistema proposto é realmente capaz de superar as limitações do boot remoto através de um sistema de arquivos remoto, comparando a implementação feita em *CephFS* com o método tradicional via NFS.

Embora o sistema proposto tenha se mostrado mais eficiente nos testes de escalabilidade, confiabilidade e leitura de metadados, ele se mostrou inferior ao NFS no quesito desempenho na obtenção de arquivos, sendo mais lento em operações de leitura de arquivos.

O presente trabalho está organizado da seguinte forma. O Capítulo 2 apresenta conceitos e definições de sistema de arquivos em Linux, além de introduzir o *OverlayFS*, um sistema de arquivos de união, o NFS, um sistema de arquivos remoto, e o *CephFS*, um sistema de arquivos distribuído. Já o Capítulo 3 explica o funcionamento do boot remoto, apresentando várias características deste e uma breve introdução ao ambiente de boot remoto do Departamento de Informática da UFPR. O Capítulo 4 apresenta o sistema proposto por este trabalho, os testes realizados e os resultados obtidos. As conclusões seguem no Capítulo 5.

2 SISTEMAS DE ARQUIVOS EM LINUX

Neste capítulo são introduzidos os principais conceitos relacionados a sistemas de arquivos, sistemas de arquivos remotos, sistemas de arquivos distribuídos.

Um arquivo é uma abstração fornecida pelo sistema operacional. O sistema operacional abstrai, das propriedades físicas de seus dispositivos de armazenamento, a definição de uma unidade lógica de armazenamento, o arquivo (Silberschatz, Galvin e Gagne, 2015).

Processos podem ler arquivos existentes e criar novos se necessário. Informações armazenadas em arquivos devem ser persistentes, isto é, não devem ser afetadas pela criação e término de um processo. Um arquivo deve desaparecer apenas quando o seu proprietário o remove explicitamente. Embora as operações para leitura e escrita de arquivos sejam as mais comuns, existem muitas outras (Tanenbaum e Bos, 2016, p. 182).

Como um usuário pode ser proprietário de milhões de arquivos, estes são organizados em estruturas hierárquicas denominadas diretórios, para facilitar sua localização e acesso pelos usuários. O sistema de arquivos consiste em duas partes distintas: uma coleção de arquivos, cada um deles armazenando dados relacionados, e uma estrutura de diretórios, que organiza e fornece informações sobre todos os arquivos no sistema (Silberschatz, Galvin e Gagne, 2015, p. 409).

Bovet e Cesati (2005) expõem que o design do sistema operacional UNIX, em que se baseia o Linux, é centrado em seu sistema de arquivos. Nesse sentido, do ponto de vista do usuário, o aspecto mais importante de um sistema de arquivos é como ele aparece, em outras palavras, o que constitui um arquivo, como os arquivos são nomeados e protegidos, quais operações são permitidas e assim por diante (Tanenbaum e Bos, 2016, p. 182).

Ainda no contexto de design do UNIX, o POSIX (*Portable Operating System Interface* — interface portátil para sistemas operacionais), foi criado com o objetivo de tornar possível escrever programas que pudessem ser executados em qualquer sistema UNIX. Ele define uma interface minimalista de chamadas de sistema à qual os sistemas operacionais em conformidade devem dar suporte (Tanenbaum e Bos, 2016).

Cada sistema de arquivos possui formas diferentes de realizar operações nos dispositivos, forçando a criação de uma interface comum para padronizar o acesso das aplicações a esses sistemas. O *Virtual Filesystem* (VFS), explorado na seção 2.1, foi a abordagem implementada no Linux Kernel para solução deste problema.

Existem diversos sistemas de arquivos com diferentes objetivos. Para este texto, são abordados os sistemas de arquivos remoto, sistemas de arquivos distribuídos e sistemas de arquivo de união.

Os sistemas de arquivos remoto têm a finalidade de permitir que usuários de computadores fisicamente distribuídos compartilhem dados e recursos de armazenamento usando um sistema de arquivos comum. Uma configuração típica para um sistema de arquivos remoto é uma coleção

de estações de trabalho e servidores conectados por uma rede local (Levy e Silberschatz, 1990). Pode-se dizer que um sistema de arquivos remoto fornece serviços de arquivo para clientes. A interface de cliente de um serviço de arquivo é formada por um conjunto de operações primitivas de arquivo, como criação, exclusão, leitura e gravação de um arquivo. O principal componente de hardware que um servidor de arquivos controla é um conjunto de dispositivos locais de armazenamento secundário (como discos rígidos ou de estado sólido), em que os arquivos são armazenados e a partir dos quais eles são recuperados de acordo com as solicitações dos clientes (Silberschatz, Galvin e Gagne, 2015). Para o elaboração deste trabalho, é abordado o sistema de arquivos remoto *Network Filesystem* (NFS), na seção 2.3.

Os sistemas de arquivos distribuídos são sistemas onde vários servidores são responsáveis por oferecer o serviço de arquivos para vários clientes. O fato de possuir mais de um servidor torna os sistemas de arquivos distribuídos sejam mais confiáveis, uma vez que possibilita a replicação de arquivos e tolerância a falhas. O sistema de arquivo distribuído escolhido para o desenvolvimento deste trabalho foi o *CephFS*, abordado na seção 2.4.

Os sistemas de arquivos de união permitem que dois ou mais sistemas de arquivos sejam unidos para a visão do usuário. No contexto deste trabalho, isso é útil para a montagem de uma raiz remota *read-only* que, na visão do usuário, se torna *read-write*. O sistema de arquivos de união *OverlayFS* é abordado na seção 2.2.

2.1 *VIRTUAL FILESYSTEM* (VFS)

A presença de múltiplos sistemas de arquivos levou à criação do conceito de *virtual filesystem*, cujo objetivo é integrar múltiplos sistemas de arquivos em uma estrutura única para ser acessado pelo sistema operacional (Tanenbaum e Bos, 2016, p. 204).

A fim de alcançar esse objetivo, o Linux Kernel implementa uma camada de abstração, a qual "funciona definindo as interfaces conceituais básicas e estruturas de dados que todos os sistemas de arquivos suportam" (Love, 2010, p. 262).

Para as aplicações as chamadas de sistema sempre serão as mesmas e o VFS fica responsável por traduzir essas chamadas de sistema em instruções específicas para o sistema de arquivo real (Tanenbaum e Bos, 2016, p. 204). A figura 2.1 mostra o funcionamento do VFS como um intermediário entre as aplicações e os sistemas de arquivos.

