UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Lucas Micol Policarpo

UMA ABORDAGEM PARA ORQUESTRAÇÃO DE FUNÇÕES DE APRENDIZADO DE MÁQUINA VIRTUALIZADAS EM CONTAINERS

Santa Maria, RS
2019

Lucas Micol Policarpo

UMA ABORDAGEM PARA ORQUESTRAÇÃO DE FUNÇÕES DE APRENDIZADO DE MÁQUINA VIRTUALIZADAS EM CONTAINERS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da computação**.

ORIENTADOR: Prof. Celio Trois

COORIENTADOR: Prof. João Carlos Damasceno Lima

TG 461
Santa Maria, RS
2019

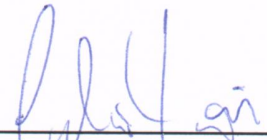
End. Eletr.: lmpolicarpo@inf.ufsm.br

Lucas Micol Policarpo

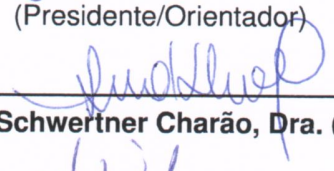
UMA ABORDAGEM PARA ORQUESTRAÇÃO DE FUNÇÕES DE APRENDIZADO DE MÁQUINA VIRTUALIZADAS EM CONTAINERS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da computação**.

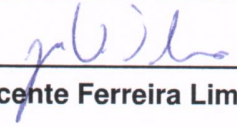
Aprovado em 2 de dezembro de 2019:



Celio Trois, Dr. (UFSM)
(Presidente/Orientador)



Andrea Schwertner Charão, Dra. (UFSM)



João Vicente Ferreira Lima, Dr. (UFSM)

Santa Maria, RS
2019

RESUMO

UMA ABORDAGEM PARA ORQUESTRAÇÃO DE FUNÇÕES DE APRENDIZADO DE MÁQUINA VIRTUALIZADAS EM CONTAINERS

AUTOR: Lucas Micol Policarpo
ORIENTADOR: Celio Trois
COORIENTADOR: João Carlos Damasceno Lima

O aprendizado de máquina é uma das áreas que mais cresce na computação, um dos fatores para esse crescimento é que a utilização de aplicações inteligente vem se tornarem cada vez mais comum nos nossos dias. Junto ao aumento do poder computacional dos dispositivos a nossa volta, foi possível o desenvolvimento de novas aplicações capazes de obter um melhor resultado na criação e execução de uma inteligência artificial, com a utilização de sistemas distribuídos somos capazes de separar as funções relativas ao aprendizado de máquina a fim de obter melhor desempenho. Entretanto, podem ocorrer inconsciências na execução distribuída de determinadas tarefas, uma vez que funções necessitam de bibliotecas específicas para serem executadas. Uma abordagem que soluciona esse tipo de problema são as máquinas virtuais e os *containers*, que podem ser executados como serviços de instanciação utilizando parte do processamento da máquina do usuário para virtualizar um processo. Este trabalho apresenta uma abordagem para virtualização de funções de aprendizado de máquina com a utilização de *containers dockers*, fazendo uso de um sistema distribuído na borda da rede para processamento e considerando a capacidade de cada um dos nós do sistema.

Palavras-chave: Sistemas de Computação, Aprendizado de Máquina, Sistemas distribuídos, Containers, Virtualização de Funções

ABSTRACT

AN APPROACH FOR ORCHESTRATING VIRTUALIZED MACHINE LEARNING FUNCTIONS ON CONTAINERS

AUTHOR: Lucas Micol Policarpo
ADVISOR: Celio Trois
CO-ADVISOR: João Carlos Damasceno Lima

Machine learning is one of the fastest growing areas in computing, one of the factors behind this growth is that smart application utilization is becoming increasingly common today. Along with the increased computing power of the devices around us, it has been possible to develop new applications capable of achieving better results in the creation and execution of artificial intelligence. Using distributed systems we are able to separate the functions related to learning from machine for best performance. However, unconsciousness may occur in the distributed execution of certain tasks as functions require specific libraries to be performed. One approach that solves this type of problem is virtual machines and *containers*, which can be run as instantiation services using part of user machine processing to virtualize a process. This paper presents an approach to virtualizing machine learning functions using container dockers, making use of a distributed system at the network edge for processing and considering the capacity of each node of the system.

Keywords: Computer Systems, Machine Learning, Distributed Systems, Containers, Function Virtualization

LISTA DE FIGURAS

Figura 2.1 – Arquitetura RPC 11
Figura 2.2 – Arquitetura Computação de Borda e Nuvem 14
Figura 2.3 – Arquitetura de Virtualização. 15
Figura 2.4 – Arquitetura de Container Docker. 16
Figura 3.1 – Arquitetura do Zoo 17
Figura 3.2 – Visão geral do Dask 18
Figura 3.3 – Arquitetura MLFV 19
Figura 4.1 – Arquitetura Proposta 21
Figura 4.2 – Passo a passo do registro de um cliente até a execução de uma requisição 22
Figura 4.3 – Exemplo de execução de duas requisições com necessidade de instalação de bibliotecas 23
Figura 5.1 – Visão global das Etapas e Funções de ML 24
Figura 5.2 – Cadeias MLFV 26
Figura 5.3 – Sobrecarga imposta pelas abordagens em comparação com um ambiente Local com docker inicializando do zero 28
Figura 5.4 – Sobrecarga imposta pelas abordagens em comparação com um ambiente Local com docker não instanciado na máquina do host 29
Figura 5.5 – Sobrecarga imposta pelas abordagens em comparação com um ambiente Local com docker já instanciado na máquina do host 29
Figura 5.6 – Múltiplas requisições simultâneas 30

