UNIVERSIDADE FEDERAL DE SÃO CARLOS
Bacharelado em Ciência da Computação
Gabriel Olivato

INFRAESTRUTURA COMPUTACIONAL ALTAMENTE REPLICÁVEL E PORTÁVEL PARA PESQUISA EM CIÊNCIA DE DADOS UTILIZANDO OPENHPC

São Carlos, SP
2021

Gabriel Olivato

INFRAESTRUTURA COMPUTACIONAL ALTAMENTE REPLICÁVEL E PORTÁVEL PARA PESQUISA EM CIÊNCIA DE DADOS UTILIZANDO OPENHPC

Trabalho de conclusão do curso de graduação em ciência da computação submetido ao corpo docente do Departamento de Computação da Universidade Federal de São Carlos como parte dos requisitos necessários para obtenção do título de cientista da computação

Orientador: Prof. Dr. Paulo Matias

São Carlos, SP
2021

Gabriel Olivato

INFRAESTRUTURA COMPUTACIONAL ALTAMENTE REPLICÁVEL E PORTÁVEL PARA PESQUISA EM CIÊNCIA DE DADOS UTILIZANDO OPENHPC

Trabalho de conclusão do curso de graduação em ciência da computação submetido ao corpo docente do Departamento de Computação da Universidade Federal de São Carlos como parte dos requisitos necessários para obtenção do título de cientista da computação

Orientador: Prof. Dr. Paulo Matias

Aprovado em: ___/___/_____.

Prof. Dr. Paulo Matias
Orientador

Prof. Dr. Hélio Crestana Guardia
UFSCar - Universidade Federal de São Carlos

Prof. Dr. André de Freitas Smaira
UFSCar - Universidade Federal De São Carlos

São Carlos, SP

Este trabalho é dedicado aos meus pais Rafael e Daiane, por todo carinho e apoio que me fizeram alcançar essa conquista, aos meus avós, por todo o amor que sempre recebi e aos amigos que fiz durante essa jornada

AGRADECIMENTOS

Agradeço primeiramente aos meus pais Rafael e Daiane, por todo amor e apoio que recebi durante toda a minha vida. Graças ao incentivo e suporte, sempre me guiando no caminho correto, tive a oportunidade de seguir meu sonho que sempre foi ser um cientista da computação.

Aos meus avós Francisco e Eliza, que sempre cuidaram de mim, sempre me mimando, me apoiando e contribuindo para todas as minhas conquistas até aqui.

À minha namorada e amiga Esther Hoffmann, que foi a melhor companhia que eu poderia ter nos últimos anos.

Aos meus amigos de longa data, Gabriel Teston, Danilo Queiroz Barbosa, Carlos Lucchesi e Sahmira Bianchi, companheiros que sempre estiveram comigo.

Aos amigos que fiz na faculdade, especialmente Igor, Catito, Chermont, João, Sugaya e Nardy que foram parceiros em momentos de descontração e estudo.

Aos meus professores, que me ensinaram o que eu sei hoje, em especial Priscila Keli Frizzarin, Danilo Queiroz Barbosa, Mario César San Felice, Murilo Naldi e Paulo Matias.

Ao meu amigo Lucas Ribeiro, que me apoiou e me deu o suporte necessário para a conclusão deste trabalho.

Ao meu time de CTF, Shingeki No Chikungunya, com quem aprendi e dei muitas risadas.

Ao meu colega de equipe HPC, André Smaira, que sempre se mostrou prestativo e me ajudou neste estudo sempre que precisei.

Ao meu orientador, professor e amigo Paulo Matias, por me auxiliar e guiar no desenvolvimento deste estudo.

"A simplicidade é a conquista final.
Depois de ter tocado uma vasta quantidade de notas,
é a simplicidade que emerge
como a recompensa coroada da arte"
(Frédéric Chopin)

RESUMO

Computação de alto desempenho ou HPC (*High-performance computing*) refere-se ao uso de supercomputadores ou ao uso de múltiplos computadores em tarefas que exigem uma grande quantidade processamento. Uma infraestrutura de HPC é requisito para realizar pesquisas nas mais variadas áreas do conhecimento. Implantar e manter esse tipo de infraestrutura não é tarefa simples e por isso, em grandes centros de computação científica, existem grandes equipes responsáveis por essa tarefa. Este trabalho relata as lições aprendidas com a implantação de uma infraestrutura de HPC na Universidade Federal de São Carlos, que dispõe de uma equipe reduzida. O projeto OpenHPC, disponibilizado e mantido pela comunidade de software livre, auxilia a reduzir a complexidade dessa infraestrutura. No entanto, realizaram-se algumas adaptações no processo de instalação padrão do OpenHPC: (1) a correção de uma falha de segurança relacionada à forma como o provisionador de nós é configurado; e (2) a utilização do Ceph como alternativa de sistema de arquivos de rede com maior desempenho e confiabilidade que o NFS, porém de menor complexidade de operação que o Lustre. Em seguida, este estudo aborda a utilização de *containers* como forma de promover a reprodutibilidade e portabilidade de experimentos científicos. Embora tecnologias de *container* para ambientes de HPC, como o Singularity, sejam relativamente maduras, ainda não há uma abundância tão grande de componentes prontos para reutilização como em Kubernetes e outras plataformas baseadas em nuvem. Este trabalho colabora, portanto, com a implementação de um *container* Singularity para execução da plataforma Apache Spark, amplamente utilizada em pesquisas na área de ciência de dados, em um ambiente de HPC. Além disso, este trabalho propõe e documenta uma série de facilidades para o dia-a-dia de um grupo de pesquisa, por exemplo a notificação da conclusão de experimentos por meio de mensageiros instantaneos. Por fim, a infraestrutura completa é validada com a realização de alguns experimentos.

Palavras-chave: Computação de alto desempenho, OpenHPC, HPC, Infraestrutura computacional, Slurm

ABSTRACT

High-performance computing or HPC refers to the use of supercomputers or the use of multiple computers in tasks that require a large amount of processing. An HPC infrastructure is a requirement to carry out research in the most varied areas of knowledge. Deploying and maintaining this type of infrastructure is not a simple task and that is why, in large scientific computing centers, there are large teams responsible for this task. This work reports the lessons learned from the implementation of HPC infrastructure at the Federal University of São Carlos, which has a reduced staff. The OpenHPC project, made available and maintained by the free software community, helps to reduce the complexity of this infrastructure. However, there were some adaptations to the standard OpenHPC installation process: (1) the correction of a security hole related to the way the node provisioner is configured; and (2) the use of Ceph as an alternative network file system with greater performance and reliability than NFS, but less complex to operate than Lustre. Then, this study addresses the use of *containers* as a way to promote the reproducibility and portability of scientific experiments. While *container* technologies for HPC environments such as Singularity are relatively mature, there is still not as plentiful an abundance of ready-to-use components as in Kubernetes and other cloud-based platforms. Therefore, this work collaborates with the implementation and documentation of a *container* Singularity to run the Apache Spark platform, widely used in data science research, in an HPC environment. Furthermore, this work proposes and documents a series of facilities for the day-to-day activities of a research group, for example, notification of the completion of experiments through instant messengers. Finally, the complete infrastructure is validated by performing some experiments.

Keywords: High-performance computing, OpenHPC, HPC, Computational infrastructure, Slurm

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HPC	High-performance computing
SMS	System Management Server
bash	Bourne-Again SHell, um Unix shell e linguagem de programação
VM	Virtual Machine
sh	Interpretador do tipo shell
I/O	Input/Output
PBS	Portable Batch System
Unix	É um sistema operativo portável, multitarefa e multiutilizador originalmente criado por Ken Thompson, Dennis Ritchie.
GNU	Projeto era criar um sistema operacional parecidparte físicade o com o Unix
DNS	Domain Name Server
PAM	Pluggable Authentication Module
DKMS	Dynamic Kernel Module Support
RPM	Red Hat Package Manager
MPI	Message Passing Interface
IPMI	Intelligent Platform Management Interface
NTP	Network Time Protocol
IP	Internet Protocol
MAC	Media Access Control
E/S	Entrada e Saída
RDMA	Remote Direct Memory Access
GPU	Graphics Processing Unit
BOS	Base Operational System
CephFS	Ceph filesystem

MDS	Meta Data Server
LDAP	Lightweight Directory Access Protocol
DHCP	Dynamic Host Configuration Protocol
PXE	Pre BootExecution
BIOS	Basic Input/Output System
XML	Extensible Markup Language
XDR	Extended detection and response
CVE	Common Vulnerabilities and Exposures
POSIX	Portable Operating System Interface
ACPI	Advanced Configuration and Power Interface
FQDN	Fully Qualified Domain Name
QoS	Quality of Service
AUR	Arch User Repository

SUMÁRIO

1 INTRODUÇÃO

Computação de alto desempenho é utilizada em uma grande gama de sistemas, desde os nossos computadores pessoais até grandes sistemas de processamento paralelo (1). Para pesquisadores, *High Performance Computing* (HPC) tornou-se uma ferramenta essencial, pois a maior parte dos problemas pode ser simulado, clarificado ou experimentalmente testado utilizando simulações computacionais. Como os pesquisadores possuem pouco ou nenhum conhecimento em ciência da computação, são necessárias ferramentas para facilitar a execução destes experimentos (2).

Com isto em mente, criou-se o *OpenHPC*, um conjunto de ingredientes necessários para implantar e gerenciar um grupo de máquinas Linux para computação de alto desempenho, incluindo ferramentas de provisionamento, clientes de I/O (E/S ou Entrada e Saída), ferramentas de desenvolvimento e uma variedade de bibliotecas científicas. O OpenHPC é um Projeto Colaborativo da Linux Foundation (*Linux Foundation Collaborative Project*) cuja missão é ser a referência em componentes de software HPC open-source (código aberto), diminuindo a barreira de entrada para a implantação, avanço e uso de métodos e ferramentas modernas de HPC. Os pacotes e manuais do OpenHPC são mantidos e atualizados pela comunidade.

Um cluster (grupo de máquinas) HPC é um conjunto de computadores ou nós que trabalham juntos e podem ser vistos como um único sistema. Esses nós são geralmente conectados em uma rede local, na qual cada nó executa sua instância de sistema operacional, e possui capacidade de armazenamento e poder de processamento (3).

Figura 1 – Identidade visual da HPC-nee, criada para motivação interna da equipe

Fonte: Gerada aleatoriamente pela Waifu Labs

Este trabalho apresenta as lições aprendidas com a implementação do OpenHPC em máquinas na UFSCar e apresenta testes realizados utilizando Apache Spark para verificar se o escalonamento e distribuição está sendo feito de forma correta.

O Cluster UFSCar recebeu um apelido — *HPC-nee*, nome que será usado neste trabalho. Criou-se também uma identidade visual (Figura 1) para o cluster.

1.1 Motivação

Durante um tempo, a UFSCar disponibilizou à comunidade o acesso a um conjunto de máquinas no modelo de *cloud*, em que os pesquisadores podiam alocar máquinas virtuais em hardware compartilhado. No entanto, esse modelo mostrou-se pouco adequado à realidade da universidade, uma vez que a demanda por recursos era maior que a quantidade de hardware disponível. A utilização desses recursos nem sempre era realizada de forma consciente, levando à alocação de muitas máquinas virtuais ociosas que nunca eram desligadas, gerando uma sobrecarga de trabalho à pequena equipe de operação para manter a infraestrutura funcionando. Além disso, experimentos que envolviam medidas de desempenho ficavam pouco reprodutíveis, já que havia *overcommitment*, ou seja, a quantidade de recursos virtuais alocados superava a de recursos físicos reais.

Assim, percebeu-se que seria necessário trabalhar com um modelo de fila para a alocação de recursos para os pesquisadores. O uso de filas é muito comum em infraestruturas de HPC, sendo que gerenciadores de fila como o Slurm e o PBS são muito anteriores à proposta do termo *cloud computing*. No entanto, os ambientes de fila tradicionais apresentavam um grande entrave quando comparados à flexibilidade da *cloud*: os pesquisadores precisavam pedir para o administrador instalar os softwares que desejavam utilizar. Felizmente, a introdução de tecnologias de *container* como o Singularity tornou esses ambientes praticamente tão flexíveis quanto as tecnologias de *cloud*, permitindo que os pesquisadores instalem e gerenciem seus próprios ambientes.

Desta forma, o Núcleo de Computação Científica da UFSCar tomou a decisão de migrar os recursos computacionais disponíveis para uma tecnologia baseada no modelo de filas. Com isto em mente, criou-se uma equipe temporária composta por: André de Freitas Smaira, Paulo Matias, Gabriel Olivato e Pedro Henrique Lara Campos, para a instalação do conjunto de softwares fornecidos pelo OpenHPC nestas máquinas, além da realização de testes com a plataforma e da adequação de cargas de trabalho comuns para a nova infraestrutura.

1.2 Objetivos

Este trabalho tem como objetivo apresentar um procedimento de instalação genérico de um cluster utilizando componentes de software disponibilizados pelo OpenHPC, junto com problemas encontrados e soluções para estes. Também serão abordadas as decisões

tomadas na escolha dos softwares instalados. O processo de criar um cluster HPC pode ser extremamente entediante (4).

Além disso serão abordados a criação de um container utilizando Singularity, a execução de um experimento que recebe como entrada um conjunto de dados que será obtido de um serviço externo, e o envio do resultado do experimento para outro serviço externo.

2 CONCEITOS FUNDAMENTAIS

2.1 GNU/Linux

GNU/Linux é um sistema operacional baseado em UNIX, que possue um núcleo Linux e um conjunto de ferramentas e aplicações para a utilização do sistema operacional.