Todas as chamadas de sistemas relativas a arquivos são direcionadas ao sistema de arquivos virtual para processamento inicial. Essas chamadas, vindas de outros processos de usuários, são as chamadas POSIX padrão, como `open`, `read`, `write` e assim por diante. Desse modo, o VFS tem uma interface "superior" para os processos do usuário, a interface POSIX. O VFS também tem uma interface "inferior" para os sistemas de arquivos reais, que consiste em várias dúzias de chamadas de funções que os VFS podem fazer para cada sistema de arquivos a fim de realizar o trabalho. Assim, para criar um sistema de arquivos que funcione com o VFS, os

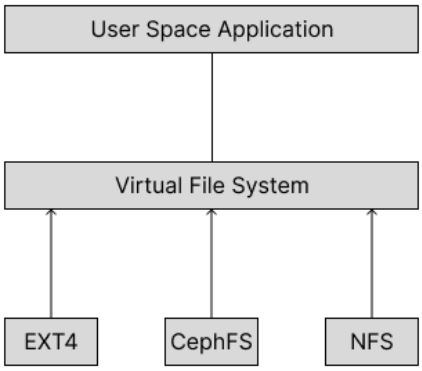

Figura 2.1: Representação simplificada do VFS.

projetistas do novo sistema de arquivos devem se certificar de que ele proporcione as chamadas de funções que o VFS exige (Tanenbaum e Bos, 2016, p. 204).

2.1.1 Montagem de Sistemas de Arquivos

Da mesma forma que um arquivo deve ser aberto antes de ser usado, um sistema de arquivos deve ser montado antes que possa ficar disponível para processos no sistema. (Silberschatz, Galvin e Gagne, 2015).

Para realizar a montagem, o sistema operacional recebe o nome do dispositivo e o ponto de montagem – localização dentro da estrutura de arquivos onde o sistema de arquivos será anexado. O Linux é capaz de avaliar o tipo do sistema de arquivos. Quando isso não ocorre é necessário especificar o tipo de sistema de arquivos do dispositivo ao realizar a montagem. Geralmente, o ponto de montagem é um diretório vazio na estrutura. Quando o diretório não está vazio o sistema oculta o conteúdo dele até que o sistema de arquivos montado na estrutura seja desmontado.

Em seguida, o sistema operacional verifica se o dispositivo contém um sistema de arquivos válido. Para concluir, o sistema operacional registra em sua estrutura de diretórios que um sistema de arquivos está montado no ponto de montagem especificado. Esse esquema habilita o sistema operacional a percorrer sua estrutura de diretórios, alternando-se entre os sistemas de arquivos, até mesmo entre sistemas de arquivos de tipos diferentes, quando apropriado (Silberschatz, Galvin e Gagne, 2015).

A figura 2.2 ilustra a montagem de um dispositivo com sistema de arquivos ext4 no diretório `/home` e um sistema de arquivos NFS no diretório `/remote`, através de conexão via rede.

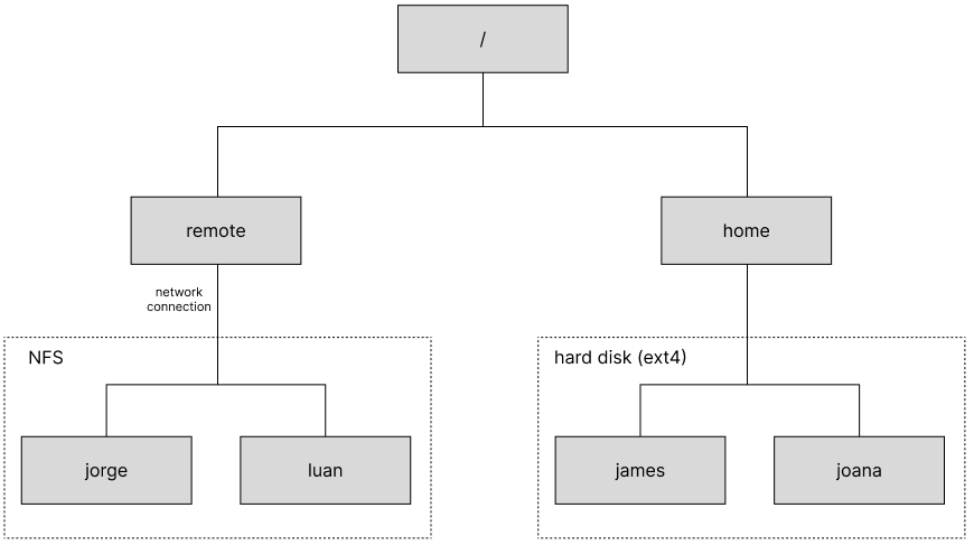

Figura 2.2: Exemplo de montagem de um sistema de arquivos NFS e ext4.

2.2 *OVERLAYFS*

Pendry e McKusick (1995) propuseram o modelo de *"union mount filesystem"* (em português, sistema de arquivos de união) que, ao contrário de uma montagem tradicional que oculta o conteúdo do diretório no qual está localizada, apresenta uma visão de fusão dos dois diretórios. Na visão do usuário, a união permite que todos os arquivos possam ser modificados, mesmo que apenas o diretório no topo tenha permissão para escrita.

Após um tempo o modelo ficou mais conhecido pelo nome *"overlay filesystem"* (em português, sistema de arquivos de sobreposição), principalmente devido à grande adesão ao *OverlayFS*, uma das implementações do modelo que foi incorporada ao Linux kernel em 2014.

A abordagem dessa implementação é "híbrida", porque os objetos que aparecem no sistema de arquivos nem sempre parecem pertencer a esse sistema de arquivos. Em muitos casos, um objeto acessado na união será indistinguível de acessar o objeto correspondente do sistema de arquivos original (Brown, 2020).

O *OverlayFS* combina dois sistemas de arquivos – um *upper* e um *lower*. Quando existe um nome em ambos os sistemas de arquivos, o objeto no sistema de arquivos *upper* é visível, enquanto o objeto no sistema de arquivos *lower* está oculto ou, no caso de diretórios, mesclado com o objeto em *upper*. A figura 2.3 ilustra o funcionamento da montagem do *OverlayFS*. Na figura, a camada *merged* é a vista pelo usuário, enquanto as camadas *upper* e *lower* são as camadas de cima e de baixo, respectivamente. Quando o arquivo `a.txt` existe nas duas camadas (*upper* e *lower*), o que é mostrado ao usuário é a camada mais acima.

O sistema de arquivos *lower* pode ser qualquer sistema de arquivos suportado pelo Linux e não precisa ter permissão de escrita, podendo até mesmo ser outro *OverlayFS*. O sistema de

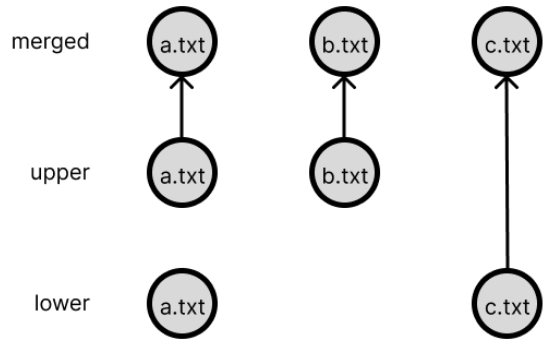

Figura 2.3: Fusão de dois *filesystems* no *OverlayFS*.

arquivos *upper*, quando usado com permissão de escrita, possui algumas restrições que tornam alguns sistema de arquivos incompatíveis.