LISTA DE ABREVIATURAS E SIGLAS

CPU	Unidade central de processamento (Central Processing Unit)
DT	Árvore de decisão
KNN	Algoritmo K vizinhos maispróximos
ML	Aprendizado de Máquina (Machine Learning)
$MLaaS$	Aprendizado de Máquina como Serviço (Machine Learning as a Service)
MLP	Perceptron multicamadas
$MLFV$	Machine Learning Function Virtualization
NFC	Encadeamento de funções de rede (Network Function Chaining)
NFV	Virtualização de Função de Rede (Network Function Virtualization)
IOT	Internet das Coisas (Internet of Things)
RF	Algoritmo Floresta aleatória
RPC	Chamada de procedimento remoto (Remote Procedure Call)
SVM	Máquina de vetores de suporte
VM	Máquina Virtual
XDR	External Data Representation

SUMÁRIO

1 INTRODUÇÃO 8
2 FUNDAMENTAÇÃO TEÓRICA 10
2.1 APRENDIZADO DE MÁQUINA 10
2.2 CHAMADA DE PROCEDIMENTO REMOTO (RPC) 11
2.3 COMPUTAÇÃO NA NUVEM 12
2.3.1 Aprendizado de máquina como Serviço 13
2.4 COMPUTAÇÃO DE BORDA 14
2.5 VIRTUALIZAÇÃO 15
2.5.1 Máquinas Virtuais 15
2.5.2 Containers 16
3 TRABALHOS RELACIONADOS 17
3.1 ZOO 17
3.2 DASK 18
3.3 MLFV 18
4 ORQUESTRAÇÃO DE FUNÇÕES DE APRENDIZADO DE MÁQUINA EM CONTAINERS DOCKER 20
4.1 ARQUITETURA PROPOSTA 20
4.2 DO REGISTRO À EXECUÇÃO 22
5 TESTES E RESULTADOS 24
5.1 METODOLOGIA 24
5.1.1 Etapa de preparação 25
5.1.2 Etapa de treinamento 25
5.1.3 Etapa de classificação 25
5.2 CADEIAS DE FUNÇÕES 26
5.3 IMPLEMENTAÇÃO 27
5.4 TESTES 27
6 CONCLUSÃO 32
6.1 TRABALHOS FUTUROS 32
REFERÊNCIAS BIBLIOGRÁFICAS 33

1 INTRODUÇÃO

Com o crescimento dos dispositivos móveis e de tecnologias como Internet das Coisas (IoT) e Cidades Inteligentes, surgem novas oportunidades para criação e utilização de uma variedade de computadores e sensores, permitindo, também, o desenvolvimento de aplicações que gerenciam esses equipamentos. De acordo com a Fundação Getúlio Vargas (Fundação Getúlio Vargas, 2019), no ano de 2019 possuímos mais de 420 milhões de dispositivos digitais no Brasil, sendo destes, 215 milhões de dispositivos móveis. Junto a isso, o aumento da capacidade desses aparelhos e a necessidade dos usuários por conectividade de alta velocidade, sempre ativa e orientada a multimídia, resulta em uma enorme quantidade de dados a serem transmitidos (ODLYZKO, 2003).

Com isso, diferentes técnicas surgiram para auxiliar no processamento e extração de informações desses dados. Técnicas como Aprendizado de Máquina, utilizam diferentes ferramentas e métodos matemáticos para criar uma aproximação do perfil do usuário, assim, tendo uma visualização mais precisa de quem é o consumidor (BERRY; LINOFF, 2004). Essas informações podem ser usadas mais tarde para distribuição de anúncios especializados de acordo com o perfil da pessoa. Esse tipo de processamento possui um cronograma de atividades bem definidas que pode ser separado e processado em paralelo em busca de maior desempenho na sua execução.

Técnicas como a Chamada de Procedimento Remoto (RPC) (THURLOW, 2009) possibilitam o compartilhamento de hardware local para realização desse tipo de atividade. Outra técnica muito utilizada para processamento é a Computação em Nuvem, onde os dados são enviados para um servidor para execução. Contudo, nem sempre os dados podem ser enviados para um servidor com grande poder computacional, uma vez que o custo de envio pela rede pode ser muito alto dependendo do tamanho de nossos dados. Com isso, surgiu a computação de borda, que tenta reduzir os gastos de envio dos dados, buscando executar os procedimentos o mais próximo do usuário.