2.1.1 Estrutura do sistema

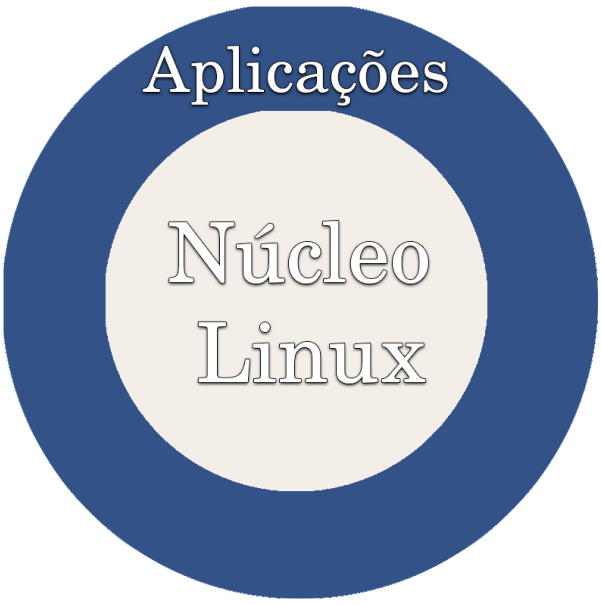
Um sistema Unix é composto por duas partes:

a) Um núcleo ou kernel, que é a parte que interage com o hardware (parte física de um computador). Ele é responsável por escalonar processos, gerenciar memória e controlar o acesso a arquivos. O acesso ao kernel é feito através de syscalls (chamadas de sistema), que são funções disponibilizadas para o uso. O kernel é executado em espaço de memória privilegiado.

b) Aplicações, interagem com o usuário para a utilização do sistema operacional. Dentre elas, destacam-se:

- Shell, recebe comando do usuário para executar outras aplicações
- Conjunto de bibliotecas para a linguagem C (libc)

Figura 2 – Estrutura do sistema Unix



Fonte: Autor

Alguns comandos básicos, que são vastamente utilizados, podem ser vistos na Tabela 1.

Tabela 1 – Comandos básicos do sistema GNU/Linux

Comando	Descrição
dig	Faz buscas em servidores DNS
rsync	Transfere arquivos
scp	Copia de forma segura para um sistema remoto
ssh	Executa um shell em um sistema remoto
diff	Compara dois arquivos, linha por linha
cat	Lista o conteúdo de um arquivo
chmod	Altera os modos de acesso de um arquivo
chown	Altera a posse de um arquivo
ls	Lista arquivos no diretório corrente
mv	Move arquivos
cp	Copia arquivos
rm	Remove arquivos
vi	Abre um de texto
mkdir	Cria diretórios

2.1.2 Estrutura de diretórios

A árvore de diretórios é um mapa onde os arquivos são alocados. Os diretórios na raíz mais comuns são (5):

a) **/** - A própria raíz do sistema de arquivos. Todos os diretórios e arquivos são encontrados a partir da raíz

b) ***/bin*** - Diretório com aplicações executáveis do sistema que podem ser usados pelos usuários

c) ***/boot*** - Diretório com arquivos para a inicialização do sistema

d) ***/dev*** - Diretório com arquivos especiais que são pontos de acesso aos controladores de dispositivos no computador

e) ***/etc*** - Diretório com arquivos de configuração do computador

f) ***/home*** - Diretório com arquivos dos usuários

g) ***/lib*** - Diretório com bibliotecas compartilhadas pelo sistema e módulos de kernel

h) ***/mnt*** - Diretório para a montagem de outros sistemas de arquivos.

i) ***/proc*** - Pseudo sistema de arquivos que contém informações dos processos correntes

j) ***/root*** - Diretório do usuário *root*

k) ***/sbin*** - Diretório com aplicações utilizadas pelo super usuário para administrar o sistema

l) ***/tmp*** - Diretório com arquivos temporários, normalmente criados por aplicações

m) ***/usr*** - Diretório com a maior parte das aplicações instaladas no sistema local

n) ***/var*** - Diretório com arquivos que são gravados com frequência por aplicações do sistema

2.1.3 Distribuições Linux

Uma distribuição Linux é um sistema operacional criado a partir de uma coleção de softwares com o uso do kernel Linux e um gerenciador de pacotes. Existem mais de 600 distribuições Linux, e a maioria está em desenvolvimento ativo (6).

2.1.4 fstab

O arquivo **/etc/fstab** pode ser utilizado para definir como as partições do disco, dispositivos de armazenamento e arquivos de sistemas remotos devem ser montados no sistema.

2.1.5 PAM module

PAM (acrônimo para o inglês *Pluggable Authentication Modules*) são bibliotecas que o administrador do sistema pode utilizar para definir como aplicações autenticam usuários.

2.1.6 dkms

DKMS (Dynamic Kernel Module Support) é um programa/framework que permite gerar módulos do kernel do Linux cujos fontes residam fora da árvore de fontes. O conceito é poder ter os módulos automaticamente reconstruídos quando uma nova versão do kernel é instalada.

2.1.7 chroot

chroot é uma operação que altera o diretório raiz aparente para o processo atual de execução e seus filhos. Um programa que é executado em tal ambiente modificado não consegue acessar os arquivos e comandos fora dessa árvore de diretórios. Isso é chamado de um prisão chroot (ou chroot jail) (7).

Alterar a raiz geralmente é um procedimento comum para realizar a manutenção em sistemas nos quais a inicialização ou a autenticação não são mais possíveis. Exemplos comuns são:

a) Reinstalação do gerenciador de boot.

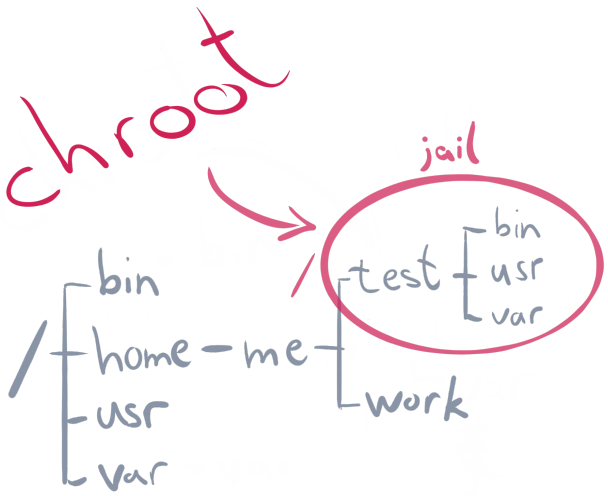
b) Reconstrução da imagem de initramfs.

c) Atualizar ou fazer downgrade de pacotes.

d) Redefinir uma senha esquecida.

Um exemplo para o entendimento prático do chroot é: ao realizarmos chroot em um sistema operacional e instalarmos um programa, o programa será instalado no sistema que está especificado pelo chroot e não no sistema base.

Figura 3 – Representação visual da execução de um chroot, note a repetição dos diretórios /bin, /usr, /var



Fonte: Suse Communities

2.2 Gerenciadores de pacotes

Sistemas GNU/Linux geralmente possuem gerenciadores de pacotes: programas que podem instalar, atualizar e remover pacotes. Geralmente cada distribuição Linux possui seu próprio gerenciador de pacotes. Entre eles, destacam-se:

a) **pacman** - utilizado em distribuições baseadas Arch Linux

b) **dnf** - utilizado em distribuições que utilizam pacotes RPM

c) **apk** - utilizado em distribuições baseadas em Alpine Linux

d) **apt** - utilizado em distribuções baseadas em Debian

2.3 Virtualização e *containers*

Existem três tipos principais de virtualização de servidores (8):

a) Virtualização de sistemas operacionais (geralmente chamados de *containers*)

b) Emulação de Hardware

c) Paravirtualização

O kernel Linux suporta a existência de várias instâncias isoladas de espaços de usuário (*user spaces*). Essas instâncias são chamadas de containers, partições ou *virtualization engines*. Uma aplicação executando no sistema operacional tem acesso a todos os recursos,

como dispositivos conectados e sistema de arquivos daquele computador. Já aplicações executando dentro de containers só podem ver o conteúdo e dispositivos designados ao container. Um container não é uma máquina virtual.

Figura 4 – Representação de um container

Fonte: Autor

2.4 Imagem de sistema operacional

No contexto de arquivos e programas, uma "imagem", seja uma ISO ou outra imagem de mídia, é simplesmente um arquivo que pode ser usado como uma cópia idêntica da mídia original. Este arquivo não contém apenas arquivos de dados individuais, mas também contém informações de trilha e setor e organiza todas essas informações em um sistema de arquivos, como uma mídia de disco. Os arquivos de imagem, ao contrário dos arquivos normais, geralmente não são abertos diretamente em programas do espaço de usuário; em vez disso, eles são montados.

Pode-se realizar o download de um sistema operacional, instalar programas, configurá-lo e gerar uma imagem. Ao instalarmos esta imagem em uma outra máquina, todo o processo de instalar programas e realizar configurações já estará concluído.

2.5 MPI

Message Passing Interface (MPI) é um padrão para comunicação de dados em computação paralela. Existem várias modalidades de computação paralela e dependendo do problema que se está tentando resolver, pode ser necessário passar informações entre os vários processadores ou nodos de um cluster. O MPI oferece um protocolo para essa tarefa.

2.6 Apache Spark

O Apache Spark, ou apenas Spark, é uma ferramenta para Big Data que tem o objetivo de processar grandes conjuntos de dados de forma paralela e distribuída. Ela estende o modelo de programação MapReduce popularizado pelo Apache Hadoop, facilitando bastante o desenvolvimento de aplicações de processamento de grandes volumes de dados. Além do modelo de programação estendido, o Spark também apresenta uma performance muito superior ao Hadoop, chegando em alguns casos a apresentar uma performance quase 100 vezes maior. (9)

2.7 Scala

Scala (Scalable language) é uma linguagem de programação de propósito geral, multiparadigma, projetada para expressar padrões de programação comuns de uma forma concisa, elegante e *type-safe*. Ela incorpora recursos de linguagens orientadas a objetos e funcionais. Geralmente é utilizada juntamente com Spark por conta de seu suporte ao Spark e seu paradigma funcional.

2.8 bash

O Bash é um interpretador e uma linguagem de programação de alto nível. É compatível por configuração com as normas POSIX, de forma que os scripts Bash podem ser executados em diversos sistemas "tipo unix"; e quando invocado com o nome de "sh", transforma-se automaticamente em seu predecessor (sh).

2.9 Computação distribuída

Sistema distribuído é uma porção de computadores, todos conectados em rede, coordenados por um ou mais máquinas administradoras e que utilizam softwares que permitam o compartilhamento de seus recursos, como memória, processamento e hardware para realizar tarefas, como quebrar um código, criptografar, descobrir a melhor solução de um problema em matemática ou para propósito geral.

3 TRABALHOS RELACIONADOS

O documento criado pela comunidade do OpenHPC serviu como diretriz deste trabalho. Ele fornece um passo a passo de como criar um cluster OpenHPC, com todos os componentes e recursos necessários (10).

Em (11), os autores apresentam uma maneira de utilizar OpenHPC em máquinas virtuais com OpenStack, também seguindo as diretrizes do manual original.

Em (12), os autores apresentam uma introdução ao uso de um sistema OpenHPC que utiliza o sistema de armazenamento de arquivos Lustre.

Em (4), os autores mostram uma visão geral sobre OpenHPC e apresenta uma abordagem em alto nível de como seria uma instalação de OpenHPC.

Já em (13), os autores apresentam uma versão mais rápida de instalação de OpenHPC com scripts previamente criados utilizando Ansible.

4 IMPLEMENTAÇÃO DO MODELO OPENHPC

Utilizou-se como base o manual de instalação fornecido pelo própria comunidade do OpenHPC (10). A instalação do sistema OpenHPC é semelhante à construção de um castelo feito de LEGO™, são dezenas de pequenas peças que se juntam e criam o todo. Analogamente, no OpenHPC são dezenas de softwares que realizam uma tarefa específica e que juntos criam todo o ambiente HPC.

4.1 Configurações iniciais

Antes de iniciar o processo de instalação, definiram-se algumas variáveis, que podem ser vistas na Tabela 2, e que serão utilizadas ao longo do processo, por meio de substituição de variáveis (`$variável`). Informações como *MAC address*, *ip address*, etc devem ser coletadas antes do início do processo de instalação. Pode-se também, ao invés de definí-las antes realizar a substituição da variável no momento do uso da mesma.