Ao montar um *overlay*, é necessário especificar também um *workdir*, que deve ser um diretório vazio no mesmo sistema de arquivos que o *upper*.

Um exemplo de boa utilidade para o *OverlayFS* é a união de um diretório *read-only* montado de um servidor NFS como *lower* e um sistema de arquivos em memória (*ramfs*) como *upper*. Dessa forma, o usuário pode efetuar modificações nos arquivos da união sem afetar os arquivos originais.

2.3 *NETWORK FILESYSTEM* (NFS)

O NFS é um protocolo de sistema para permitir o compartilhamento de arquivos entre sistemas residentes em uma rede local. Cada servidor NFS exporta um ou mais dos seus diretórios para serem acessados por clientes remotos. Quando um diretório é disponibilizado, todos os seus subdiretórios também são, então na realidade árvores de diretórios inteiras são normalmente exportadas como uma unidade. Neste texto, recorre-se à versão 3 do NFS.

O *Network Filesystem* utiliza um protocolo *stateless*. Os parâmetros de cada chamada de procedimento contêm todas as informações necessárias para concluir a chamada e o servidor não mantém o registro de solicitações anteriores. Este tipo de protocolo evita a complexa recuperação de falhas. Se um cliente apenas reenviar solicitações até que uma resposta seja recebida, os dados nunca serão perdidos devido a uma falha no servidor. Na verdade, o cliente não consegue distinguir entre um servidor que travou e se recuperou e um servidor que está lento (Sandberg et al., 1985).

O NFS também utiliza o *Remote Procedure Call* (RPC), que tem por objetivo simplificar a definição, organização e implementação de serviços remotos.

A figura 2.4 expõe o funcionamento desse sistema de arquivos. No lado do cliente, uma *syscall* é enviada ao VFS, que é responsável por enviar a operação a interface do NFS. Essa

traduz a operação para uma RPC e envia para o servidor através da rede. O servidor recebe a RPC, realiza as rotinas do servidor e envia a operação para o VFS do servidor, que buscará as informações nos dispositivos de armazenamento locais.

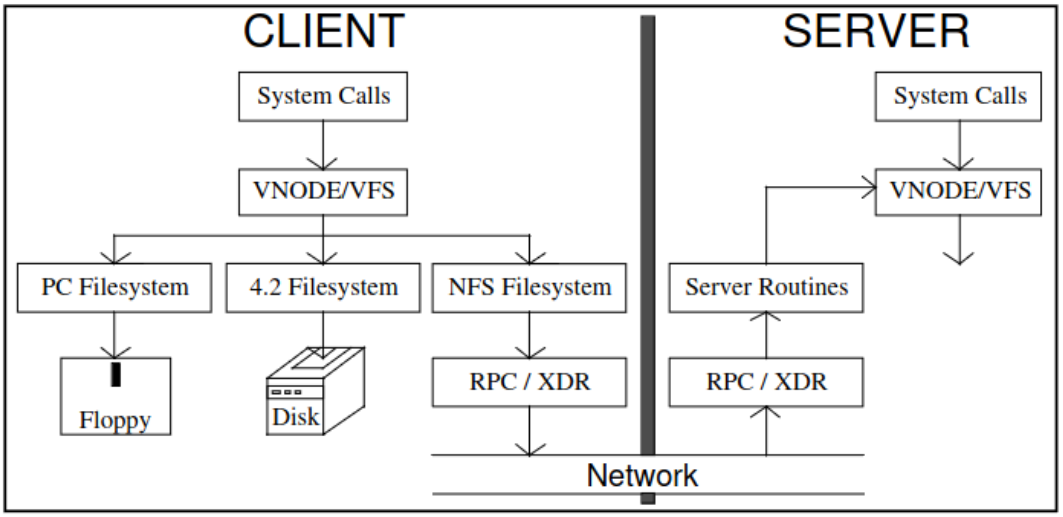

Figura 2.4: Visão geral do funcionamento do NFS (Sandberg et al., 1985).

Para que a comunicação entre cliente e servidor seja padronizada, o NFS define dois protocolos (Tanenbaum e Bos, 2016).

O primeiro protocolo NFS encarrega-se da montagem: um cliente envia um nome de caminho para um servidor e solicita permissão para montar aquele diretório em sua própria hierarquia. Se o nome do caminho é legal e o diretório especificado tiver sido exportado, o servidor retorna um manipulador de arquivo ao cliente, que contém campos identificando o tipo do sistema de arquivos, o disco e informações de segurança.

O segundo protocolo NFS serve para o acesso de diretório e arquivos. Clientes podem enviar mensagens para os servidores manipularem diretórios e ler e escrever arquivos. Eles podem também acessar atributos dos arquivos como modo do arquivo, tamanho e horário da última modificação.

A maioria das chamadas do sistema Linux tem o suporte do NFS, com exceção do `open` e `close`. Em vez disso, para ler um arquivo, um cliente envia ao servidor uma mensagem `lookup` contendo o nome do arquivo, com uma solicitação para examiná-lo e retornar um manipulador dele, que é uma estrutura que identifica o arquivo.

O NFS é muito popular e muito utilizado em diversas aplicações. Entretanto, os sistemas de arquivos remotos não preveem problemas como tolerância a falhas e redundância de arquivos.

2.4 CEPHFS

Proposto e desenvolvido por Weil et al. (2006), Ceph é um sistema de armazenamento distribuído que armazena dados como objetos dentro de *pools* de armazenamento lógico. O sistema maximiza a separação entre dados e metadados, substituindo as tabelas de alocação por

uma função de distribuição de dados pseudo-aleatória (CRUSH). O Ceph tem o objetivo de oferecer excelente desempenho, confiabilidade e escalabilidade.

De acordo com a documentação do Ceph (Ceph Team, 2016), o sistema é composto pelos seguintes principais componentes:

- **RADOS**: o *Reliable Autonomic Distributed Object Store* é o armazenamento de objetos que fornece um serviço escalável para objetos de tamanho variável. O armazenamento de objetos RADOS é o componente principal de um cluster do Ceph.

- ***Object Storage Devices* (OSD)**: são os dispositivos de armazenamento propriamente ditos, podendo ser discos rígidos ou discos de estado sólido. Cada OSD possui um *daemon* responsável por armazenar objetos em um sistema de arquivos local e fornecer acesso a eles pela rede, bem como trabalhar em conjunto para replicar dados, detectar e recuperar de falhas ou migrar dados quando OSDs ingressam ou saem do cluster.

- **Monitores**: são responsáveis por manter os mapas do estado do cluster, incluindo o mapa do monitor, o mapa do *manager*, o mapa de OSDs, o mapa de MDSs e o mapa CRUSH. São conjuntamente responsáveis por manter a coordenação do sistema distribuído e, para isso, utilizam de algoritmos clássicos de sistemas distribuídos.