Uma dessas técnicas é o Dask (ROCKLIN, 2015) que realiza distribuição de processamento computacional em Pyhton. O Dask realiza planejamento dinâmico de tarefas para otimizar as execuções de funções. Outro sistema que efetua distribuição de processos, focado no âmbito do aprendizado de máquina, é o MLFV (SOUZA, 2019), que trabalha utilizando Aprendizado de Maquina como Serviço (*MLaaS*) para execução distribuída das tarefas, e também propõe uma forma de separar as cadeias de funções de maneira a buscar maior desempenho. Entretanto, determinados procedimentos necessitam de bibliotecas e pacotes específicos para serem executados, o que conflita com a crescente heterogeneidade dos dispositivos computacionais.

Uma das soluções que pode ser usada para resolver o problema das bibliotecas são as Máquinas Virtuais (GOLDBERG, 1974). Essa proposta busca encapsular todo um sis-

tema para a execução, de forma que os pacotes necessários estejam agrupados com a máquina virtual. Porém, máquinas virtuais podem exigir muitos recursos em um computador e acabar se tornando muito pesadas de executar. Com isso, surgiu uma outra abordagem para virtualização, a virtualização em nível de sistema operacional, os containers (CONTAINERS, 2009). A ideia dos containers é diminuir as camadas existentes nas máquinas virtuais, tornando o sistema virtualizado mais leve e com menor sobrecarga (JOY, 2015). Um desses sistemas de containers é o Docker (DOCKER Inc., 2013), que apresenta vantagens como instanciação rápida e pouca utilização de recursos do hospedeiro.

Levando em conta a variedade de dispositivos à nossa volta e a crescente demanda por processamento, este trabalho apresenta uma proposta de virtualização e orquestração de funções de aprendizado de máquina via containers utilizando computação de borda, prevenindo possíveis erros causados pela ausência de pacotes ou incompatibilidade de hardware, levando em consideração o estado e a capacidade computacional dos nós associados ao sistema. Portanto, o sistema a ser proposto, tem como objetivo virtualizar funções de maneira controlada para não sobrecarregar os computadores e nem a rede os interliga, assim como prevenir possíveis erros de incompatibilidade de bibliotecas com a utilização de virtualização via Docker, visando obter um melhor desempenho na execução de tarefas.

Este trabalho irá apresentar um comparativo com outras abordagens de processamento distribuído, tais como Azure, Dask e MLFV. Também serão apresentados os resultados das execuções nas máquinas locais e a sobrecarga do sistema em comparação com as abordagens similares.

O restante do texto está organizado da seguinte maneira. O capítulo 2 apresentará o referencial teórico utilizado para a confecção deste trabalho, abordando temas como aprendizado de máquina, distribuição de processamento, computação em nuvem, entre outros. No capítulo 3 serão comentados os trabalhos relacionados e a arquitetura dos sistemas correlatos. O capítulo 4 é onde será apresentada a proposta, bem como, sua arquitetura e o passo a passo da execução de uma chamada de execução. O capítulo 5 apresentará a forma como os testes foram realizados e os resultados obtidos com os testes. Por fim o capítulo 6 conclui o texto junto com as ideias para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para a confecção deste trabalho foi realizada uma revisão bibliográfica na literatura, visando conhecer as técnicas disponíveis e as tecnologias que poderiam facilitar o desenvolvimento da proposta. Primeiramente na seção 2.1 são apresentadas as técnicas de aprendizado de máquina, seguindo para a seção 2.2 onde será mostrado o método da chamada de procedimento remoto, uma forma de compartilhamento de recursos. Na seção 2.3 será mostrado o paradigma de computação na nuvem, junto ao aprendizado de máquina como serviço. Seguido das seções 2.4 que abordará a computação na borda da rede e por fim a seção 2.5 onde será apresentado técnicas de virtualização.

2.1 APRENDIZADO DE MÁQUINA

O conceito de aprendizado de máquina foi criado com o intuito de desenvolver técnicas e algoritmos capazes de obter conhecimento de forma automática, possibilitando a criação de sistemas inteligentes aptos a tomarem decisões e capazes de aprender com as suas escolhas (NILSSON, 1996). Essa área trata-se de um ramo dentro da inteligência artificial normalmente associado à tarefas como reconhecimento de fala, controle de robôs, predições, etc. Podemos dividir os algoritmos de aprendizado de máquina em duas grandes categorias:

- Algoritmos supervisionados: Os supervisionados possuem um conjunto de entradas para treinamento, bem como as saídas (respostas corretas) esperadas. Assim, os modelos irão se ajustando durante a execução e mapeando os valores de entrada com os de saída.
- Algoritmos não supervisionados: Já nesse caso, os dados são agrupados por padrões próprios que os algoritmos reconhecem em tempo de execução, sem que haja uma única resposta possível para o agrupamento.

Algumas dessas técnicas são: Regressão linear, Máquina de vetores de suporte (SVM), Árvore de decisão (DT), Floresta aleatória (RF), Mapas auto-organizáveis, K vizinhos mais próximos (KNN), Perceptron multicamadas (MLP), etc.

Um dos conceitos que surgiram com o aprendizado de máquina foi o do neurônio artificial. Donald Hebb (HEBB, 1949), foi o pesquisador que introduziu o primeiro método de treinamento para um neurônio artificial, demonstrando que ao modificar os pesos das entradas é possível alterar o modelo de aprendizado. Foi então com esse conceito de neurônio que surgiram as redes neurais.