Tabela 2 – Variáveis necessárias para a instalação e suas descrições

Variável	**Breve descrição**
sms_name	Hostname do nó principal
sms_ip	IP do nó principal
sms_eth_internal	Interface de rede do SMS
internal_netmask	Máscara de rede utilizada na rede interna do cluster
ntp_server	Server NTP[1] utilizado para a sincronização do tempo
bmc_username, bmc_password	Credenciais BMC[2] para a utilização pelo IPMI[3]
IPMI_PASSWORD	Senha utilizada pelo IPMI
provision_wait	Tempo adicional para provisionar os nós compute
domain_name	Nome do domínio local, utilizado pelo xCAT
iso_path	Caminho para a imagem ISO, utilizada pelo xCAT
ohpc_repo_dir , epel_repo_dir	Caminho local para o web roots, utilizada pelo xCAT
enable_mpi_defaults	Habilitar a comunicação utilizando MPI
slurm_node_config	Configuração padrão do slurm para nós especificados
num_computes	Número total de nós compute
compute_regex	Padrão de regex que identifica os computes
c_name[*]	Array que recebe o nome dos computes
c_ip[*]	Array que recebe o IP dos computes
c_mac[*]	Array que recebe o endereço MAC dos computes
c_bmc[*]	Array que recebe o endereço BMC dos computes
kargs	Argumentos para serem passados para o kernel inicial

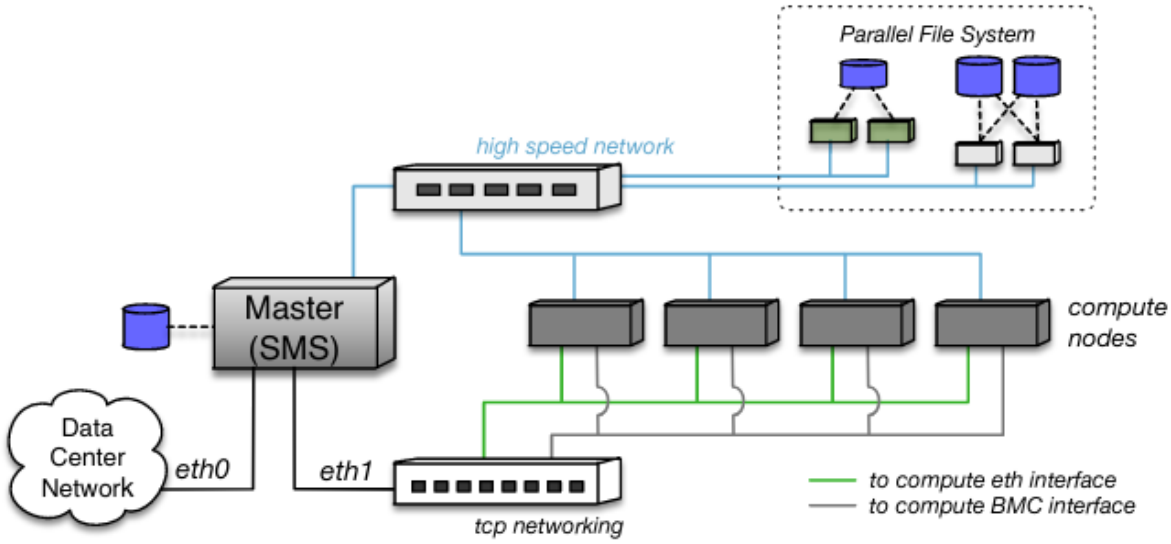
Pode-se definir os valores destas variáveis em um arquivo qualquer e carregá-las utilizando

```
1 [sms]$ source arquivo
```

4.2 Recursos disponíveis

As máquinas que são usadas como nós da HPC-nee serão referidas neste trabalho com um prefixo *c* (compute) e um número. Teremos as máquinas c1, c2, c3 e c4 que não possuem GPU (Unidade de processamento gráfico) e possuem especificações hardware idênticas e a máquina c11 que possui uma GPU da fabricante Nvidia. A foto a seguir representa a organização das máquinas na HPC-nee. Temos também a máquina SMS, o nó mestre da HPC-nee.

Figura 5 – Representação visual da organização das máquinas no cluster



Fonte: OpenHPC, a Linux Foundation Collaborative Project

4.2.1 SMS

Inicia-se o processo instalando um sistema operacional base para a máquina SMS que será usada para gerenciar os computes[4]. Escolheu-se o sistema CentOS8, como indicado no manual do OpenHPC, que pode ser obtido no repositório da UFSCar em <http://mirror.ufscar.br/centos/8/isos/x86_64/CentOS-8.3.2011-x86_64-boot.iso>. Após a instalação do sistema operacional base ou BOS (Base Operational System).
Adiciona-se então uma entrada de DNS que identifica a SMS com[5]:

```
1 [sms]$ echo ${sms_ip} ${sms_name} >> /etc/hosts
```

Para instalar os pacotes, utiliza-se o repositório de programas do OpenHPC.

```
1 [sms]$ dnf install http://repos.openhpc.community/OpenHPC/2/
    CentOS_8/x86_64/ohpc-release-2-1.el8.x86_64.rpm
```

[4] Não será abordado a configuração de rede realizada na HPC-nee, pois contém informações sensíveis e varia de instalação para instalação

[5] Note que toda vez que um comando é executado, é indicado entre colchetes onde ele deve ser executado

Pode-se agora adicionar os componentes desejados para OpenHPC no nó mestre para que provisione, gerencie recursos e serviços de toda a HPC-nee.

4.3 Softwares para administração da HPC-nee

Para administração da HPC-nee precisa-se de uma ferramenta projetada para gerenciar, instalar, monitorar e ver eventos dos computes. De acordo com o manual do OpenHPC destacam-se duas ferramentas: *warewulf3* e *xCAT*.

4.3.1 warewulf3

De início, warewulf3 foi escolhido pela equipe como primeiro candidato devido à sua simplicidade, mas como o software parou de ser atualizado em 2019 e existem vulnerabilidades conhecidas, decidiu-se que era uma péssima ideia utilizá-lo.

4.3.2 xCAT e a base do OpenHPC na SMS

O programa xCAT oferece gerenciamento completo de clouds, clusters, HPC, renderfarms e qualquer próxima buzzword que surja (14). Este pacote permite que o administrador da HPC-nee possa descobrir máquinas ligadas na rede, realizar execução de manutenção remota, provisionar sistemas operacionais em máquinas físicas ou virtuais, provisionar máquinas stateful e stateless, instalar e configurar aplicações de usuário.

Utiliza-se o xCAT majoritariamente para criar uma imagem de um sistema operacional e enviá-lo para os nós compute descobertos em rede, iniciá-los e reiniciá-los.

Para usá-lo é necessário habilitar o repositório do mesmo.

```
1 [sms]$ dnf -y install yum-utils
2 [sms]$ wget -P /etc/yum.repos.d https://xcat.org/files/xcat/
    repos/yum/latest/xcat-core/xcat-core.repo
```

O xCAT possui inúmeras dependencias que são necessárias para a sua instalação que estão localizadas em múltiplos repositórios públicos. Devemos habilitá-los então para a utilização.

```
1 [sms]$ dnf -y install centos-release-stream
2 [sms]$ wget -P /etc/yum.repos.d http://xcat.org/files/xcat/
    repos/yum/devel/xcat-dep/rh8/x86_64/xcat-dep.repo
```

Além dos pacotes do OpenHPC e xCAT, a SMS requer acesso aos repositórios de sistemas operacionais padrão. Para o CentOS8 os requisitos são acesso aos repositórios do BaseOS, Appstream, Extras, PowerTools e EPEL cujos *mirrors* estão disponíveis gratuitamente em:

a) CentOS-8 Base 8.2.2004 (e.g. <http://mirror.centos.org/centos-8/8/>)

b) EPEL 8 (e.g. <http://download.fedoraproject.org/pub/epel/8/>)

O repositório EPEL será habilitado automaticamente após a instalação do pacote ***ohpc-release***. O repositório ***Extras*** é habilitado por padrão no CentOS8, mas o repositório ***PowerTools*** é desabilitado. pode-se habilitá-lo então utilizando:

```
[sms]$ dnf install dnf-plugins-core
[sms]$ dnf config-manager --set-enabled PowerTools
```

Com os repositórios do OpenHPC e do xCAT habilitados pode-se efetivamente iniciar a adição de componentes desejados na SMS. Instala-se então os pacotes base do OpenHPC e o xCAT.

```
[sms]$ dnf -y install ohpc-base
[sms]$ dnf -y install xCAT
[sms]$ . /etc/profile.d/xcat.sh
```

Quando o xCAT é instalado ele desabilita o firewall, reatíva-se então com:

```
[sms]$ systemctl enable firewalld
[sms]$ systemctl start firewalld
```

Ainda não está finalizada a instalação do xCAT. Ela será retomada nas próximas seções.

4.3.3 Serviços de NTP

Serviços de NTP (Network Time Protocol), são utilizados em sistemas HPC pois dependem de relógios sincronizados em todos os nós do sistema. Pode-se realizar essa sincronização utilizando um protocolo de sincronição de relógios por meio da internet chamado NTP. Para habilitar esse serviço na SMS, e permitir que o servidor de NTP especificado sirva a HPC-nee, utilizamos o seguinte:

```
[sms]$ systemctl enable chronyd.service
[sms]$ echo "server ${ntp_server}" >> /etc/chrony.conf
[sms]$ echo "allow all" >> /etc/chrony.conf
[sms]$ systemctl restart chronyd
```

4.3.4 Slurm

Slurm é um sistema de escalonamento de trabalhos amplamente utilizado em ambientes HPC para gerenciamento de recursos distribuídos. Ao rodar um experimento, envia-se o script de execução de experimento para o Slurm, que se encarrega de escolher a(s) máquina(s) que possui(em) recursos disponíveis. A posição do experimento na fila pode ser acompanhada por um utilitário oferecido chamado **squeue**. Apresentamos a seguir uma lista de comandos que o Slurm oferece e que são amplamente utilizados:

a) **sacct**: exibir dados dos usuários para todos os trabalhos e etapas do trabalho no banco de dados Slurm.

b) **sacctmgr**: exibir e modificar as informações da conta Slurm

c) **sbatch**: exibir e modificar as informações da conta Slurm

d) **scancel**: cancelar um trabalho ou etapa de trabalho ou sinalizar um trabalho em execução ou etapa de trabalho

e) **scontrol**: exibir (e modificar quando permitido) o status das entidades Slurm. As entidades incluem: trabalhos, etapas de trabalho, nós, partições, reservas, etc.

f) **sinfo**: exibir informações resumidas da partição do nó (fila)

Instala-se o Slurm, copia-se as configurações padrões e identifica-se o gerenciador de recursos no SMS com:

```
1 [sms]$ dnf -y install ophpc-slurm-server
2 [sms]$ cp /etc/slurm/slurm.conf.ohpc /etc/slurm/slurm.conf
3 [sms]$ perl -pi e "s/ControlMachine=.*/ControlMachine=${
    sms_name}/" /etc/slurm/slurm.conf
```

Ressaltamos que isso é apenas a instalação do Slurm. Ele será habilitado e utilizado em outra seção.

4.4 Finalizando a configuração do xCAT

Neste ponto tem-se todos os pacotes necessários para utilizar o xCAT no SMS. Adiciona-se então suporte para provisionamento local utilizando uma segunda interface de rede interna, como pode ser visto na Figura 5, e registra-se essa interface de rede interna com xCAT:

```
1 [sms]$ ip link set dev ${sms_eth_internal} up
2 [sms]$ ip address add ${sms_ip}/${internal_netmask} broadcast
     + dev ${sms_eth_internal}

4 [sms]$ chdef -t site dhcpinterfaces="xcatmn|${
    sms_eth_internal}"
```

4.5 Criando uma imagem para os computes

Com os serviços de provisionamento habilitados, o próximo passo é definir e customizar a imagem de sistema que será usada nos computes.

4.5.1 Utilizando uma imagem do CentOS 8

Para instalá-lo, realiza-se download da imagem do CentOS 8 do repositorio da UFSCar e copia-se a imagem para o dirétório de instalação do xCAT[6]:

[6] copycds é um utilitário do xCAT

```
[sms]$ wget http://mirror.ufscar.br/centos/8/isos/x86_64/
   CentOS-8.3.2011-x86_64-boot.iso
[sms]$ copycds CentOS-8.3.2011-x86_64-boot.iso
```

Lista-se as imagens disponíveis utilizando um comando do pacote xCAT.
Para verificar as imagens disponíveis:

```
[sms]$ lsdef -t osimage
```

Cria-se então uma imagem *stateless* (inicializável via rede) para os nós compute e utiliza-se o programa *genimage* para inicializar uma instalação baseada em **chroot**.
Para realizar este processo adiciona-se a localização do **chroot**, em uma variável de ambiente, para a imagem que será usada nos computes e constroe-se a imagem **chroot**[7] inicial:

```
[sms]$ export CHROOT=/install/netboot/centos8/x86_64/compute/
   rootimg/
[sms]$ genimage centos8-x86_64-netboot-compute
```

Com a imagem gerada pode-se instalar pacotes via **chroot**.

4.5.2 Instalando pacotes na imagem dos computes

O comando *genimage* é usado para que ele construa uma configuração mínima do CentOS, a partir desta imagem instala-se pacotes adicionais, como drivers e serviços para gerenciar recursos nos computes. Para instalar os pacotes nas imagens, habilita-se o repositório ***base***.

```
[sms]$ yum-config-manager --installroot=$CHROOT --enable base
[sms]$ cp /etc/yum.repos.d/OpenHPC.repo $CHROOT/etc/yum.repos
   .d
[sms]$ cp /etc/yum.repos.d/epel.repo $CHROOT/etc/yum.repos.d
```

Depois instala-se um meta pacote base do OpenHPC na imagem dos computes e desabilitamos seu firewall:

```
[sms]$ dnf -y --installroot=$CHROOT install ohpc-base-compute
[sms]$ chroot $CHROOT systemctl disable firewalld
```

É importante copiar os arquivos de credenciais para a imagem, garantindo assim UID e GIDS consistentes para o Slurm na instalação

```
[sms]$ cp /etc/passwd /etc/group $CHROOT/etc
```

[7] Toda vez que utiliza-se a variável $CHROOT, estamos modificando a imagem que será usada em algum nó compute

Adicionamos os pacotes para o nó compute suportar o Slurm, registra-se o servidor Slurm nos computes, instala-se NTP para a sincronização do horário, driver de kernel e módulos para o ambiente de usuário.

```
[sms]$ dnf -y --installroot=$CHROOT install ohpc-slurm-client
[sms]$ echo SLURMD_OPTIONS="--conf-server ${sms_ip}" >
   $CHROOT/etc/sysconfig/slurmd

[sms]$ dnf -y --installroot=$CHROOT install chrony
[sms]$ echo "server ${sms_ip}" >> $CHROOT/etc/chrony.conf

[sms]$ dnf -y --installroot=$CHROOT install kernel
[sms]$ genimage centos8-x86_64-netboot-compute -k `uname -r`

[sms]$ dnf-config-manager --installroot=$CHROOT --enable
   baseos
[sms]$ dnf -y --installroot=$CHROOT install --enablerepo=
   powertools lmod-ohpc
```

Após esses passos, já pode-se instalar os pacotes necessários na imagem dos *computes*.