- ***Manager***: é responsável por acompanhar as métricas e o estado atual do cluster, incluindo utilização de armazenamento, métricas de desempenho atuais e carga do sistema. Pelo menos dois gerentes são normalmente necessários para alta disponibilidade.

- ***Metadata Server* (MDS)**: armazena metadados para o Ceph File System (*CephFS*). Os MDSs permitem que os usuários do sistema de arquivos POSIX executem comandos básicos (como `ls`, `find`) sem sobrecarregar muito o cluster de armazenamento Ceph.

Um cluster de armazenamento Ceph requer pelo menos um monitor, pelo menos um *manager* e pelo menos tantos OSDs quantas forem as cópias de um objeto armazenado no cluster.

O *CephFS* é um sistema de arquivos compatível com POSIX construído sobre o armazenamento de objetos distribuídos do Ceph. O CephFS busca fornecer um armazenamento de arquivos de última geração, multiúso, altamente disponível e de alto desempenho para uma variedade de aplicativos, incluindo casos de uso tradicionais, como diretórios "*home*" compartilhados (Ceph Team, 2016).

O *CephFS* atinge esses objetivos através de novas técnicas utilizadas na arquitetura do sistema. Notavelmente, os metadados do arquivo são armazenados em um *pool* separado dos dados do arquivo e servidos por meio de um cluster redimensionável de MDSs, que podem ser dimensionados para suportar maiores cargas de trabalho. Os clientes do sistema de arquivos têm acesso direto ao RADOS para leitura e gravação de blocos de dados de arquivo. Ou seja, não há nenhum *gateway* ou intermediário mediando operações de entrada e saída de dados para clientes.

O acesso aos dados é coordenado por meio do cluster de MDS que serve como autoridade para o cache de metadados distribuído mantido cooperativamente por clientes e MDS. As alterações nos metadados são agregadas por cada MDS em uma série de gravações eficientes em um *journal* no RADOS; nenhum estado de metadados é armazenado localmente pelo MDS. Este modelo permite uma colaboração coerente e rápida entre clientes no contexto de um sistema de arquivos POSIX (Ceph Team, 2016).

Na figura 2.5 pode-se observar o funcionamento do *CephFS*. O cliente realiza: operações de metadados que são enviadas aos MDSs e, posteriormente, escritas nos *journal* e no *pool* de metadados; operações de dados, que são escritos diretamente no *pool* de dados.

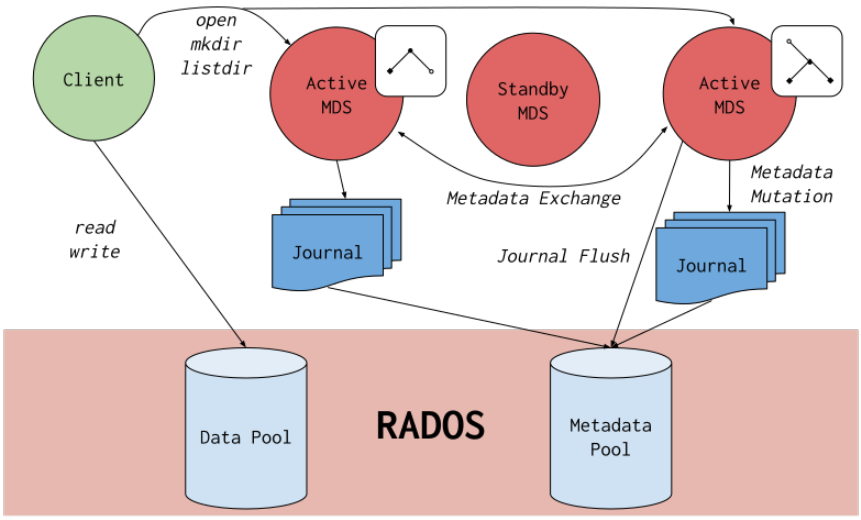

Figura 2.5: Arquitetura do *CephFS* (Ceph Team, 2016).

3 BOOT REMOTO

Existem diversas formas de proporcionar um ambiente de boot remoto com vários clientes, como o LTSP (*Linux Terminal Server Project*) e o *OpenSLX*. Entretanto, para o desenvolvimento deste trabalho foi utilizado o sistema do Departamento de Informática (DINF) da UFPR, que utiliza uma solução alternativa de boot remoto exitosamente há mais de 20 anos. Detalhes sobre a implementação no ambiente do DINF são descritos na seção 3.3.

Para que os clientes sejam capazes de inicializar o sistema operacional remotamente, suas interfaces de rede devem ser compatíveis com o padrão PXE (Preboot Execution Environment), que é explorado mais a fundo na seção 3.1.

O boot é o processo de inicialização de um computador com o carregamento do sistema operacional ao ligar a máquina. Logo que o computador é ligado, ele não tem um sistema operacional na memória, o que impede o hardware de realizar ações como, por exemplo, carregar um programa do disco.

A solução está na utilização de um programa pequeno e especial, chamado carregador de inicialização, *boot loader* ou *bootstrap*. Um *boot loader* não tem a completa funcionalidade de um sistema operacional, mas é especialmente construído para que seja capaz de carregar um outro programa a fim de permitir a iniciação do sistema operacional. Este outro programa é responsável por realizar a montagem da raiz do sistema operacional alvo, o esquema abordado para este trabalho é o *initramfs* (na seção 3.2).

O processo de inicialização pode ser considerado completo quando o computador está pronto para interagir com o usuário ou o sistema operacional é capaz de executar programas do sistema ou aplicativos.

3.1 PXE

Um *Preboot Execution Environment* (PXE) é uma interface cliente-servidor que permite que computadores em uma rede sejam inicializados a partir da imagem de boot obtida de um servidor, para clientes habilitados para PXE. Em outras palavras, permite que máquinas sejam bootadas pela rede.

A especificação do PXE (Intel Corporation, 1999) incorpora as seguintes tecnologias:

- Um protocolo uniforme para o cliente solicitar a alocação de um endereço de rede e, posteriormente, solicitar o download de um *Network Bootstrap Program* (NBP) de um servidor de inicialização de rede;
- Um conjunto de APIs disponível no ambiente de *firmware* pré-inicialização da máquina que constitui um conjunto consistente de serviços que podem ser empregados pelo NBP ou pelo BIOS;

- Um método padrão de iniciar o *firmware* de pré-inicialização para executar o protocolo PXE na máquina cliente.

O PXE geralmente faz uso dos protocolos TFTP e DHCP (usando *DHCP options* específicos para seu funcionamento). O TFTP (*Trivial File Transfer Protocol*) é um protocolo de transferência de arquivos projetado para ser o mais simples e rápido possível. Desse modo o protocolo é capaz de realizar apenas operações de `read` e `write` (Sollins, 1992).