As redes neurais artificiais são compostas por conjuntos de camadas de neurônios artificias, sendo a primeira a camada as entradas, as camadas intermediárias os neurônios da rede, e a camada final as saídas. Essa forma de aprendizado conecta as camadas entre sí e utiliza diferentes formas de regressão e combinação matemática para gerar um classificador (TU, 1996).

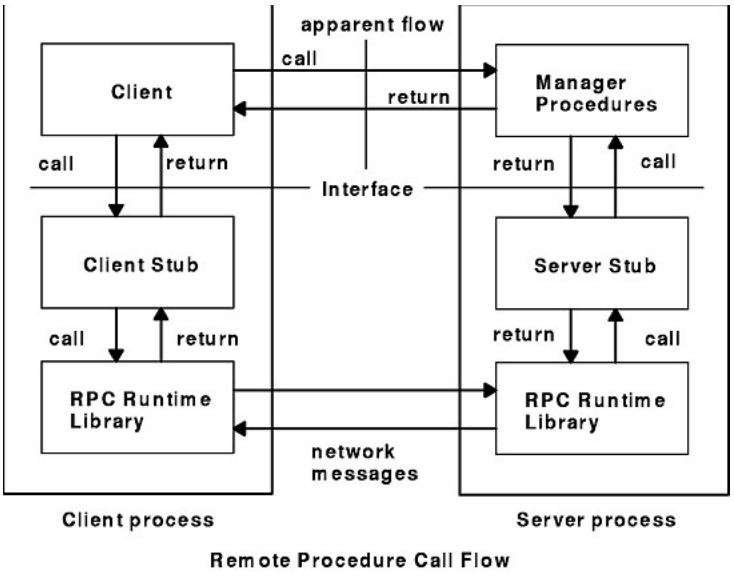
Outra forma de aprendizado de máquina é o método de aprendizado profundo (LECUN; BENGIO; HINTON, 2015), onde as entradas passam por uma série de pré-processamento e refinamento antes de serem entregues às camadas de treinamento, assim, predições como reconhecimento de fala ou de imagens se tornam mais precisos (LECUN; BENGIO; HINTON, 2015).

Com o custo computacional elevado para o treinamento e classificação das diferentes formas de aprendizado de máquinas, surgiu a oportunidade de utilizar algumas técnicas de compartilhamento de recursos para distribuição de tarefas entre diferentes computadores, uma das abordagens foi a chamada de procedimento remoto (RPC).

2.2 CHAMADA DE PROCEDIMENTO REMOTO (RPC)

Um dos trabalhos a propor compartilhamento de recursos via internet foi a especificação RFC707 (WHITE, 1975), esta padronizou um modelo de comunicação e troca de mensagens para chamadas remotas através de um framework de alto nível. Sendo este, o RPC, a primeira versão desta arquitetura pode ser visualizada na Figura 2.1

Figura 2.1 – Arquitetura RPC



Fonte: (IBM, 2019)

Analisando a Figura 2.1 pode-se notar que a proposta do RPC não é somente a de

compartilhamento de recursos. A ideia por trás dessa tecnologia é abstrair do usuário a conexão entre ele e o Servidor/Gestor de Processos. Nessa lógica o cliente se comunica com um método que encapsula as informações necessárias em sua máquina, esse método é o Stub. Após isso, o Stub, envia os dados para o Stub do servidor, e no servidor, o Stub do servidor desempacota e gerencia os dados recebidos, abstraindo a comunicação entre eles. Porém, tanto para o usuário como para o servidor, essas chamadas ao Stub acontecem de modo indireto. Contudo, para ambos a comunicação aparenta estar acontecendo sem intermediários.

Uma das implementações mais conhecidas dessa especificação é o *Open Network Computing Remote Procedure Call* (ONC RPC) (DEUTSCH; GAILLY, 1996). Essa implementação se baseia nas chamadas de procedimento usadas no Unix e na linguagem C e serializa os dados com External Data Representation (XDR) (SRINIVASAN, 1995). A maioria dos sistemas baseados em Unix contêm essa implementação. O ONC RPC trata-se de uma implementação mais enxuta do RPC e é muito utilizado em ambiente de comunicação Local (WAN's). A especificação mais atual dessa implementação é a RFC5531 (THURLOW, 2009)

A principal vantagem que o ONC RPC traz a um sistema é a possibilidade de compartilhamento de CPU entre computadores na mesma rede, de maneira prática e garantindo a integridade do sistema devido as verificações de segurança da última especificação.

Junto ao crescimento do poder computacional e das técnicas de divisão de processamento surgiram novas formas de processamento distribuído. Como é o caso dos grandes centros de processamentos de dados, que forneciam suas máquinas para outros usuários computarem seus dados, surgindo assim o paradigma de computação na nuvem.