4.6 Atualizando a imagem nos computes

Toda vez que se instala algum pacote novo, é necessario empacotar esta imagem e atualizá-la em todos os nós da HPC-nee. Pode-se visualizar este fluxo na Figura 6.
Para isso utilizamos o seguinte comando :

```
  [sms]$ packimage nome-da-imagem
```

Para empacotar as imagens mais rápido instala-se o pacote *pigz* e então quando executarmos *packimage* desta vez utilizaremos com um parâmetro adicional.
O comando completo que será usado para empacotar a imagem e enviar aos nós é:

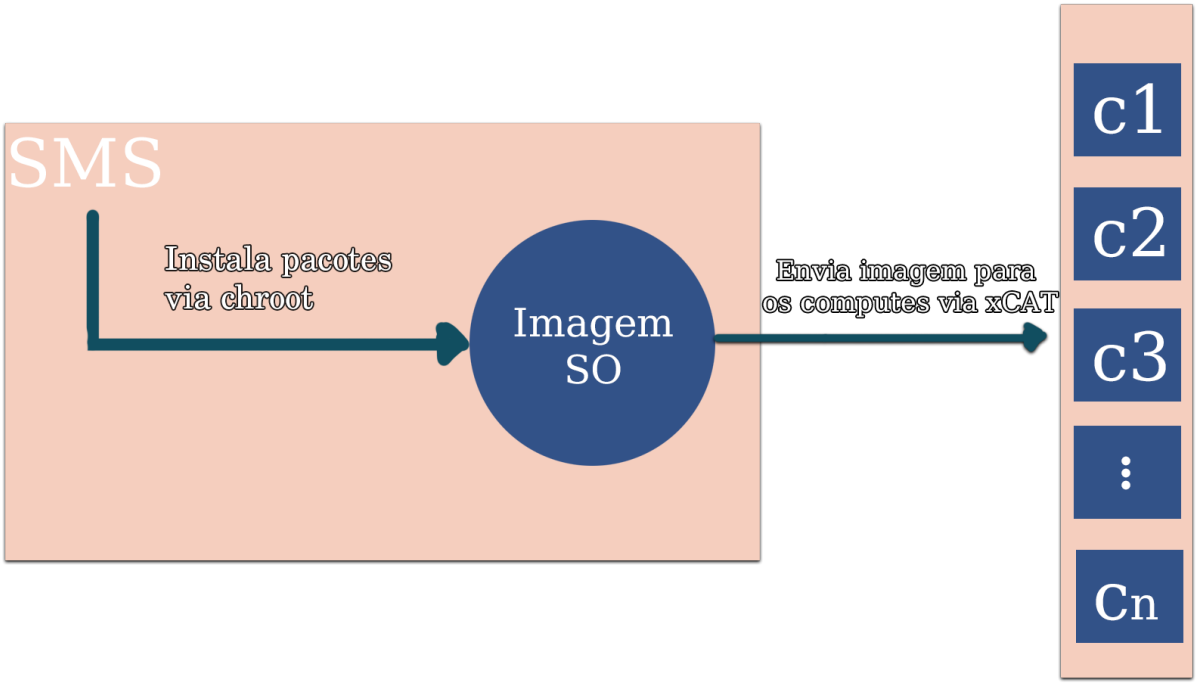
```
[sms]$ packimage -c pigz --nosyncfiles centos8.2-x86_64-
   netboot-compute
```

A opção –***nosyncfiles*** é necessária pois, sem este parâmetro, arquivos sensíveis como as chaves do Munge (sistema de autenticação do Slurm) e o */etc/shadow* são incluídos na imagem. Como a imagem pode ser baixada por qualquer nó via HTTP o que é, inclusive, um requisito para a inicialização via rede funcionar, os arquivos ficam expostos para qualquer usuário do cluster. Um usuário malicioso poderia se aproveitar disso para obter acesso *root*, colocando em risco a segurança da HPC-nee. Note que a sincronia dos arquivos que não são inclusos na imagem, a princípio, é segura. Isso porque a sincronização é iniciada pelo SMS, e não pelo nó, e porque ela ocorre utilizando o protocolo SSH.

Após enviar a imagem para os nós, pode-se reiniciá-las com o seguinte comando:

```
[sms]$ rpower compute reset
```

Figura 6 – Representação visual do fluxo do empacotamento de uma imagem e o envio desta para os nós compute



Fonte: Autor

4.6.1 Atualizando o kernel na imagem

Sempre que o kernel da imagem dos computes for atualizado, é necessário configurar o xCAT para utilizar o novo kernel no boot. Abaixo pode-se ver um exemplo de atualização de kernel. Nota-se que é necessário definir a variável `$version`, que recebe uma versão válida de kernel. O nome da imagem gerada com *genimage* é totalmente arbitrário, mas como a imagem gerada é de um sistema CentOS, usou-se um certo padrão.

```
[sms]$ cd /install/centos8.3/x86_64/Packages
[sms]$ wget http://mirror.ufscar.br/centos/8/os/x86_64/
   Packages/kernel-${versao}.rpm
[sms]$ genimage centos8.3-x86_64-netboot-compute -k ${versao}
```

Precisa-se ainda refazer o **$CHROOT/etc/fstab** com os mounts do Ceph (seção 4.7), habilitar novamente os repositórios base do CentOS e empacotar a nova imagem com *genimage*.

```
[sms]$ yum-config-manager --installroot=$CHROOT --enable base
[sms]$ packimage -c pigz --nosyncfiles centos8.3-x86_64-
   netboot-compute
```

4.6.2 Aumentar limites de memória

Altera-se o arquivo ***/etc/security/limits.conf*** para aumentar os limites padrão de memória utilizados tanto para a SMS quando para os nós compute.

```
[sms]$ perl -pi -e 's/# End of file/\* soft memlock unlimited
    \n$&/s' /etc/security/limits.conf
[sms]$ perl -pi -e 's/# End of file/\* hard memlock unlimited
    \n$&/s' /etc/security/limits.conf

[sms]$ perl -pi -e 's/# End of file/\* soft memlock unlimited
    \n$&/s' $CHROOT/etc/security/limits.conf
[sms]$ perl -pi -e 's/# End of file/\* hard memlock unlimited
    \n$&/s' $CHROOT/etc/security/limits.conf
```

4.6.3 Adicionando Node Health Check

Uma tarefa essencial em um cluster é saber o status atual dos nós, se eles estão "saudáveis" e funcionais ou "não saudáveis" e precisam de alguma intervenção manual. Nós podem ser marcados como "não operacionais" ou "offline", assim nenhum trabalho é alocado para aquele nó. Para realizar isto instala-se o *NHC* (Node Health Check) tanto na SMS quanto nos nós.

```
[sms]$ dnf -y install nhc-ohpc
[sms]$ dnf -y --installroot=$CHROOT install nhc-ohpc
```

Informa-se ao Slurm pra utilizar o NHC como o programa para checagem da saúde do nó a cada 5 minutos.

```
[sms]$ echo "HealthCheckProgram=/usr/sbin/nhc" >> /etc/slurm/
    slurm.conf
[sms]$ echo "HealthCheckInterval=300" >> /etc/slurm/slurm.
    conf
```

4.7 Sistema de arquivos compartilhados, utilizando Ceph

Para facilidade de uso decidiu-se adicionar um sistema de arquivos distribuídos para compartilhar o diretório ***/home*** do usuário corrente entre a SMS e os nós compute. Ou seja, para o usuário, todo arquivo criado no diretório ***/home/usuario_atual***[8] na máquina SMS, pode ser utilizado pelo usuário em qualquer nó compute. Analogamente, qualquer arquivo criado em algum nó compute neste diretório pode ser encontrado na

[8] usuario_atual se refere ao usuário atualmente utilizando seu diretório em /home

SMS e em qualquer outro nó compute, por isso sistema de arquivos compartilhado. O atual estado da arte em sistemas de arquivos compartilhados é o CephFS.

Ceph File System ou CephFS, é um sistema de arquivos construído em cima do sistemas de armazenamento distribuido de objetos RADOS(15). CephFS provê o estado da arte em armazenamento de arquivos, possui alta disponibilidade e boa performance para uma grande variedade de aplicações (16).

) CephFS consegue alcançar estes objetivos por conta de escolhas arquiteturais. Os metadados dos arquivos são armazenados em um *pool RADOS* separado dos dados do arquivo e servidos por meio de um cluster redimensionável de Servidores de Metadados, ou MDS, que pode ser escalonado para suportar cargas de trabalho de metadados de maior rendimento. Os clientes do sistema de arquivos têm acesso direto ao RADOS para ler e gravar blocos de dados de arquivos. Por esse motivo, as cargas de trabalho podem ser escalonadas linearmente com o tamanho do armazenamento de objeto RADOS subjacente; ou seja, não há *gateway* ou *broker* mediando E/S de dados para clientes.

CephFS foi escolhido para o compartilhamento de arquivos no cluster por conta de sua robustez, facilidade de operação e baixa necessidade de intervenções manuais. Trata-se de uma solução muito mais fácil de instalar e operar que um sistema de arquivos Lustre. O desempenho do CephFS é inferior ao do Lustre. No entanto, a UFSCar não dispõe atualmente de hardware compatível com RDMA (*Remote Direct Memory Access*), de forma que não seria possível configurar o Lustre para obter o máximo desempenho possível. Por fim, por se tratar de um sistema distribuído, a disponibilidade e o desempenho de um sistema de arquivos CephFS são superiores aos do NFS, solução geralmente adotada em cenários nos quais não se deseja a complexidade do Lustre.

Como a UFSCar já possuía um cluster Ceph, não será abordada a instalação, apenas a utilização de um CephFS já pré existente, que aqui será chamado de *storage*.

Adiciona-se então o repositório do CephFS em ***$CHROOT/etc/yum.repos.d/ceph_stable.repo*** para instalar nos nós compute e também em ***/etc/yum.repos.d/ceph_stable.repo*** para instalar na SMS.

```
1  [ceph_stable]
2  baseurl = http://download.ceph.com/rpm-octopus/el7/$basearch
3  gpgcheck = 1
4  gpgkey = https://download.ceph.com/keys/release.asc
5  name = Ceph Stable $basearch repo
6  priority = 2
7
8  [ceph_stable_noarch]
9  baseurl = http://download.ceph.com/rpm-octopus/el7/noarch
10 gpgcheck = 1
11 gpgkey = https://download.ceph.com/keys/release.asc
```

```
12 name = Ceph Stable noarch repo
13 priority = 2
```

Com o repositório adicionado, instala-se o Ceph

```
1     [sms]$ dnf -y --installroot=$CHROOT install ceph-common
2     [sms]$ dnf -y install ceph-common
```

Entra-se então em um dos nós do *storage* (e.g. s1), monta-se o CephFS temporariamente em */mnt* e cria-se um diretório para o OpenHPC.

```
1     [s1]$ mount -t ceph s1:6789:/ /mnt -o name=admin,secret=$
         (ceph-authtool -p /etc/ceph/ceph.client.admin.keyring)
2     [s1]$ mkdir -p /mnt/openhpc/
3     [s1]$ umount /mnt
```

Cria-se um usuário para a HPC-nee, este comando gera uma chave que será usada para a autenticação dos nós e da SMS no CephFS.

```
1     [s1]$ ceph fs authorize cephfs client.openhpc /openhpc rw
```

Como o CentOS utiliza ***systemd***, as definições no arquivo ***fstab*** são automaticamente convertidas em unidades de montagem durante a inicialização, a configuração vai executar ***fsck*** e montar os sistemas de arquivos antes de iniciar serviços que precisam deles montados. Portanto, adiciona-se o CephFS no ***fstab*** para sempre que inicializar a máquina, ter disponível e montado o sistema de arquivos.

Adiciona-se então a entrada em ***$CHROOT/etc/fstab*** com a montagem do CephFS[9]:

```
1 ceph_ip:6789:/openhpc/home   /home   ceph    nodev,nosuid,
    _netdev,noatime,name=openhpc,secretfile=/etc/ceph/ceph.
    client.openhpc.secret       0       0
2 ceph_ip:6789:/openhpc/pub    /opt/ohpc/pub   ceph    ro,nodev
    ,_netdev,noatime,name=openhpc,secretfile=/etc/ceph/ceph.
    client.openhpc.secret      0       0
```

Realizamos a alteração no arquivo ***/etc/fstab*** também na SMS.

```
1 ceph_ip:6789:/openhpc/home       /home   ceph    nodev,nosuid
    ,_netdev,noatime,name=openhpc,secretfile=/etc/ceph/ceph.
    client.openhpc.secret       0       0
2 ceph_ip:/openhpc/pub        /opt/ohpc/pub   ceph    nodev,
    _netdev,noatime,name=openhpc,secretfile=/etc/ceph/ceph.
    client.openhpc.secret      0       0
```

Antes de realizar a montagem via *fstab*, monta-se manualmente e copia-se o conteúdo inteiro de */home* e */opt/ohpc/pub* originais para o CephFS.

[9] o IP foi omitido e substituido pelo texto ceph_ip para preservar a segurança

```
[sms]$ mount -t ceph ceph_ip:6789:/openhpc /mnt -o name=
   openhpc,secretfile=/etc/ceph/ceph.client.openhpc.secret
[sms]$ cp -a /home /mnt
[sms]$ cp -a /opt/ohpc/pub /mnt
[sms]$ umount /mnt
[sms]$ mount -a
```

Com isso concluímos a configuração do sistema de arquivos compartilhados CephFS. Agora todo arquivo em **/home** será sincronizado entre todos os nós da HPC-nee.