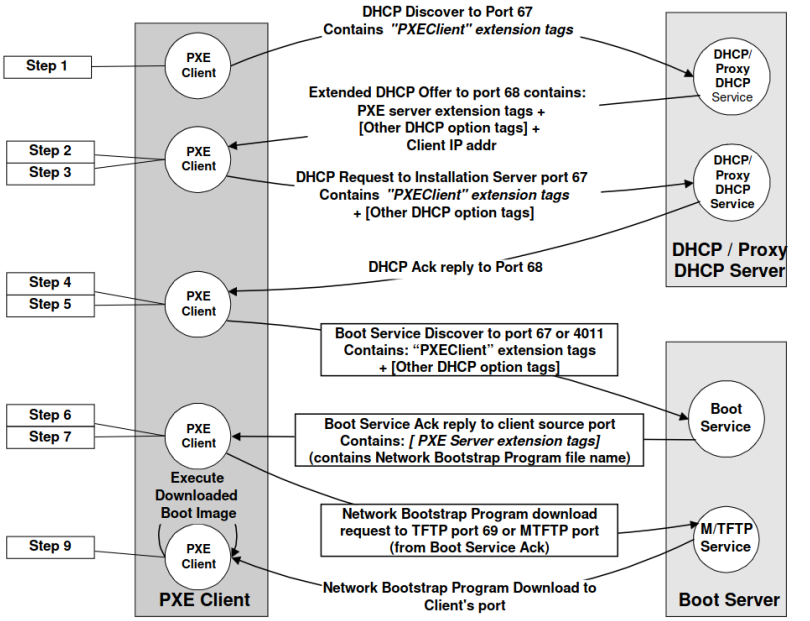

Figura 3.1: Boot com PXE (Intel Corporation, 1999).

A partir da figura 3.1, é possível identificar os passos do procedimento de boot via PXE:

- **Step 1**: O cliente envia uma requisição para o servidor DHCP, expressando sua intenção de boot PXE.
- **Step 2**: O servidor responde o cliente com um IP para ser usado pelo cliente e informações sobre o servidor de boot.
- **Step 3 e Step 4**: O cliente grava as informações recebidas e completa o protocolo DHCP enviando o endereço IP para o servidor e esperando um ACK.
- **Step 5**: O cliente seleciona e descobre o servidor de boot enviando uma mensagem via broadcast.
- **Step 6**: O servidor de boot responde o cliente com um pacote contendo: o nome do arquivo do NBP (*Network Bootstrap Program*), parâmetros de configuração TFTP e outras opções requeridas pelo NBP.
- **Step 7 e Step 8**: O cliente faz o *download* do arquivo executável do programa de bootstrap.

- **Step 9**: O cliente PXE inicia a execução do arquivo NBP baixado.

3.1.1 PXELINUX

PXELINUX é um *boot loader* para Linux capaz de inicializar a partir da rede via PXE. É usado em conjunto com um sistema compatível com PXE, que permite receber um programa de *bootstrap* pela rede local. Este programa carrega e configura um kernel do sistema operacional que coloca o usuário no controle do computador. Normalmente, o PXELINUX é usado para executar instalações do Linux a partir de um servidor de rede central ou para inicializar estações de trabalho sem disco (The Syslinux Project, 2009).

O PXELINUX é um exemplo de *network bootstrap program* obtido no passo 9 da figura 3.1. Ele é responsável por carregar o kernel e a imagem inicial do sistema de arquivos raiz (explorado na seção 3.2) na memória e, em seguida, iniciar o kernel, passando o endereço de memória da imagem. No final de sua sequência de inicialização, o kernel tenta determinar o formato da imagem a partir de seus primeiros blocos de dados, o que pode levar ao esquema *initrd* ou *initramfs*, descrito abaixo.

3.2 INITIAL RAMDISK

O *Initial Ramdisk* é um esquema para carregar um sistema de arquivos raiz temporário na memória, para ser usado como parte do processo de inicialização do Linux. O *initrd* e o *initramfs* são dois métodos diferentes e amplamente utilizados para este esquema. Para este trabalho, será abordado apenas o método *initramfs*.

No método *initramfs*, a imagem pode ser um arquivo `cpio` (geralmente compactado). O arquivo é descompactado pelo *kernel* em uma instância especial de um `tmpfs` que se torna o sistema de arquivos da raiz inicial (`rootfs`). Após a extração, o *kernel* verifica se o `rootfs` contém um arquivo `init` e, se houver, ele o executa como PID 1 (Landley, 2005).

O processo `init` é responsável por conduzir a inicialização até o sistema operacional final, incluindo localização e montagem do verdadeiro dispositivo raiz. Primeiramente, ele realiza o carregamento de qualquer módulo do *kernel* que ele irá precisar. Em caso de uma raiz fornecida por NFS, por exemplo, o processo deve ativar a interface de rede, adquirir um IP, extrair o nome do caminho NFS e o endereço do servidor e montar o diretório NFS. Após a montagem, o processo realiza o pivoteamento para a raiz real.

O *script* de inicialização `init` também encarrega-se dos parâmetros do kernel, que são *strings* de texto interpretadas pelo sistema para alterar comportamentos específicos e ativar ou desativar determinados recursos.

No Debian, a ferramenta *initramfs-tools* é responsável pela geração da imagem *initramfs* usada no boot. Essa ferramenta permite que em diferentes etapas do processo `init` seja possível a adição de *scripts* criados pelo usuário, capazes de executar diversas ações durante o boot.

3.3 DINF

A solução de boot remoto do Departamento de Informática (DINF) faz uso do PXE, TFTP e DHCP (Dynamic Host Configuration Protocol) para inicialização dos terminais e fornece as raízes via NFS.

Ao inicializar um computador conectado à rede, o servidor DHCP fornece à interface de rede do cliente um IP e o endereço do servidor PXE na rede (através de DHCP *options*). A interface entra em contato com o servidor PXE e carrega, via TFTP, o kernel e o arquivo de raiz temporário usado durante o processo de inicialização, que na implementação do DINF é o *initramfs*.

O processo `init`, invocado pelo *initramfs*, prepara o ambiente para a montagem da raiz do sistema operacional, interpretando os parâmetros do kernel, carregando os diversos módulos necessários para o boot e obtendo, via DHCP, o caminho da raiz no servidor NFS. Em seguida, é realizada a montagem da raiz no diretório `/root`, em modo *read-only*.

Durante a fase *init-bottom* do `init`, a última fase antes do pivoteamento para a raiz real, é realizada a chamada de um *script* personalizado (apêndice A). Esse *script* realiza a criação de um *overlay*: a raiz do sistema operacional (montada no diretório `/root`) como *lowerdir* e um sistema de arquivos em memória como *upperdir* e *workdir*. Ao final do boot, o usuário encontra uma raiz que podem ser modificada (*read-write*). Entretanto, as modificações não são permanentes, pois ficam armazenadas localmente na memória do cliente.