2.3 COMPUTAÇÃO NA NUVEM

O sistema de computação na nuvem traz ao usuário uma conexão com um servidor de processamento, o que garante a ele um maior poder para processar seus dados. Esse método é recente e inovador, uma vez que o usuário somente precisa transportar seus dados até a nuvem para poder utilizar o máximo da capacidade disponível (JOSEP et al., 2010). A computação na nuvem traz ao usuário vantagens como a elasticidade do serviço, grande quantidade de recursos computacionais e elimina a necessidade de aquisição de um equipamento físico (TAURION, 2009). Assim o consumidor final apenas paga pelo que for usado e ainda não precisa gastar com novos equipamentos.

Com a crescente demanda do mercado por aplicações inteligentes (CHAN et al., 2013) para realização de tarefas do dia a dia (tais como comandar veículos autônomos, detecção e tradução de fala, robôs inteligentes, sistemas de monitoramento de saúde móvel, recomendação de produtos, etc.) os servidores de alta capacidade de processamento

se adaptaram, e começaram a disponibilizar serviços de aprendizado de máquina.

2.3.1 Aprendizado de máquina como Serviço

De acordo com Chan (CHAN et al., 2013), a escolha de um algoritmo específico de ML para o desenvolvimento de um software pode ser algo demorado. Uma vez que existem inúmeras técnicas, frameworks e ajustes a serem efetuados, essa tarefa acaba sendo algo complicado. Com a ideia de abstrair do usuário a complexidade por trás dos diversos algoritmos e prover, até mesmo para o usuário mais leigo, a capacidade de utilizar essas ferramentas, foram criados os serviços de aprendizado de máquina (RIBEIRO; GROLINGER; CAPRETZ, 2015).

Esses serviços comportam um conjunto de equipamentos de treinamento para inteligência artificial. Alguns funcionam até mesmo como "caixa preta", onde não temos acesso a como os dados são treinados.

Yao et al. (YAO et al., 2017), em seu trabalho, comparam seis plataformas de MLaaS, que incluem Amazon Machine Learning, Google Prediction API, Microsoft Azure ML Studio, Automatic Businesse Modeler, BigML, PredictionIO e um Cliente local totalmente customizado utilizando bibliotecas de ML. Seus resultados apresentam a relação custo-benefício entre os sistemas de MLaaS e técnicas de ML aplicadas localmente em apenas um computador. Com isso demonstraram que a utilização de sistemas locais permite uma maior e melhor customização das tarefas e técnicas de ML, dependendo apenas dos softwares e hardwares suficientes para execução. Embora tenham avaliado conjuntos de dados de vários tamanhos, eles não apresentaram resultados comparando o tempo de execução de cada plataforma.

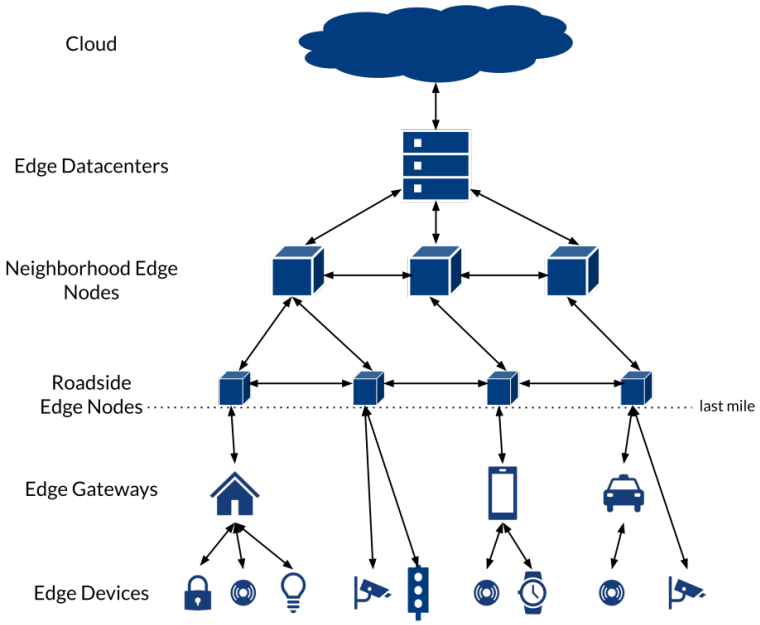
De acordo com uma pesquisa realizada em 2018 (DATAREPORTAL, 2018), possuímos no mundo mais de 4 bilhões de usuários conectados na internet, e em 2019 a cada segundo são gerados mais de 80 mil GB de dados pela internet (Internet Live Stats, 2019). Visto que grande parte desses dados são levados para serem processados na nuvem, podemos afirmar que isso gera uma sobrecarga imensa para as redes de comunicação.

Junto a pouca customização disponível nos serviços de Aprendizado de máquina e a quantidade de informação que sobrecarrega as redes de comunicação, surge uma proposta para reduzir essa quantidade de dados e distribuir o processamento de maneira local: a Computação de Borda.

2.4 COMPUTAÇÃO DE BORDA

Essa técnica tenta reduzir a quantidade de dados a serem transportados pela rede realizando o processamento dos dados o mais próximo do usuário. Esse método prevê a utilização de dispositivos de IoT, visto que os controladores e microcomputadores vêm se tornando cada vez menores e com maior capacidade de processamento, fazendo com que muito de sua capacidade seja desperdiçada. Nessa abordagem, a ideia é utilizar os dispositivos a nossa volta para fazerem o processamento de determinada tarefa, sem termos a necessidade de enviar dados para a nuvem processar.