4.8 Identificando arquivos para sincronização utilizando xCAT

O sistema do xCAT possui a funcionalidade de sincronizar arquivos localizados na SMS para a distribuição nos nós gerenciados. Esta é uma maneira de compartilhar as credenciais dos usuários com os nós compute (Alternativamente poderia ser utilizado um serviço como LDAP). Para sincronizar os arquivos, habilita-se o arquivo ***synclist*** e altera-se o seu conteúdo para que sincronize alguns arquivos nessários como **/etc/passwd**, **/etc/group** e **/etc/shadow**. Adiciona-se em ***$CHROOT/etc/ceph/ceph.client.openhpc.secret*** e em ***/etc/ceph/ceph.client.openhpc.secret*** a chave gerada.

```
[sms]$ mkdir -p /install/custom/netboot
[sms]$ chdef -t osimage -o centos8-x86_64-netboot-compute
   synclists="/install/custom/netboot/compute.synclist"



[sms]$ echo "/etc/passwd -> /etc/passwd" > /install/custom/
   netboot/compute.synclist
[sms]$ echo "/etc/group -> /etc/group" >> /install/custom/
   netboot/compute.synclist
[sms]$ echo "/etc/shadow -> /etc/shadow" >> /install/custom/
   netboot/compute.synclist
[sms]$ echo "/etc/ceph/ceph.client.openhpc.secret -> /etc/
   ceph/ceph.client.openhpc.secret" >> /install/custom/
   netboot/compute.synclist
```

Utiliza-se o mesmo serviço para sincronizar as configurações do Slurm e chaves do **munge** nos nós compute.

```
[sms]$ echo "/etc/slurm/slurm.conf -> /etc/slurm/slurm.conf "
    >> /install/custom/netboot/compute.synclist
```

```
2 [sms]$ echo "/etc/munge/munge.key -> /etc/munge/munge.key"
    >>/install/custom/netboot/compute.synclist
```

4.9 Controle de acesso aos nós compute via ssh, utilizando pam_slurm_adopt

É interessante que usuários que estejam executando experimentos em um nó, possam acessar um interpretador de comandos shell deste nó via ssh. E também é necessário que o sistema previna usuários acessarem nós que não possuam um experimento em execução.

Para isso utiliza-se um *PAM module* que já vem incluído com o pacote slurm, chamado de **pam_slurm_adopt.so**.

Para adicioná-lo removemos a prioridade do **pam_slurm.so** e adiciona-se o **pam_slurm_adopt.so**.

```
1 [sms]$ echo "account    sufficient    pam_access.so" >>
    $CHROOT/etc/pam.d/sshd
2 [sms]$ echo "account    required      pam_slurm_adopt.so" >>
    $CHROOT/etc/pam.d/sshd
```

É necessário que o plugin de controle de processo do Slurm utilize *cgroups*. Também adiciona-se **PrologFlags=contain** que configura a etapa "externa" na qual os processos iniciados por ssh serão adotados.[10]

```
1 [sms]$ sed -ri 's,ProctrackType=.*,ProctrackType=proctrack/
    cgroup,' /etc/slurm/slurm.conf
2 [sms]$ sed -ri 's,TaskPlugin=.*,TaskPlugin=task/cgroup,' /etc
    /slurm/slurm.conf
3 [sms]$ echo 'PrologFlags=contain' >>/etc/slurm/slurm.conf
```

Cria-se então o arquivo de configuração dos cgroups em **/etc/slurm/cgroup.conf** com as seguintes configurações:

```
1 CgroupAutomount=yes
2 ConstrainDevices=no
3 ConstrainCores=yes
4 TaskAffinity=yes
5 ConstrainRAMSpace=yes
6 ConstrainSwapSpace=yes
```

Depois disso o sistema já está apto a realizar a tarefa de executar ssh em nós que estão executando jobs do usuário corrente.

[10] adopteds, pegou o trocadilho?

4.10 Habilitando o Slurm

Como não copia-se os arquivos da lista de sincronização para a imagem por motivos de segurança (opção `--nosyncfiles`), precisa-se copiar manualmente alguns arquivos que são necessários no processo de boot. Também removemos o hash das senhas para não ficarem expostas

```
[sms]$ cp -a /etc/passwd /etc/group /etc/shadow $CHROOT/etc
[sms]$ sed -ri 's,^([^:]+:)[^:]+,\1!!,' $CHROOT/etc/shadow
```

Precisa-se também instruir o munge, o serviço de autenticação utilizado pelo Slurm, a iniciar somente após os arquivos terem sido sincronizados. Edita-se o arquivo do serviço munged, ***$CHROOT/usr/lib/systemd/system/munge.service***, para esse fim.

```
[Unit]
Description=MUNGE authentication service
Documentation=man:munged(8)
After=network.target
After=time-sync.target
After=xcatpostinit1.service

[Service]
Type=forking
ExecStart=/usr/sbin/munged --syslog
PIDFile=/var/run/munge/munged.pid
User=munge
Group=munge
Restart=on-abort
PermissionsStartOnly=true
ExecStartPre=-/usr/bin/mkdir -m 0755 -p /var/log/munge
ExecStartPre=-/usr/bin/chown -R munge:munge /var/log/munge
ExecStartPre=-/usr/bin/mkdir -m 0755 -p /var/run/munge
ExecStartPre=-/usr/bin/chown -R munge:munge /var/run/munge

[Install]
WantedBy=multi-user.target
```

Depois disso, precisa-se habilitar o Slurm para iniciar automaticamente via **systemd**, no serviço chamado ***slurmd***

```
[sms]$ chroot $CHROOT systemctl enable slurmd
```

Após essas alterações pode-se empacotar a imagem.

```
[sms]$ packimage -c pigz --nosyncfiles centos8-x86_64-netboot
   -compute
```

4.11 Adicionando os nós compute no xCAT

Para finalizar a configuração do xCAT, adiciona-se os nós de compute e defini-se suas propriedades no banco de dados do xCAT. Esses hosts são agrupados logicamente em grupos chamados *compute* para facilitar comandos ao nível de grupo.

```
[sms]$ for ((i=0; i<$num_computes; i++)) ;
domkdef -t node ${c_name[$i]} groups=compute,all ip=${c_ip[$i
   ]} mac=${c_mac[$i]} netboot=xnba \arch=x86_64 bmc=${c_bmc[
   $i]} bmcusername=${bmc_username} bmcpassword=${
   bmc_password} \mgt=ipmi serialport=0 serialspeed=115200
done
```

Nota-se que para uma grande quantidade de nós é inviável utilizar o comando acima, para isso o xCAT provê ferramentas para o descobrimento de máquinas em rede. xCAT necessita ainda de um nome de domínio para realizar a resolução de nomes no sistema todo. Esse valor pode ser definido para corresponder com o esquema de DNS local ou pode ser chamado simplesmente de “local”.

```
[sms]$ chdef -t site domain=${domain_name}
```

Com os nós compute designados e domínio identificado, pode-se completar as configurações de rede como DNS e DHCP e selecionar efetivamente uma imagem para ser usada nos computes

```
[sms]$ makehosts
[sms]$ makenetworks
[sms]$ makedhcp -n
[sms]$ makedns -n

[sms]$ nodeset compute osimage=centos8-x86_64-netboot-compute
```

Pode-se então após estes passos inicializar os computes!

4.12 Inicializando os computes

Antes de inicializar, configura-se-los eles para usarem PXE (Pre Boot Execution) no seu próximo boot. Antes do primeiro PXE, a sequência de boot retornará ao uso do dispositivo de inicialização padrão especificada na BIOS.

```
[sms]$ rsetboot compute net
```

A SMS é capaz de inicializar os nós compute. Isto é feito utilizando o comando *rpower* que vem junto com o xCAT aproveitando o protocolo IPMI configurado durante a definição do nó compute. Pode-se então inicializar os computes.

```
1 [sms]$ rpower compute reset
```

Pode-se verificar se tudo ocorreu como esperado.

```
1 [sms]$ psh compute uptime
```

Caso esteja tudo certo, deve-se observar o tempo que cada nó está em pé.

4.13 Pacotes não instalados

O manual de instalação do OpenHPC sugere alguns pacotes opcionais que a equipe optou por não instalar, pois como as imagens são replicadas para todas as máquinas, quanto menos softwares desnecessários melhor, mantendo uma instalação mínima, que ocupa menos memória dos nós, contendo apenas o necessário para o funcionamento e gerenciamento da HPC-nee. Os pacotes não instalados são: Lustre, Nagios, Ganglia, ClusterShell, mrsh e genders.

4.13.1 Lustre

O Lustre é um sistema de arquivos distribuído que tem o propósito de fornecer um namespace coerente e global compatível com POSIX para infraestrutura de computadores em grande escala. Ele pode suportar centenas de petabytes de armazenamento de dados e centenas de gigabytes por segundo em taxa de transferência simultânea. Algumas das maiores instalações HPC atuais usam Lustre. Utilizamos o Ceph como uma alternativa mais simples de operar, embora com menor desempenho com relação ao Lustre.

4.13.2 Nagios

Nagios é um programa de monitoramento de host/serviço/rede escrito em C e lançado sob a licensa GNU General Public License Version 2. Programas CGI são incluídos para permitir que você visualize o status das máquinas, histórico, etc. através de uma interface da web (17). Pretende-se utilizar o monitoramento através do Prometheus como substituto ao Nagios no futuro.

4.13.3 Ganglia

Ganglia é um sistema de monitoramento distribuído escalonável para sistemas de computação de alto desempenho, como clusters e Grids. É baseado em um design hierárquico voltado para fazendas de clusters. Ele aproveita tecnologias amplamente utilizadas, como XML para representação de dados, XDR para transporte de dados compacto e portátil

e RRDtool para armazenamento e visualização de dados. Ele usa estruturas de dados e algoritmos cuidadosamente projetados para atingir sobrecargas por nó muito baixas e alta simultaneidade (18). No entanto, o Ganglia possui um histórico um pouco preocupante de vulnerabilidades de segurança reportadas (CVEs). Pretende-se utilizar o monitoramento via Prometheus como substituto ao Ganglia no futuro.

4.13.4 ClusterShell

ClusterShell é uma biblioteca Python de código aberto orientada a eventos, projetada para executar comandos locais ou remotos em paralelo em fazendas de servidores ou em grandes clusters Linux. Ele cuida de problemas comuns encontrados em clusters HPC, como operar em grupos de nós, executar comandos distribuídos usando algoritmos de execução otimizados, bem como reunir resultados e mesclar saídas idênticas ou recuperar códigos de retorno. O ClusterShell aproveita as vantagens dos recursos de shell remoto existentes já instalados em seus sistemas, como o SSH (19). Utilizaremos outro pacote para realizar essas tarefas.

4.13.5 mrsh

Uma POSIX shell mínima. Não houve interesse algum neste pacote.

4.13.6 genders

Genders é um banco de dados de configuração de cluster estático usado para gerenciamento de configuração de cluster. Ele é usado por uma variedade de ferramentas e scripts para gerenciamento de grandes clusters. O banco de dados de gêneros é normalmente replicado em todos os nós do cluster. Ele descreve o layout e a configuração do cluster para que as ferramentas e scripts possam detectar as variações dos nós do cluster. Ao abstrair essas informações em um arquivo de texto simples, é possível alterar a configuração de um cluster modificando apenas um arquivo (20). Optou-se por não realizar a instalação deste pacote pois as suas funcionalidades não seriam relevantes para a utilização da HPC-nee.

4.13.7 ConMan

ConMan é um gerenciador de console serial criado para suportar múltiplos dispositivos e usuários simultâneos. Ele suporta: dispositivos seriais, telnet, IPMI Serial-Over-LAN (via FreeIPMI), Unix Domain Socket. O ConMan possui uma vulnerabilidade em que os usuários poderiam possuir acesso a BIOS das computes durante o reboot, representando assim uma ameaça para a segurança (21). Ele é desnecessário pois as funções desempenhadas por eles já são feitas de maneira segura através da ferramenta *rcons* do xCAT.

5 INSTALANDO COMPONENTES DE DESENVOLVIMENTO DO OPENHPC

É necessário instalar uma gama de ferramentas para que o usuário possa utilizar a HPC-nee para executar seus programas, sejam eles distribuídos ou locais.

5.1 Bibliotecas de MPI

Para a execução de programas que utilizem MPI, instala-se bibliotecas de MPI.

```
1 [sms]$ dnf -y install mpich-gnu9-ohpc openmpi4-gnu9-ohpc lmod
    -defaults-gnu9-openmpi4-ohpc
```

5.2 Engines de Container

Para aplicações que utilizam containers seguiu-se a recomendação do manual do OpenHPC instalando duas engines de container sugeridas.