No ambiente do DINF, os serviços de servidor PXE, TFTP, DHCP e NFS estão localizados em uma única máquina, que possui as seguintes especificações:

- 2 x Processador Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz;
- 512 GB de memória RAM;
- 3 x 500 GB SSD (em raid 5);
- Conexão de rede de 40 Gb/s.

4 BOOT REMOTO COM CEPHFS

Atualmente, o método mais comum para obtenção da raiz nos ambientes de boot remoto é a partir de um servidor NFS. Entretanto, isso faz com que a infraestrutura de boot via PXE se torne dependente do funcionamento adequado desse serviço. Em caso de falha ou indisponibilidade do servidor, nenhum terminal seria capaz de realizar o boot devido a falta de conexão e, consequentemente, erro na obtenção de uma raiz.

A proposta deste trabalho é verificar se o fornecimento de raiz para terminais via sistema de arquivos distribuído consegue superar a limitação do NFS, em ambiente de boot remoto. Para o desenvolvimento desse trabalho, o sistema de arquivos distribuído utilizado foi o *CephFS*. A hipótese orientadora é que o *CephFS*, ao contrário do NFS, é um sistema distribuído capaz de atender os clientes de forma mais confiável, com alta disponibilidade e escalabilidade.

O *initramfs* suporta o boot remoto apenas com aquisição da raiz via NFS, ou seja, não possui suporte a qualquer sistema de arquivos distribuído. Para superar essa limitação e tornar o boot remoto compatível com o *CephFS*, o presente trabalho propõe um *script*, disponível no apêndice B, que foi construído com o objetivo de adicionar ao *initramfs* a compatibilidade com montagem de raiz fornecida via *CephFS*. O sistema proposto foi implementado com base no *script* de boot via NFS, disponível online, presente nativamente no *initramfs*.

Para que o *script* de montagem execute corretamente, é necessário adicioná-lo ao diretório `/etc/initramfs-tools/scripts`, incluir o módulo `ceph` ao arquivo de módulos `/etc/initramfs-tools/modules` e inserir a configuração de montagem em `/etc/initramfs-tools/initramfs.conf`. Além disso, o parâmetro `boot=ceph` deve ser incluído como parâmetro do kernel, forçando a invocação do *script* de montagem do *CephFS* pelo processo `init` .

A configuração de montagem envolve as variáveis: `CEPHROOT`, caminho do diretório que deve ser montado do servidor *CephFS*; `CEPHOPTS`, define as opções para montagem do sistema de arquivos; `CEPHSERVER`, indica os servidores de monitoramento do Ceph. Um exemplo de configuração pode ser encontrado no apêndice C.

Com todos os arquivos criados e configurações realizadas, é necessário gerar uma imagem atualizada do *initramfs* utilizando o comando `update-initramfs -u`. Após a geração, a imagem precisa ser transferida para o servidor responsável pelo fornecimento dela ao ambiente de boot remoto.

A primeira tarefa do *script* de montagem do *CephFS* é carregar o módulo do Ceph. Como é necessária uma conexão de rede para conectar-se aos servidores do *CephFS*, o *script* chama uma função implementada no *initramfs-tools* para adquirir uma conexão de rede. Dessa forma, o sistema já adquire via DHCP o caminho o qual deverá ser montado do *CephFS*. Em

seguida, o *script* interpreta as configurações definidas pelas variáveis apresentadas anteriormente e, se tudo estiver correto, monta o diretório do *CephFS* em `/root`.

4.1 EXPERIMENTOS

Uma série de experimentos foram realizados no ambiente do NFS e no ambiente do *CephFS*. O principal objetivo é analisar os resultados obtidos e comparar os dois métodos a fim de validar o *CephFS* como uma alternativa viável ao NFS. A comparação é feita nos quesitos de desempenho, disponibilidade e escalabilidade.

Os testes foram executados em um ambiente composto por máquinas virtuais fazendo o papel de clientes do sistema, com a seguinte especificação:

- Software de virtualização: qemu;
- CPU: 2 cores;
- 4 GB de memória RAM;
- iPXE boot.

A implementação iPXE é utilizada como interface PXE nas máquinas virtuais que realizam boot remoto. O iPXE é uma implementação de código aberto do carregador de inicialização PXE (iPXE Team, 2021).

Para os testes de boot, a *distro* utilizada foi a *Linux Mint Debian Edition 5* customizada para funcionamento no DINF. A imagem possui tamanho de 30 GB e 975 mil *inodes*. O *CephFS* cria dois objetos para cada arquivo ou diretório, um para dados e outro para metadados (Weil et al., 2006). Logo, é possível chegar a conclusão que a imagem de boot gera um total de 1,95 milhões de objetos no Ceph.

O *cluster* do Ceph usado nos experimentos é composto por 4 servidores de OSDs com a seguinte especificação:

- 2 x Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz;
- 512 GB de memória RAM;
- 12 x 8 TB SAS HDD;
- 1 x 2 TB SSD NVME.

Para os experimentos que fazem uso do NFS, foi utilizado o ambiente do Departamento de Informática da UFPR, apresentado na seção 3.3.

4.1.1 Desempenho

Para a avaliação de desempenho, foram realizados três experimentos nos ambientes de boot PXE com a raiz montada via NFS e via *CephFS*.

Para o primeiro experimento foi utilizado como parâmetro de comparação o tempo de boot em cada um dos métodos de montagem. Para aquisição desse dado, foi utilizada a ferramenta `systemd-analyze` e considerado apenas o tempo gasto em *userspace*. O teste para cada um dos métodos de montagem de raiz foi reproduzido 100 vezes. Os resultados podem ser visualizados na tabela 4.1.

	NFS	*CephFS*
Tempo médio	10,416	25,023
Tempo mínimo	9,365	9,219
Tempo máximo	11,590	57,298
Mediana	10,488	10,767
Desvio Padrão	0,430	18,382

Tabela 4.1: Tempo de boot, em segundos, com raiz em NFS e em *CephFS*.

A diferença de performance entre os métodos foi bem expressiva, com o tempo médio de boot no *CephFS* sendo, aproximadamente, 140% maior do que no NFS. Assim, é possível afirmar que o *CephFS* apresenta uma deficiência de performance na velocidade de acesso aos dados necessários para o boot quando comparado ao NFS.

Além disso, mesmo com o tempo mínimo de boot dos dois métodos sendo muito semelhante, a análise do desvio padrão aponta que a raiz em NFS gerou valores muito mais constantes nos tempos de boot, expondo a inconstância na performance apresentada pela raiz em *CephFS*.

O segundo experimento foi a comparação do tempo de execução para o comando `find / -exec stat {} +`. Esse comando passa por toda a árvore de diretórios da raiz e executa o comando `stat` em todos os arquivos. Em outras palavras, faz uma leitura dos metadados de todos os arquivos da raiz. Os testes foram executados 20 vezes para cada método de montagem. Os resultados são apresentados na tabela 4.2.