Figura 2.2 – Arquitetura Computação de Borda e Nuvem



Fonte: (Inovex Blog, 2019)

Como pode ser visto na Figura 2.2, a computação de borda utiliza dos mais diversos computadores para aliviar a sobrecarga da rede, assim, só enviando chamadas processadas para as camadas mais acima, como a nuvem. Outra vantagem da computação de borda é a latência em relação aos uso de servidores. Muitos servidores podem falhar, ou não estarem funcionando da maneira correta, o que acarreta em um tempo de resposta muito alto.

Com base nisso, a utilização de computação de borda se torna útil para a proposta do trabalho, uma vez que poderíamos utilizar nossos computadores de maneira interligada para realização do processamento distribuído, sem precisarmos enviar os dados pela rede até um grande servidor de processamento. Porém, tanto na nuvem como na borda, podemos ter problemas com a heterogeneidade dos dispositivos disponíveis, fazendo com que determinado serviço que pretendíamos executar não funcione da maneira esperada em outro computador. Esse tipo de problema é comum, uma vez que determinadas aplicações

necessitam de bibliotecas e pacotes específicos para serem executadas. Uma abordagem que surge para resolver esse problema é a Virtualização.

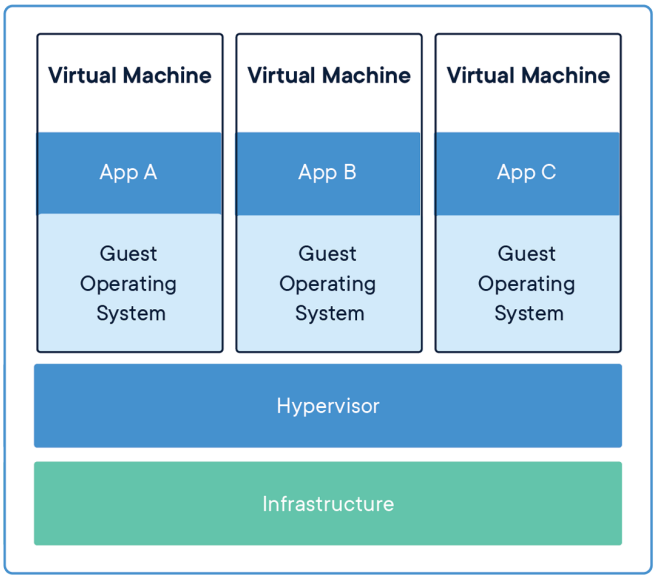
2.5 VIRTUALIZAÇÃO

Uma das formas de solucionar o problema citado acima é a realização da virtualização para execução dos serviços. A técnica de virtualização é datada de 1964 quando a IBM iniciou um projeto que deu forma ao termo Máquina Virtual (CREASY, 1981), sendo essa a primeira técnica de virtualização que surgiu na computação.

2.5.1 Máquinas Virtuais

Uma máquina virtual é uma simulação de um sistema computacional, essa simulação, normalmente, implementa desde o hardware até software do dispositivo, no quesito de custo computacional, tendem a ser caras e lentas de executar dependendo de alto poder de processamento do dispositivo hospedeiro. Em contrapartida, as máquinas virtuais possuem alta compatibilidade, uma vez que independente de onde sejam executadas, os processos internos irão se comportar da mesma forma em todos os dispositivos (SMITH; NAIR, 2001). Sua arquitetura pode ser visualizada na Figura 2.3.

Figura 2.3 – Arquitetura de Virtualização.



Fonte: (DOCKER Inc., 2013).

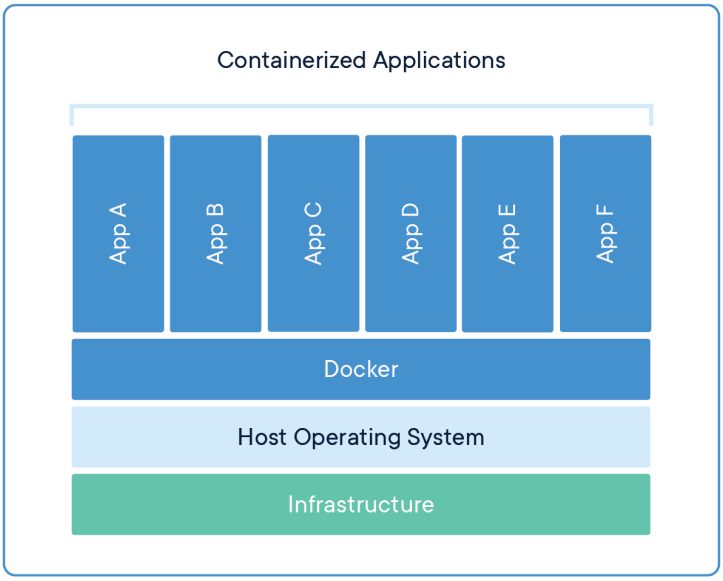
Para a abordagem proposta, máquinas virtuais se tornam muito custosas, já que a ideia do trabalho é utilizar o poder computacional de diversos dispositivos, não podemos

presumir que todos os aparelhos conectados em nosso sistema são capazes de executar uma máquina virtual. Além disso, máquinas virtuais podem demandar uma configuração prévia da imagem de virtualização para a execução correta de determinado procedimento. Para solucionar esses problemas, possuímos um tipo de virtualização mais leve, também conhecida como virtualização em nível de sistema operacional.