5.2.1 Singularity

Singularity é uma plataforma de container. Ele permite que se crie e execute containers que empacotam pedaços de software de uma forma que seja portátil e reproduzível (22). Constroe-se um container usando Singularity em seu laptop e, em seguida, executa-se em muitos dos maiores clusters de HPC do mundo, clusters de universidades ou empresas locais, um único servidor, na nuvem ou em uma estação de trabalho no corredor. Um container é um único arquivo. (23)

Pode-se instalá-la então na SMS.

```
1 [sms]$ dnf -y install singularity-ohpc
```

5.2.2 Charlie-Cloud

Charliecloud usa namespaces de usuário Linux para executar contêineres sem operações ou daemons privilegiados e mudanças mínimas de configuração nos recursos do kernel. Essa abordagem simples evita a maioria dos riscos de segurança enquanto mantém o acesso ao desempenho e à funcionalidade já oferecidos. (24)

Pode-se instalar com o seguinte comando:

```
1 [sms]$ dnf -y install charliecloud-ohpc
```

5.2.3 Parâmetros de kernel utilizados nos nós compute

Para a containerização funcionar corretamente é necessário que a opção do kernel que habilita a criação de namespaces do usuário nas máquinas compute esteja habilitado. Pode-se

habilitá-la passando o seguinte parâmetro para o kernel **namespace.unpriv_enable=1**. Utilizamos o programa xCAT para definir os parâmetros de inicialização do kernel. Passa-se um parâmetro para o módulo de ACPI, para desabilitá-lo já que não serve para nada a não ser consumir tempo de CPU para não deixar o sistema ocioso. Desabilitamos também o subsistema de escalamento de CPUs da intel.

```
[sms]$ chdef -t osimage centos8.3-x86_64-netboot-compute
   addkcmdline="acpi_pad.disable=1 intel_pstate=disable
   namespace.unpriv_enable=1"

[sms]$ nodeset compute osimage=centos8.3-x86_64-netboot-
   compute
```

Ao habilitar a opção de criação de namespaces que será utilizada pelos containers, também aumenta-se o limite de namespaces que o sistema suporta adicionando a seguinte linha.

```
[sms]$ echo "user.max_user_namespaces=15000" >> $CHROOT/etc/
   sysctl.conf
```

Para utilizar namespaces com a engine de container Singularity, precisa-se também instalar o pacote *squashfs-tools* e adicionar a sincronização dos arquivos subgid e subuid. Após isso atualizamos os nós.

```
[sms]$ dnf -y --installroot=$CHROOT install squashfs-tools
[sms]$ echo "/etc/subuid -> /etc/subuid" >> /install/custom/
   netboot/compute.synclist
[sms]$ echo "/etc/subgid -> /etc/subgid" >> /install/custom/
   netboot/compute.synclist
[sms]$ updatenode compute -F
```

Já para o uso do motor de containers Charlie Cloud, são necessários os seguintes pacotes na imagem:

```
[sms]$ dnf -y --installroot=$CHROOT install fuse squashfuse
```

Depois disso, atualizamos os nós.

5.3 Inicialização de gerenciamento de recursos

Previamente, instala-se o Slurm e o configura-se para uso tanto na SMS quanto nos nós compute. Com os nós compute ligados e funcionando, pode-se agora inicializar o gerenciador de recursos para execução de experimentos. Inicializa-se os serviços na SMS e nos nós compute. O Slurm utiliza o Munge para a autenticação nos nós compute. O seguinte comando pode ser utilizado para inicializar os serviços necessários para o gerenciamento de recursos através do slurm, na SMS e nos nós compute.

```
[sms]$ systemctl enable munge
[sms]$ systemctl enable slurmctld
[sms]$ systemctl start munge
[sms]$ systemctl start slurmctld

[sms]$ pdsh -w $compute_prefix[1-4, 11] systemctl start munge

[sms]$ pdsh -w $compute_prefix[1-4, 11] systemctl start
   slurmd
```

O Slurm por padrão utiliza o plugin de seleção de nós select/linear, que aloca um nó inteiro para cada experimento. Caso o usuário não especifique a quantidade de memória, defini-se abaixo de 8000MB (valor escolhido por ser a quantidade de memória dividida por vCPU) por vCPU como padrão. Atualiza-se o arquivo **/etc/slurm/slurm.conf** para alterar essas configurações.

```
[sms]$ echo "SelectType=select/cons_res"  >> /etc/slurm/slurm
   .conf
[sms]$ echo "SelectTypeParameters=CR_CPU_Memory" >> /etc/
   slurm/slurm.conf
[sms]$ echo "DefMemPerCPU=8000"  >> /etc/slurm/slurm.conf
```

Uma configuração a ser feita no slurm é a adição do programa que será responsável por reiniciar os nós. Altera-se o arquivo **/etc/slurm/slurm.conf** e adiciona-se essa especificação.

```
[sms]$ echo "RebootProgram=/usr/sbin/reboot"  >> /etc/slurm/
   slurm.conf
```

5.3.1 Finalizando configuração do NHC

Para evitar que o NHC utilize o FQDN, que o slurm não reconhece, cria-se um arquivo em **$CHROOT/etc/sysconfig/nhc** especificando o hostname a ser usado pelo *nhc*.

```
[sms]$ echo "\$(hostname -s)" > $CHROOT/etc/sysconfig/nhc
```

Pode-se utilizar o arquivo de configuração base do NHC, para gerar este arquivo utiliza-se o seguinte comando:

```
[sms]$ pdsh -w c1 "/usr/sbin/nhc-genconf -H '*' -c -" |
   dshbak -c
```

Após gerar a configuração inicial remove-se as 3 primeiras linhas onde está o nome do host que gerou a configuração. Após ter acesso à configuração inicial pode-se sincronizar este arquivo com o comando:

```
[sms]$ echo "/etc/nhc/nhc.conf -> /etc/nhc/nhc.conf" >> /
   install/custom/netboot/compute.synclist
[sms]$ updatenode compute -F
```

5.3.2 Carregando o módulo do singularity automaticamente

Com a configuração atual para utilizar o singularity é necessário carregar seu módulo toda vez que for utilizá-lo.

```
[sms]$ module load singularity
```

Como a maior parte dos experimentos serão executados com singularity, pode-se carregar automaticamente este módulo por conveniência do usuário. Para isso altera-se o arquivo **/opt/ohpc/pub/modulefiles/ohpc** e adiciona-se o módulo singularity na lista de módulos que deve ser carregado automaticamente com o OpenHPC.

```
proc ModulesHelp { } {
puts stderr "Setup default login environment"
}

#
# Load Desired Modules
#

prepend-path     PATH   /opt/ohpc/pub/bin

if { [ expr [module-info mode load] || [module-info mode
   display] ] } {
        prepend-path MANPATH /usr/local/share/man:/usr/share/
           man/overrides:/usr/share/man/en:/usr/share/man
        module try-add autotools
        module try-add prun
        module try-add gnu8
        module try-add openmpi3
        module try-add singularity
}

if [ module-info mode remove ] {
        module del openmpi3
        module del gnu8
        module del prun
        module del autotools
}
```

5.3.3 Disco local dos computes

Os nós compute utilizam, por causa da configuração, o sistema de arquivos compartilhado CephFS. Mas como essas máquinas são físicas e possuem uma memória secundária, é interessante que se faça uso desses recursos. Para isso precisa-se adicionar a entrada de montagem no **fstab**. Escolheu-se o nome **/scratch** para o ponto de montagem do disco local de cada compute. Pode-se alterar o fstab adicionando a nova entrada.

```
[sms]$ echo "LABEL=scratch /scratch ext4 rw,noatime,nofail,x-
   systemd.device-timeout=5 0 0" >> $CHROOT/etc/fstab
```

Devemos criar o caminho **/scratch** que ainda não existe nas imagens dos nós.

```
[sms]$ mkdir -p $CHROOT/scratch
[sms]$ chmod 1777 $CHROOT/scratch
```

Após a criação dos diretórios e a aplicação das mudanças nos nós, deve-se formatar a partição, deixá-la do tipo ext4, adicionar as permissões corretas e checar a saúde dos nós.

```
[sms]$ pdsh -w $compute_prefix[1-4, 11] mkfs.ext4 -L scratch
   /dev/sda

[sms]$ pdsh -w $compute_prefix[1-4, 11] chmod 1777 /scratch

[sms]$ pdsh -w $compute_prefix[1-4, 11] nhc

# Nota-se que este comando so funciona pois as especificacoes
    de montagem ja estao no fstab
[sms]$ pdsh -w $compute_prefix[1-4, 11] mount -a
```

Como a área **/scratch** será um ponto de montagem utilizado por muitos usuários é interessante que os arquivos permaneçam lá por um tempo limitado. Para apagar automaticamente os arquivos temporários após 10 dias sem modificações, pode-se utilizar o seguinte comando:

```
[sms]$ echo "v /scratch 1777 root root 10d" > $CHROOT/etc/
   tmpfiles.d/scratch.conf
```

É necessário informar o ponto de montagem do disco temporário para o Slurm no arquivo **/etc/slurm/slurm.conf**.

```
[sms]$ echo "TmpFS=/scratch" >> /etc/slurm/slurm.conf
```

Pode-se também utilizar a memória swap dos nós computes, adiocionando o ponto de montagem da swap no **fstab**.

```
[sms]$ echo "LABEL=swap none swap nofail,x-systemd.device-
   timeout=5 0 0" >> $CHROOT/etc/fstab
```

Cria-se a swap nos nós, monta-se e ativa-se a swap.

```
[sms]$ pdsh -w $compute_prefix[1-4, 11] mkswap -L swap /dev/
   sdb

[sms]$ pdsh -w $compute_prefix[1-4, 11] mount -a

[sms]$ pdsh -w $compute_prefix[1-4, 11] swapon -a
```

5.3.4 CUDA

Como a máquina c11 possue uma GPU Nvidia, é essencial que instalemos os drivers da placa gráfica. Para isto, adiciona-se os repositórios oficiais da Nvidia e instala-se os drivers com:

```
[sms]$ wget https://developer.download.nvidia.com/compute/
   cuda/repos/rhel8/x86_64/cuda-repo-rhel8-10.2.89-1.x86_64.
   rpm

[sms]$ dnf --installroot=$CHROOT install -y cuda-repo-rhel8
   -10.2.89-1.x86_64.rpm

[sms]$ dnf --installroot=$CHROOT install -y kernel-devel
   kernel-headers cuda-drivers cuda-libraries-10-2
```

Para não esperar a compilação do módulo de kernel durante o boot, pode-se pré compilá-lo.

```
for x in sys proc dev; do mount --bind /$x $CHROOT/$x; done

chroot $CHROOT dkms autoinstall --verbose --kernelver $(uname
    -r)

for x in sys proc dev; do umount $CHROOT/$x; done
```

Após instalarmos os drivers, precisa-se especificar quais máquinas possuem GPU Nvidia, informando o número de placas que o nó possui, informando a arquitetura e o modelo da placa editando o arquivo **/etc/slurm/slurm.conf** e inserindo essas informações.

```
GresTypes=gpu
NodeName=c[1-4] Sockets=2 CoresPerSocket=10 ThreadsPerCore=2
   RealMemory=354505 State=UNKNOWN
NodeName=c11 Sockets=2 CoresPerSocket=10 ThreadsPerCore=2
   RealMemory=322138 Gres=gpu:turing:2 State=UNKNOWN
```

Depois inclui-se o arquivo **/etc/slurm/gres.conf** na lista de sincronização.

```
echo "/etc/slurm/gres.conf -> /etc/slurm/gres.conf" >> /
   install/custom/netboot/compute.synclist
```

Com isso finaliza-se a configuração das máquinas com Nvidia.

5.3.5 Slurmdb

Slurmdbd é um programa que provém uma interface segura, do Slurm para um banco de dados, especialmente útil para configura-se prioridade de usuários na alocação de recursos

nos nós. Pode-se criar um arquivo de configuração do **/etc/slurm/slurmdbd.conf** e configurar os parâmetros como desejado.

```
ArchiveEvents=yes
ArchiveJobs=yes
ArchiveResvs=yes
ArchiveSteps=yes
ArchiveSuspend=yes
ArchiveTXN=yes
ArchiveUsage=yes
AuthType=auth/munge
DbdHost=localhost
DbdPort=****
DebugLevel=debug5
DebugLevelSyslog=debug5
PurgeEventAfter=1month
PurgeJobAfter=12month
PurgeResvAfter=1month
PurgeStepAfter=1month
PurgeSuspendAfter=1month
PurgeTXNAfter=12month
PurgeUsageAfter=24month
LogFile=/var/log/slurmdbd.log
PidFile=/var/run/slurmdbd.pid
SlurmUser=slurm
StoragePass=senha_segura_redacted
StorageType=accounting_storage/mysql
StorageUser=slurm
StorageLoc=slurm_acct_db
StorageHost=localhost
StoragePort=****
```

Depois de criar este arquivo, altera-se a propriedade do arquivo para o usuário slurm e deixamos ele apenas read-only.

```
[sms]$ chown slurm:slurm /etc/slurm/slurmdbd.conf
[sms]$ chmod 600 /etc/slurm/slurmdbd.conf
```

Instala-se então o **slurmdb** que é a interface do slurm para um banco de dados.

```
[sms]$ dnf install -y slurm-slurmdbd-ohpc
```

Instala-se então efetivamente o banco de dados, inicializa-se os seus serviços e cria-se um usuário root com uma senha para utilizá-lo.

```
[sms]$ dnf install -y mariadb-server mariadb-devel

[sms]$ systemctl enable --now mariadb

[sms]$ /usr/bin/mysql_secure_installation
```

Depois disso, conecta-se ao banco de dados e cria-se um usuário para o Slurm. Adiciona-se acesso irrestrito ao banco de dados local e cria-se um banco de dados para ele utilizar.

```
[sms]$ mysql -uroot -p
```

```
create user 'slurm'@'localhost' identified by 'password';
grant all on *.* TO 'slurm'@'localhost' identified by '
   senha_segura_redacted' with grant option;
create database slurm_acct_db;
```

Inicia-se então o serviço do slurmdbd.

```
[sms]$ systemctl enable --now slurmdbd
```

Agora pode-se utilizar o programa ***sacctmgr*** para gerenciar usuários na alocação de experimentos feitas pelo slurm e QoS.

5.3.6 Qualidade do Serviço (QoS)

QoS ou quality of service, é um termo usado para definir quais tipos de serviços e prioridades de jobs no existirão na HPC-nee. Por exemplo, pode-se definir um QoS apenas para jobs rápidos que irão demorar até 1h, ou um QoS para jobs que utilizarão apenas a GPU. Esses QoS são definidos em um banco de dados, e na hora da alocação de recursos o Slurm confere este banco para definir as prioridades de cada usuário em relação ao QoS requisitado.