	NFS	*CephFS*
Tempo médio	4 min 20 s	1 min 55 s
Tempo mínimo	4 min 6 s	1 min 31 s
Tempo máximo	5 min 22 s	2 min 31 s
Mediana	4 min 16 s	1 min 54 s
Desvio Padrão	17 s	15 s

Tabela 4.2: Tempo de execução do comando *find* em metadados, com raiz em NFS e em *CephFS*.

A análise dos dados indica uma performance superior do *CephFS* em comparação ao NFS, visto que a execução do comando ocorre em 66% menos tempo na comparação entre os

tempos médios. É possível afirmar que o *CephFS* resolve de forma mais eficiente o acesso aos metadados de arquivos.

O terceiro experimento foi a comparação do tempo de execução para o comando `find /var -type f -exec dd if= of=/dev/null bs=4K count=1 \;`. Esse comando passa por toda a árvore do diretório `/var` e faz uma leitura dos primeiros 4KB de dados de todos os arquivos. Os testes foram executados 20 vezes para cada método de montagem. Os resultados são apresentados na tabela 4.3.

	NFS	*CephFS*
Tempo médio	49 s	3 min 58 s
Tempo mínimo	44 s	1 min 36 s
Tempo máximo	1 min	6 min 59 s
Mediana	49 s	3 min 15 s
Desvio Padrão	4 s	1 min 29 s

Tabela 4.3: Tempo de execução do comando *find* em dados, com raiz em NFS e em *CephFS*.

Novamente, o *CephFS* apresenta desempenho muito inferior ao NFS. Na comparação do tempo médio, a execução do comando `find` no *CephFS* aponta um tempo 485% maior do que na execução no NFS. Além disso, é possível perceber a inconstância na performance quando observado o desvio padrão dos resultados. Essa análise confirma a falta de performance do sistema de arquivos distribuído.

4.1.2 Disponibilidade

O teste de disponibilidade consiste em avaliar a continuidade do funcionamento do sistema em caso de falhas nos servidores de arquivos.

Para o NFS, a conexão com o servidor de arquivos foi interrompida durante o uso do cliente. A imagem congelou completamente, o que é estranho, já que grande parte dos arquivos necessários para a continuidade do funcionamento já deveriam estar alocados na memória da máquina cliente.

Para o *CephFS*, foi desativado um dos servidores que compõe o *cluster*. A imagem não foi afetada e seguiu funcionando normalmente.

Esse teste comprova a alta disponibilidade de um sistema de arquivos distribuído.

4.1.3 Escalabilidade

Este teste busca avaliar a capacidade dos sistemas NFS e *CephFS* em suportar uma grande quantidade de máquinas realizando tentativas de boot ao mesmo tempo. Para isso, foram utilizadas 50 máquinas virtuais efetuando o processo de boot simultaneamente.

É possível observar que tanto o NFS como o *CephFS* apresentam um aumento nos tempos de boot nos casos de aumento de carga. Entretanto, observa-se que o NFS obteve um aumento de 100% no tempo médio, enquanto no *CephFS* esse aumento foi de, aproximadamente,

	NFS		*CephFS*	
	sem sobrecarga	com sobrecarga	sem sobrecarga	com sobrecarga
Tempo médio	14,958	30,382	30,751	37,816
Tempo mínimo	13,552	25,605	13,438	16,471
Tempo máximo	18,487	32,727	61,933	46,166
Mediana	14,864	30,605	16,957	39,350
Desvio Padrão	0,818	1,254	18,656	6,255

Tabela 4.4: Tempo de boot, em segundos, sem e com sobrecarga em raiz em NFS e em *CephFS*.

23%. A partir disso, é possível concluir que o *CephFS* conseguiu resolver de forma mais eficaz a sobrecarga no acesso aos arquivos, o que já era esperado por se tratar de um sistema de arquivos distribuído.

4.2 ANÁLISE DOS RESULTADOS

Com base nos experimentos, é possível observar que o sistema proposto possui tanto vantagens como desvantagens em relação ao sistema tradicional utilizado em boot remoto.

Uma das desvantagens é a falta de performance no acesso à arquivos. O *CephFS* mostrou uma grande lentidão no acesso aos arquivos necessários para o boot, tornando o boot dos clientes mais lento quando comparado ao NFS. Além disso, o sistema proposto se mostrou inconstante nos tempos de boot, ocorrendo uma grande variação entre os tempos medidos durante o experimento.

Entretanto, o *CephFS* se mostrou muito mais eficiente no acesso aos metadados em relação ao NFS. O que já era esperado, visto que a arquitetura do *CephFS* busca principalmente a velocidade de acesso aos metadados, como abordado na seção 2.4.

Outra vantagem apresentada foi a estabilidade mantida pelo *CephFS* mesmo com a grande quantidade de clientes realizando solicitações ao mesmo tempo. Houve um aumento do tempo médio de boot quando comparado ao boot sem sobrecarga, porém o aumento foi inferior ao ocorrido no NFS nas mesmas condições.

Os problemas performáticos do *CephFS* podem ser explicados pela falta de customizações nas configurações do Ceph. O *cluster* utilizado para os experimentos poderia ser "tunado" com o objetivo de aumentar a velocidade de acesso aos dados do Ceph e, consequentemente, diminuir o tempo de boot dos clientes conectados.

Por ter uma arquitetura bem mais simples, o NFS consegue superar o *CephFS* em alguns objetivos, como mostrado nos resultados acima. Porém, fica evidente também a falta de confiabilidade e escalabilidade apresentada neste modelo.

5 CONCLUSÃO

Este trabalho propôs um sistema de boot remoto com obtenção de raiz via sistema de arquivos distribuído *CephFS*, cujo objetivo é superar as limitações do boot remoto com obtenção de raiz via sistema de arquivos remoto, como a inexistência da tolerância a falhas e falta de escalabilidade. Além disso, o sistema proposto foi avaliado e comparado com o método mais tradicional de obtenção de raiz em boot remoto, o NFS.

O *CephFS*, por ser um sistema de arquivos distribuído, se mostrou tolerante à falhas e, consequentemente, mais confiável do que o NFS. O sistema também apresentou-se mais escalável, performando melhor que o NFS em casos de alta demanda.

O sistema proposto se mostrou mais eficiente no acesso aos metadados dos arquivos, principalmente por ser o foco da arquitetura do Ceph. Entretanto, foi possível perceber uma baixa performance do *CephFS* no acesso aos arquivos em comparação ao NFS.

A arquitetura simples do sistema de arquivos remoto NFS torna esse sistema mais performático que o *CephFS* em algumas situações, principalmente em se tratando de acesso à dados.

Trabalhos futuros podem investigar mais a fundo a escalabilidade do *CephFS* em ambientes de alta carga de trabalho com grande números de clientes (Borges, Crosby e Boland, 2017) (Peters e van der Ster, 2021). Além disso, é possível realizar experimentos sobre a tunagem do *CephFS* especificamente para a aplicação em ambientes de boot remoto.