2.5.2 Containers

A abordagem de containers surge com a ideia de encapsulamento de execução e isolamento de núcleo, trazendo segurança na hora da execução de tarefas e uma velocidade maior para a instanciação da virtualização.

Figura 2.4 – Arquitetura de Container Docker.



Fonte: (DOCKER Inc., 2013).

Como pode ser visualizado na Figura 2.4, os containers executam sobre o sistema operacional do hospedeiro, porém não possuem acesso direto como super usuário, tornando assim, a execução segura tanto para o usuário quanto para o hospedeiro. O Docker foi escolhido para essa abordagem pelo fato de ser mais prático de ser utilizado, uma vez que sua arquitetura valoriza microsserviços em busca da eficiência, além de possuir controle de versão das imagens e possibilidade de reversão de iterações.

Tendo em mente as formas de virtualização que foram apresentadas, a proposta seria utilizar containers em docker para orquestrar os serviços a serem distribuídos, assim resolvendo o problema da inconsistência das execuções dos procedimentos no caso do Dask e do MLFV. Posto isto, no Capítulo 4 será apresentado um esboço do trabalho em desenvolvimento.

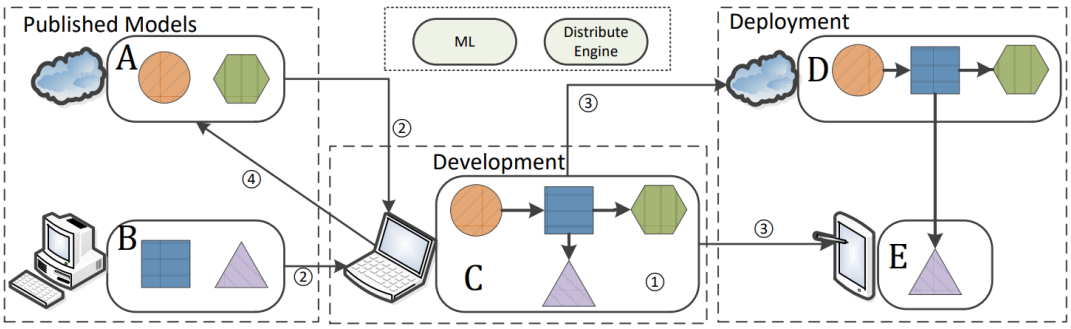
3 TRABALHOS RELACIONADOS

Tendo em mente o crescimento do número de computadores e dispositivos móveis, as abordagens de processamento em borda se expandiram. Nessa seção serão expostos três trabalhos relacionados que apresentam diferentes abordagens para distribuição de tarefas na borda da rede.

3.1 ZOO

Zoo, proposto por Zhao (ZHAO et al., 2018), apresenta uma estrutura de "Serviços compostos" para processamento de ML. A ideia apresentada é que possam ser usados conjuntos de algoritmos já existentes para execução de determinada tarefa, por exemplo, reconhecimento de fala. Assim, como pode ser visto na figura 3.1, as formas geométricas representam esses algoritmos e os nós do sistema estão conectados de maneira a compartilhar esses modelos para os repositórios na nuvem ou para as máquinas na borda.

Figura 3.1 – Arquitetura do Zoo



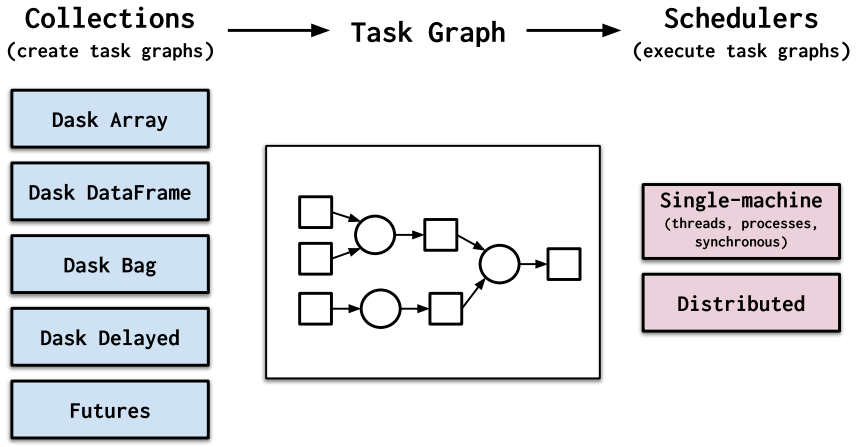
Fonte: (ZHAO et al., 2018)

O foco desse modelo é a simplificação para o usuário final, uma vez que nem todos os usuários possuem o conhecimento necessário para a estruturação e preenchimento de parâmetros de uma estrutura de ML. Os exemplos de implantação apresentados por Zoo, provam que os serviços de ML podem ser fáceis de compor e implantar utilizando a plataforma. As avaliações também comparam seu desempenho com plataformas de MLaaS e de ML na borda.