Pode-se criar um banco de dados para gerenciar configurações do tipo:

a) Quais tipos de QoS irão existir

b) Qual usuário tem acesso a qual QoS

c) Quais são as prioridades dos usuários em relação aos QoS

d) Quais recursos estarão disponíveis para cada QoS

Para habilitar o QoS no Slurm e definir algumas prioridades, altera-se o arquivo **/etc/slurm/slurm.conf**.

```
PriorityType=priority/multifactor
#PriorityDecayHalfLife=14-0
#PriorityUsageResetPeriod=14-0
PriorityWeightFairshare=1
PriorityWeightAge=1
PriorityWeightPartition=0
PriorityWeightJobSize=0
PriorityWeightQOS=100000
#PriorityMaxAge=1-0

# Configuracoes para QOS
AccountingStorageEnforce=associations,limits,qos
AccountingStorageTRES=gres/gpu
AccountingStorageType=accounting_storage/slurmdbd
AccountingStorageHost=localhost
AccountingStoragePort=6819
#AccountingStoragePass=/var/run/munge/munge.socket.2
#AccountingStorageUser=slurm
JobCompType=jobcomp/slurmdbd
#AccountingStorageLoc=/var/log/slurm/accounting
JobCompLoc=/var/log/slurm/job_completions
```

Cria-se então uma entidade cluster com a propriedade linux, finalizando assim as configurações mínimas para o slurm alocar experimentos na HPC-nee.

```
[sms]$ sacctmgr add cluster linux
```

Com isso o cluster OpenHPC está funcionando.

5.4 Pontos não abordados na instalação

a) Controlando a exposição das computes na Internet

b) Monitoramento com Prometheus

c) Configuração de rede ou firewall

d) Redirecionamento do ambiente gráfico X11

e) Email de notificação dos jobs

f) Upgrade automático da SMS

g) Upgrade automático dos nós

6 EXECUÇÃO DE UM EXPERIMENTO

Com a infraestrutura do OpenHPC pronta, executa-se um programa distribuído em Scala Spark para verificar se a stack implementada está funcionando como o esperado. Os scripts do experimento aqui mencionados podem ser encontrados em <https://gitlab.com/ufscar/hpc/spark>

6.1 Criação de um container

Para criar containers é necessário instalar a engine de containers Singularity em uma máquina pessoal local ou utilizar os serviços em <https://cloud.sylabs.io/builder> para construir containers. No sistema operacional Arch Linux, pode-se instalar singularity utilizando um script de instalação da comunidade encontrado no AUR (Arch User Repository). Realiza-se a clonagem do repositório que contém o script, e utiliza-se o programa ***makepkg*** para realizar a instalação.

```
1 [localMachine]$ git clone https://aur.archlinux.org/
     singularity-container.git && cd singularity-container
2 [localMachine]$ makepkg -si --noconfirm
```

Com a engine de container instalada localmente pode-se criar uma *recipe file*, arquivo que descreve como deve ser realizada a criação da imagem do container, se ele deve ser derivado de uma imagem já existente, quais programas serão instalados no container, quais serão as regras de execução deste container, qual programa deve ser o ponto de entrada e quais variáveis de ambiente serão definidas. O singularity possui compatibilidade com Docker, ou seja, pode-se utilizar imagens já disponíveis no DockerHub. Por motivo de familiaridade, utilizamos uma imagem de Arch Linux.

Cria-se então um arquivo chamado ***ContainerRecipe***, e adiciona-se o seguinte conteúdo:

```
1  Bootstrap: docker
2  From: archlinux
3
4  %runscript
5    exec echo "Vamos manter simples... Nao vamos criar um
       script de execucao cheio de regras. Vamos executar o
       spark diretamente com âparmetros na execucao singularity
       "
6
7  %post
8    echo "Seguindo a ArchWiki (Wiki do ArchLinux) altera-se a
       timezone"
9
10   ln -s /usr/share/zoneinfo/UTC /etc/localtime
11   echo 'en_US.UTF-8 UTF-8' > /etc/locale.gen
12   echo 'LANG=en_US.UTF-8' > /etc/locale.conf
13
14   echo "Configuramos o gerenciador de pacotes pacman"
15
16   pacman-key --init
17
18   echo "Configuramos a lista de mirrors"
19
20   echo 'Server = http://mirror.ufscar.br/archlinux/$repo/os/
       $arch' > /etc/pacman.d/mirrorlist
21   echo 'Server = http://br.mirror.archlinux-br.org/$repo/os/
       $arch' >> /etc/pacman.d/mirrorlist
22   echo 'Server = http://archlinux.c3sl.ufpr.br/$repo/os/$arch
       ' >> /etc/pacman.d/mirrorlist
23
24
25   echo "instala-se softwares necessarios para a execucao do
       experimento."
26   pacman -Syu --noconfirm base base-devel util-linux git
       scala pacman-contrib jdk8-openjdk inetutils wget rsync
27
28
29   echo "instala-se Apache Spark"
```

```
30  pacman -S apache-spark

32  echo "Trocamos a versao do java para o 8, para nao crashar
        o spark."
33  archlinux-java set java-8-openjdk

35  echo "Removemos lixo do container"
36  paccache -r -k0

38  echo "Container pronto!"
```

Pode-se então construir o container e nomeá-lo de **Spark.sif**

```
1 singularity build Spark.sif ContainerRecipe
```

Testá-se executando um comando que existe dentro do container, como **spark-shell**

```
1 singularity exec Spark.sif spark-shell
```

Caso tudo tenha ocorrido bem, a execução deste comando deve abrir shell do Spark.

6.2 Escrevendo script de execução do experimento

A execução de um experimento OpenHPC se divide majoritariamente em três partes.

a) Fazer o download de arquivos externos usados pelo experimento (datasets, arquivos grandes)

b) Execução do programa em sí

c) Recuperação do resultado

A primeira etapa, caso exista, pode ser feita realizando o upload desses arquivos para um servidor ftp ou algum serviço de cloud como **Google Cloud** ou **AWS S3**. A segunda etapa é individual de cada experimento, usualmente um programa irá rodar e gerará uma saída. A terceira etapa é recuperar a saída de alguma forma, realizando o upload dos dados para algum serviço externo como AWS S3 ou Google Cloud, ou enviando os resultados via email por exemplo.

Já execução de um experimento utilizando Spark também se divide em três partes principais: execução do nó mestre, execução dos nós trabalhadores e submissão do experimento em sí.

6.2.1 Execução do nó mestre

A priori, executar o script já definido pelo Spark **run-master.sh** já seria o suficiente para iniciar um nó mestre. Em um ambiente OpenHPC é necessária a criação de um script que será submetido ao Slurm via **sbatch**, o Slurm irá alocar um nó e executar as instruções

dadas. É necessário também que tenhamos a saída padrão e a saída de erros do programa que será executado em um nó. Para se ter acesso ao resultado final do experimento e resultado das saídas padrões, vamos utilizar o Telegram, para receber esses resultados em uma aplicação de mensagens. O script que será submetido ao Slurm é apresentado a seguir.

```
#!/bin/bash
#SBATCH --job-name=SparkMaster
#SBATCH -n 1
#SBATCH --output=MasterOutput.o
#SBATCH --error=MasterError.e

HOSTNAME=$(hostname)
localJob="/scratch/job.${SLURM_JOB_ID}"
chatId="[REDACTED]"
botKey="[REDACTED]"

# Spark config
masterNode=$MASTER_NODE
sparkSimg="/home/olivato/Spark.sif"
sparkLogPath="${localJob}/logs"

convertsecs() {
 ((h=${1}/3600))
 ((m=(${1}%3600)/60))
 ((s=${1}%60))
 printf "%02d:%02d:%02d\n" $h $m $s
}

function sendFile(){
  curl -F "chat_id=$chatId" -F "document=@$1" "https://api.
     telegram.org/$botKey/sendDocument"
}

function cleanJob(){
  echo "Limpando ambiente..."
  rm -rf "${localJob}"
}
trap cleanJob EXIT HUP INT TERM
```

```
34 function sendErr(){
35   sendMsg "Exited with error!"
36   sendFile "Error.e"
37   sendFile "Output.o"
38   cleanJob
39 }
40 trap ERR

42 function sendMsg(){
43   curl -X POST -H 'Content-Type: application/json' -d "{\"
        chat_id\": \"$chatId\", \"text\": \"$1\", \"
        disable_notification\": true}"  "https://api.telegram.
        org/$botKey/sendMessage"
44 }

46 mkdir -p "${sparkLogPath}"

48 sendMsg "Iniciando master. Hostname = $HOSTNAME ID = ${
     SLURM_JOB_ID}."

50 SECONDS=0

52 srun singularity exec \
53   --bind=/scratch:/scratch \
54   --bind=/var/spool/slurm:/var/spool/slurm \
55   --bind=$sparkLogPath:/opt/apache-spark/logs \
56   --bind=$localJob:/opt/apache-spark/work/ \
57   $sparkSimg run-master.sh -h $masterNode -p 7077

59 retCode=$?
60 if [[ "$retCode" -ne 0 ]]; then
61   sendErr
62   exit 1
63 fi

65 duration=$SECONDS
66 FinalTime=$(convertsecs $duration)
67 sendMsg "Job finalizado em $FinalTime ou $duration segundos"
68 sendFile Output.o
```

Nas linhas 2, 3, 4 e 5 defini-se algumas variáveis utilizadas pelo Slurm, no caso o nome do script que será executado. O número de CPUs necessárias para a execução, o arquivo de saída do stdout e o arquivo de saída do stderr.

Na linha 8 defini-se onde será o diretório de trabalho do experimento, arquivos de entrada, arquivos de saída, arquivos intermediários etc. Note que utilizamos o /scratch, ou seja, o disco local da máquina que será o nó mestre do Spark.

Nas linhas 9 e 10 as credenciais de API do Telegram, para onde será enviado a saída da execução

Na linha 13 a definição da variável masterNode, pode-se assim especificar qual nó queremos que seja o nó mestre.

Na linha 14 o caminho do container Spark criado.

Na linha 17 tem-se uma função auxiliar que converte segundos para horas, minutos e segundos.

Na linha 24 uma função auxiliar para enviar arquivos para o Telegram.

Na linha 28 uma função essencial que limpa o ambiente após a execução do job. Note que esta função é ativada quando os seguintes sinais forem enviados para ela: EXIT, HUP, INT e TERM. No caso ela será recebida quando o experimento finalizar, for cancelado ou ocorrer um timeout.

Na linha 32 tem-se uma função importante, que caso ocorra erros envia-se o stdout, stderr.

Na linha 42 tem-se uma função auxiliar que envia mensagens para o nosso chat no Telegram.

Na linha 52 tem-se a execução do container, o mapeamento do /scratch no container para /scratch no sistema que a está executando e a execução do nó mestre.

Na linha 59 tem-se o retorno da execução do nó, caso ele falhe envia-se um erro e saímos.

Na linha 67, e 68 envia-se a duração total do job e os arquivos de saída.

Nas demais linhas o conteúdo é trivial, e por isso não será comentado.

Salva-se esse script, envia-se ele à máquina SMS e submetem-se ele ao Slurm, para ele alocar um nó para realizar a execução.

```
1 [local]$ scp ./SlurmSparkMaster.sh sms:/home/usuario_corrente
    /
2 [sms]$ sbatch SlurmSparkMaster.sh
```

Após isso é possível criar um tunnel ssh entre a **SMS->Nó Mestre** e **Local->SMS** para visualizar a execução do nó no navegador.

Figura 7 – Exemplo de execução de um nó mestre Spark dentro do OpenHPC UFSCar. Utilizando a máquina c11 como mestre.