REFERÊNCIAS

Avramov, L. e Portolani, M. (2015). *The Policy Driven Data Center with ACI: Architecture, Concepts, and Methodology*. Networking Technology Series. Cisco Press.

Borges, G., Crosby, S. e Boland, L. (2017). Cephfs: a new generation storage platform for australian high energy physics. *Journal of Physics: Conference Series*, 898(6):062015.

Bovet, D. e Cesati, M. (2005). *Understanding the Linux Kernel*. O'Reilly Media.

Brown, N. (2020). Overlay filesystem. `https://docs.kernel.org/filesystems/overlayfs.html`. Acessado em 29/05/2023.

Ceph Team (2016). Ceph Documentation. `https://docs.ceph.com/en/latest/`.

Intel Corporation (1999). Preboot Execution Environment (PXE) Specification. `http://www.pix.net/software/pxeboot/archive/pxespec.pdf`.

iPXE Team (2021). iPXE – Open Source Network Boot Firmware. `https://ipxe.org/`.

Landley, R. (2005). ramfs, rootfs and initramfs. `https://www.kernel.org/doc/Documentation/filesystems/ramfs-rootfs-initramfs.txt`.

Levy, E. e Silberschatz, A. (1990). Distributed file systems: Concepts and examples. *ACM Computing Surveys*, 22(4):321–374. invited paper.

Love, R. (2010). *Linux Kernel Development*. Addison-Wesley.

Pendry, J.-S. e McKusick, M. K. (1995). Union Mounts in 4.4BSD-Lite. Em *USENIX 1995 Winter Conference Proceedings*, New Orleans, LA. USENIX Association.

Peters, A. J. e van der Ster, D. C. (2021). Evaluating cephfs performance vs. cost on high-density commodity disk servers. *Computing and Software for Big Science*, 5(1):25.

Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D. e Lyon, B. (1985). Design and implementation of the sun network file system. Em *USENIX 1985 Summer Conference Proceedings*, páginas 119–130, Portland, OR. USENIX Association.

Silberschatz, A., Galvin, P. e Gagne, G. (2015). *Fundamentos de sistemas operacionais*. Grupo Gen - LTC.

Sollins, D. K. R. (1992). The TFTP Protocol (Revision 2). RFC 1350.

Tanenbaum, A. e Bos, H. (2016). *Sistemas operacionais modernos*. Pearson.

The Syslinux Project (2009). PXELINUX. `https://repo.or.cz/syslinux.git/blob/HEAD:/doc/pxelinux.txt`.

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E. e Maltzahn, C. (2006). Ceph: A scalable, High-Performance distributed file system. Em *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 06)*, páginas 307–320, Seattle, WA. USENIX Association.

APÊNDICE A – SCRIPT DE OVERLAY

```
#!/bin/sh

PREREQ=""

prereqs() {
      echo "$PREREQ"
}

case $1 in
prereqs)
      prereqs
      exit 0
      ;;
esac

HOSTNAME=$(cat /proc/sys/kernel/hostname)

NEWROOT=/overlay
RW=/rw
RO=/ro

mkdir $NEWROOT
mkdir $RW
mkdir $RO
mount --move $rootmnt $RO

# Mounting RW branch
mount -n -o rw,nosuid,nodev,mode=0755 -t tmpfs tmpfs $RW

# The "workdir" needs to be an empty directory on the same
# filesystem as upperdir.
mkdir $RW/upper $RW/work

TARGET=$NEWROOT/.overlay

# Load overlay module
modprobe overlay

mount -n -t overlay overlay \
-o noatime,lowerdir=$RO,upperdir=$RW/upper,workdir=$RW/work $NEWROOT

# Mounting point for overlay moves
```

```
43 mkdir -p ${TARGET}${RW}
44 mkdir -p ${TARGET}${RO}
45
46 # Moving branches to the new root
47 mount --move $RW ${TARGET}${RW}
48 mount --move $RO ${TARGET}${RO}
49
50 echo $HOSTNAME > $NEWROOT/etc/hostname
51
52 # Moving the new root to right place
53 mount --move $NEWROOT ${rootmnt}
54
55 exit 0
```

APÊNDICE B – SCRIPT DE MONTAGEM DO *CEPHFS* NO BOOT

Script para montagem de raiz via *CephFS*.

```
# CEPH root mounting          -*- shell-script -*-

# parse ceph bootargs and mount ceph
ceph_mount_root_impl()
{
      configure_networking

      # get ceph root from dhcp
      if [ "x${CEPHROOT}" = "xauto" ]; then
            CEPHROOT=${ROOTPATH}
      fi

      if [ -z "${CEPHSERVER}" ]; then
            panic "CEPHSERVER is not set"
      fi

      if [ -z "${CEPHOPTS}" ]; then
            panic "CEPHOPTS is not set"
      else
            CEPHOPTS="-o ${CEPHOPTS}"
      fi

      ceph_premount

  mount -t ceph ${CEPHSERVER}:${CEPHROOT} ${rootmnt?} ${CEPHOPTS}
}

# ceph root mounting
ceph_mount_root()
{
      ceph_top

      modprobe ceph
      # For DHCP
      modprobe af_packet

      wait_for_udev 10

      # Default delay is around 180s
      delay=${ROOTDELAY:-180}

```

```
42        # loop until ceph mount succeeds
43        ceph_mount_root_impl
44        ceph_retry_count=0
45        while [ ${ceph_retry_count} -lt "${delay}" ] \
46              && ! chroot "${rootmnt}" test -x "${init}" ; do
47              [ "$quiet" != "y" ] && log_begin_msg "Retrying ceph mount"
48              sleep 1
49              ceph_mount_root_impl
50              ceph_retry_count=$(( ceph_retry_count + 1 ))
51              [ "$quiet" != "y" ] && log_end_msg
52        done
53  }
```

APÊNDICE C – EXEMPLO DE CONFIGURAÇÃO PARA BOOT VIA *CEPHFS*

A seguir é mostrado um exemplo de configuração de montagem para o *CephFS*. A configuração foi adicionada ao arquivo `/etc/iniramfs-tools/initramfs.conf` a fim de possibilitar o boot através do *script* apresentado no apêndice B.

```
1 CEPHROOT=auto
2 CEPHOPTS=ro,name=<clientename>,defaults,noatime,_netdev,secret=<secretkey>
3 CEPHSERVER=<mon_servers>
```

A variável `CEPHROOT` definida como `auto` estabelece que o caminho que deve ser montado será adquirido via DHCP. A variável `CEPHOPTS` define as opções para montagem do *CephFS*, outras opções podem ser consultadas na documentação oficial do sistema de arquivos. Já a variável `CEPHSERVER` indica os servidores de monitoramento do Ceph, que serão acionados para que o cliente possa realizar a montagem do *CephFS*.