Fonte: Autor

6.2.2 Execução do nó trabalhador

A ideia de criar um script para o Slurm se mantém a mesma. A única necessidade agora é specificar o nó mestre.

```
1  #!/bin/bash
2  #SBATCH --job-name=SparkWorker
3  #SBATCH -n 1
4  #SBATCH --output=Output.o
5  #SBATCH --error=Error.e

7  HOSTNAME=$(hostname)
8  localJob="/scratch/job.${SLURM_JOB_ID}"
9  chatId="[REDACTED]"
10 botKey="[REDACTED]"

12 # Spark config
13 masterNode=$MASTER_NODE
14 sparkSimg="/home/olivato/Spark.sif"
15 sparkLogPath="${localJob}/logs"

17 convertsecs() {
18  ((h=${1}/3600))
19  ((m=(${1}%3600)/60))
20  ((s=${1}%60))
21  printf "%02d:%02d:%02d\n" $h $m $s
22 }
```

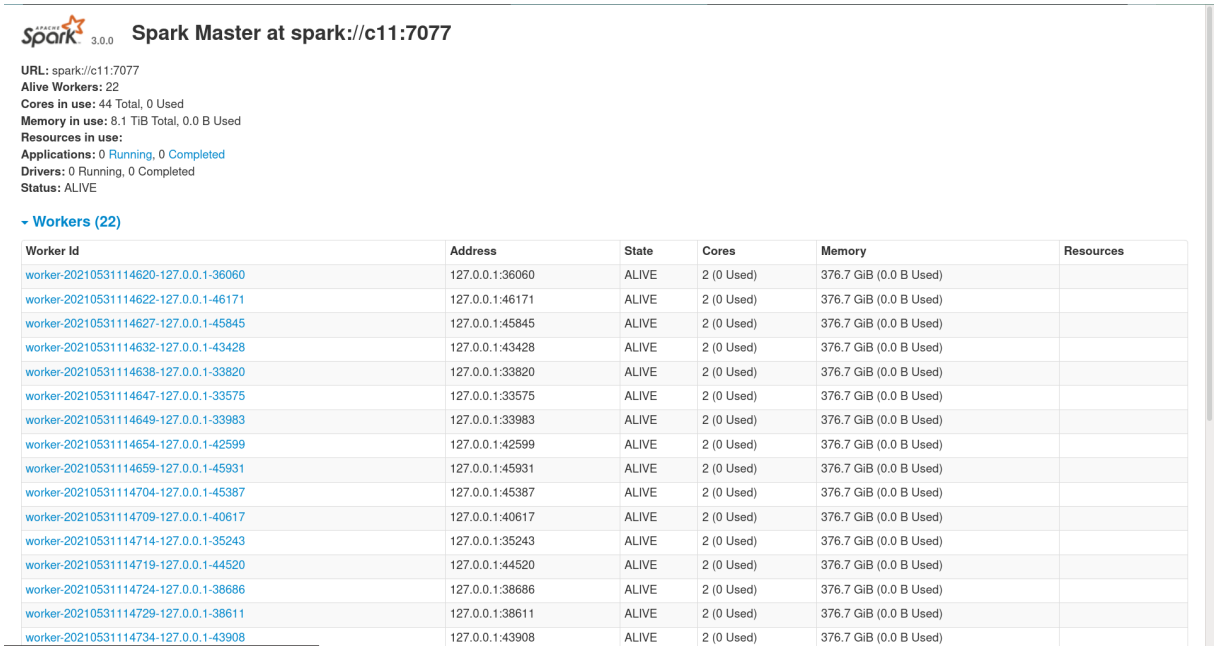
```
24 function sendFile(){
25   curl -F "chat_id=$chatId" -F "document=@$1" "https://api.
       telegram.org/$botKey/sendDocument"
26 }

28 # Function to clear job
29 function cleanJob(){
30   echo "Limpando ambiente..."
31   rm -rf "${localJob}"
32 }
33 trap cleanJob EXIT HUP INT TERM

35 function sendErr(){
36   sendMsg "Exited with error!"
37   sendFile "Error.e"
38   sendFile "Output.o"
39   cleanJob
40 }
41 trap ERR


44 function sendMsg(){
45   curl -X POST -H 'Content-Type: application/json' -d "{\"
       chat_id\": \"$chatId\", \"text\": \"$1\", \"
       disable_notification\": true}"  "https://api.telegram.
       org/$botKey/sendMessage"
46 }

48 mkdir -p "${sparkLogPath}"

50 sendMsg "Iniciando Worker: ${HOSTNAME} -- ID: ${SLURM_JOB_ID}
     "

52 SECONDS=0

54 srun singularity exec \
55   --bind=/scratch:/scratch \
56   --bind=/var/spool/slurm:/var/spool/slurm \
```

```
57    --bind=$sparkLogPath:/opt/apache-spark/logs \
58    --bind=$localJob:/opt/apache-spark/work/ \
59    $sparkSimg run-slave.sh spark://$masterNode:7077

61  retCode=$?
62  if [[ "$retCode" -ne 0 ]]; then
63    sendErr
64    exit 1
65  fi

67  duration=$SECONDS
68  FinalTime=$(convertsecs $duration)
69  # sendMsg "Worker finalizado em $FinalTime ou $duration
      segundos"
```

A única diferença do script que envia um worker para um script que envia a master é a linha que ao invés de rodar o programa **run-master.sh** é executado o **run-slave.sh** passando como parâmetro o nó mestre. Pode-se executar múltiplos jobs de trabalhadores para o cluster Spark.

```
1  [local]$ scp ./SlurmSparkWorker.sh sms:/home/usuario_corrente
     /
2  [sms]$ for i in {1..22}; do sbatch SlurmSparkWorker.sh; done
```

Figura 8 – Exemplo de um cluster Spark pronto para receber jobs



Fonte: Autor

6.2.3 Submissão do programa para executar no Spark cluster

Agora que tem-se um cluster Spark online e pronto para receber arquivos para executar, monta-se novamente um script de execução mas dessa vez utilizaremos o comando **spark-submit**, para enviar um arquivo **jar** para ser executado pelo cluster criado. Para o teste utilizamos o seguinte programa em Scala que aproxima o valor de PI.

```
val count = sc.parallelize(1 to 10000).filter { _ =>
  val x = math.random
  val y = math.random
  x*x + y*y < 1
}.count()
println(s"Pi is roughly ${4.0 * count / 10000}")
```

cria-se então um projeto Scala-Spark, utilizamos o código fonte acima e compilamos o programa para um arquivo chamado **PI.jar**

A ideia de criar um script para o Slurm se mantém a mesma. A única necessidade agora é specificar parâmetros de execução do Scala Spark.

```
#!/bin/bash
#SBATCH --job-name=SparkSubmit
#SBATCH -n 1
#SBATCH --output=SparkSubmit.o
#SBATCH --error=SparkSubmit.e



localJob="/scratch/job.${SLURM_JOB_ID}"
chatId="[REDACTED]"
botKey="[REDACTED]"

masterNode=$MASTER_NODE
sparkSimg="/home/olivato/Spark.sif"
sparkLogPath="${localJob}/logs"
jarFile="PI.jar"
class="main.Main"
sparkParameters=" --conf spark.eventLog.enabled=false --conf
   spark.app.name='Dynamic Variables' --conf spark.driver.
   cores=2 --conf spark.driver.maxResultSize=2g --conf spark.
   driver.memory=2g --conf spark.executor.memory=2g --conf
   spark.master=spark://c11:7077"
jarParameters=""
```

```
20 convertsecs() {
21  ((h=${1}/3600))
22  ((m=(${1}%3600)/60))
23  ((s=${1}%60))
24  printf "%02d:%02d:%02d\n" $h $m $s
25 }

27 function sendFile(){
28   curl -F "chat_id=$chatId" -F "document=@$1" "https://api.
        telegram.org/$botKey/sendDocument"
29 }

31 function cleanJob(){
32   echo "Limpando ambiente..."
33   rm -rf "${localJob}"
34 }
35 trap cleanJob EXIT HUP INT TERM

37 function sendErr(){
38   sendMsg "Exited with error!"
39   sendFile "Error.e"
40   sendFile "Output.o"
41   cleanJob
42 }
43 trap ERR

45 function sendMsg(){
46   curl -X POST -H 'Content-Type: application/json' -d "{\"
        chat_id\": \"$chatId\", \"text\": \"$1\", \"
        disable_notification\": true}"  "https://api.telegram.
        org/$botKey/sendMessage"
47 }

49 mkdir -p "${sparkLogPath}"

51 sendMsg "Iniciando submetendo jar. ID = ${SLURM_JOB_ID}"

53 SECONDS=0
```

```
55 srun singularity exec \
56   --bind=/scratch:/scratch \
57   --bind=/var/spool/slurm:/var/spool/slurm \
58   --bind=$sparkLogPath:/opt/apache-spark/logs \
59   --bind=$localJob:/opt/apache-spark/work/ \
60   $sparkSimg spark-submit $sparkParameters --class $class
       $jarFile $jarParameters

62 retCode=$?
63 if [[ "$retCode" -ne 0 ]]; then
64   sendErr
65   exit 1
66 fi

68 duration=$SECONDS
69 FinalTime=$(convertsecs $duration)
70 sendMsg "Job finalizado em $FinalTime ou $duration segundos"
71 tar -xcf resultado.tgz $localJob
72 sendFile "resultado.tgz"
```

Nas linhas 15, 16, 17 e 18 tem-se definições de variáveis e parâmetros que serão utilizadas pelo Spark.

Nas linhas 71 e 72 tem-se o envio dos resultados do experimento em sí. No caso nosso output da aproximação do PI é apenas uma linha em um arquivo enviado ao stdout. Mas caso o resultado fossem muitas imagens, datasets ou algum arquivo diferente ainda assim seria enviado a nós.

```
1 [local]$ scp ./SlurmSparkPIApproximation.sh sms:/home/
     usuario_corrente/
2 [sms]$ sbatch SlurmSparkPIApproximation.sh
```

Após o envio acompanha-se a execução na interface web ou ir tomar um café e aguardar o envio dos resultados no Telegram. Caso dê algum erro, o envio já foi configurado.

7 CONCLUSÃO

Este trabalho apresentou um passo a passo detalhado de como criar uma infraestrutura utilizando OpenHPC. Os passos podem ser replicados praticamente em qualquer ambiente apenas com poucas modificações. Esses ambientes podem ser possivelmente: máquinas virtuais, múltiplos containers, uma sala cheia de computadores e caso seja possível compilar os pacotes do OpenHPC para ARM, até mesmo centenas de RaspberryPIs.

Também mostrou-se como o OpenHPC utiliza um conceito parecido com LEGO^{TM}, no sentido de que tudo é construído a partir de múltiplos pequenos pedaços que formam um todo.

Além de ser entediante, descrever uma instalação passo a passo que utiliza centenas de comandos Linux em um TCC no padrão ABNT mostrou-se uma péssima ideia.

7.0.1 Trabalhos futuros

Existem muitas melhorias a serem realizadas. A mais crítica é o kernel do CentOS não suportar as novas versões da libc, o que impede a execução de containers que dependem de kernels mais recentes. Para resolver esse problema é necessário ou atualizar o CentOS, o que já foi feito uma vez, ou revisitar toda a instalação do OpenHPC e utilizar alguma rolling release, como Arch Linux, que sempre estará o mais atualizada possível. Mas para isso, boa parte (se não toda) a pilha de softwares irá mudar.

Também é interessante que sejam criados scripts para a instalação e atualização automática de toda a HPC-nee para o ambiente da UFSCar.

REFERÊNCIAS

1 SEVERANCE, K. D. C. *High Performance Computing (RISC Architectures, Optimization & Benchmarks)*. 2. ed. Rice University, Houston, Texas: Centro de Documentação e Disseminação de Informações. Fundação Intituto Brasileiro de Geografia e Estatística, 1993. Acesso em: 2021-04-15.

2 ALMEIDA, S. An introduction to high performance computing. *International Journal of Modern Physics A*, v. 28, p. 40021–, 09 2013.

3 OPENSUSE. *What is a Linux Cluster?: Answer from SUSE Defines*. 2018. Disponível em: <https://susedefines.suse.com/definition/linux-cluster/>.

4 SCHULZ, K. W. et al. Cluster computing with openhpc. 2016.

5 THE OPEN GROUP. *The UNIX Standard*. 2016. Disponível em: <https://www.opengroup.org/membership/forums/platform/unix>.

6 LWN.NET. *The LWN.net Linux Distribution List*. Disponível em: <https://lwn.net/Distributions/>.

7 CHROOT. Disponível em: <https://wiki.archlinux.org/title/Chroot>.

8 GOLDEN, B. *Virtualization for dummies*. Canada: Wiley Publishing, 2008.

9 APACHE. *Apache Spark*. Disponível em: <https://spark.apache.org/>.

10 OPENHPCCOMMUNITY. *Manual De Instalação OpenHPC*. 2016. Disponível em: <https://github.com/openhpc/ohpc/releases/download/v2.0.GA/Install_guide-CentOS8-xCAT-Stateless-SLURM-2.0-x86_64.pdf>.

11 HUSODO, S. et al. Slicing and dicing openhpc infrastructure: Virtual clusters in openstack. In: . New York, NY, USA: Association for Computing Machinery, 2019. (PEARC '19). ISBN 9781450372275. Disponível em: <https://doi.org/10.1145/3332186.3332214>.

12 PÉREZ, F.; SÁNCHEZ, P. Introducción al uso de sistemas de cálculo con openhpc y sistemas de almacenamiento masivo de datos lustre. Consejo Superior de Investigaciones Científicas (España), 2020.

13 HIGGINS, J.; AL-JODY, T.; HOLMES, V. *Rapid Deployment of Bare-Metal and In-Container HPC clusters using OpenHPC playbooks*. [S.l.], 2018.

14 XCAT. *Extreme Cloud Administration Toolkit*. Disponível em: <https://xcat-docs.readthedocs.io/en/stable/>.

15 WEIL, S. A. et al. Rados: A scalable, reliable storage service for petabyte-scale storage clusters. In: *Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07*. New York, NY, USA: Association for Computing Machinery, 2007. (PDSW '07), p. 35–44. ISBN 9781595938992. Disponível em: <https://doi.org/10.1145/1374596.1374606>.

16 WEIL, S. et al. Ceph: A scalable, high-performance distributed file system. In: *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06)*. [S.l.: s.n.], 2006.

17 NAGIOS. *The Industry Standard In IT Infrastructure Monitoring*. Disponível em: <https://www.nagios.org/>.

18 GANGLIA Monitoring System. Disponível em: <http://ganglia.info/>.

19 CEA-HPC. *cea-hpc/clustershell*. Disponível em: <https://github.com/cea-hpc/clustershell>.

20 CHAOS. *genders*. Disponível em: <https://github.com/chaos/genders>.

21 DUN. *dun/conman*. Disponível em: <https://github.com/dun/conman>.

22 KURTZER, G. M.; SOCHAT, V.; BAUER, M. W. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, Public Library of Science, v. 12, n. 5, p. 1–20, 05 2017. Disponível em: <https://doi.org/10.1371/journal.pone.0177459>.

23 SYLABS. *Introduction to Singularity*. Disponível em: <https://sylabs.io/guides/3.7/user-guide/introduction.html>.

24 PRIEDHORSKY, R.; RANDLES, T. Charliecloud: Unprivileged containers for user-defined software stacks in hpc. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: Association for Computing Machinery, 2017. (SC '17). ISBN 9781450351140. Disponível em: <https://doi.org/10.1145/3126908.3126925>.