3.2 DASK

Dask é uma biblioteca de programação paralela em Python, que possui planejamento dinâmico de tarefas e flexibilidade para a execução de serviços (ROCKLIN, 2015). O Dask traz consigo a possibilidade de escalonamento para um grande número de máquinas, assim como a possibilidade de execução local. Uma de suas principais vantagens é o fato do Dask ser executado nativamente com o Python, garantindo compatibilidade independente da plataforma. A visão geral do sistema Dask pode ser visualizada na Figura 3.2.

Figura 3.2 – Visão geral do Dask



Fonte: (ROCKLIN, 2015)

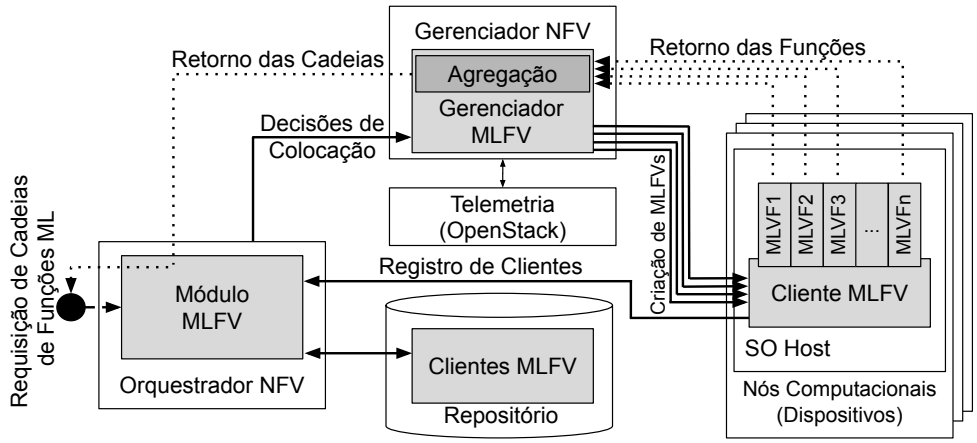
Contudo, o Dask possui algumas desvantagens. Uma destas é o fato do sistema não estar ciente do estado da rede, então ele não controla possíveis problemas na comunicação. Em caso de comunicação reduzida, por um ataque de negação de serviço, por exemplo, o gerenciador dos serviços pode continuar enviando pacotes a nós que estejam falhos.

3.3 MLFV

Um outro método que surgiu recentemente que soluciona o problema citado do Dask é o MLFV (SOUZA, 2019). Utilizando os conceitos da Virtualização de Funções de Rede (NVF), o MLFV apresenta uma abordagem de virtualização de funções focada no aprendizado de máquina. Criando uma cadeia para separação e execução das funções requisitadas, a abordagem apresenta uma proposta de distribuição de funções de ML como serviço (MLaaS). Ao mesmo tempo se apresenta ciente do comportamento da rede em que estiver sendo executada para prevenir possíveis problemas de comunicação com seus nós

vinculados. A arquitetura proposta do sistema pode ser visualizada na Figura 3.3.

Figura 3.3 – Arquitetura MLFV



Fonte: (SOUZA, 2019)

O MLFV propõem a ideia de virtualização de funções de aprendizado de máquina (MLFC), onde a virtualização acontece sob uma etapa da realização do treinamento de uma inteligência artificial. As funções virtualizadas são posteriormente alocadas para executar baseada em um modelo matemático disposto pelo trabalho.

Entretanto, o MLFV apresenta um problema: a incompatibilidade de bibliotecas entre os nós do sistema e as requisições a serem processadas. Dessa forma, caso haja algum erro gerado por biblioteca não instalada ou inconsistência de execução, esse erro não é tratado pelo sistema. Para prevenir que isso aconteça, existe uma relação de pacotes disponíveis nos nós, assim como uma relação de pacotes que serão utilizados pelos serviços a serem processados, o que não descarta a impossibilidade de execução de um pacote não previsto.

Nesse capítulo foram demonstradas algumas abordagens que possuem similaridades com o trabalho a ser desenvolvido. Com isso em mente, a proposta desse trabalho é a criação de uma plataforma para processamento distribuído de funções de aprendizado de máquina visando a virtualização das execuções. Essa proposta será trabalhada no capítulo que segue.

4 ORQUESTRAÇÃO DE FUNÇÕES DE APRENDIZADO DE MÁQUINA EM CONTAINERS DOCKER

Nesse capítulo será apresentado a proposta do trabalho: a orquestração de funções de aprendizado de máquina em containers docker, a forma como o sistema e a sua arquitetura estão organizados e o passo a passo que ocorre desde a inicialização de um nó cliente até a execução de uma função.

4.1 ARQUITETURA PROPOSTA

A figura 4.1 apresenta a arquitetura proposta do sistema, onde os quadrados em branco no Servidor representam os módulos, o quadrado cinza no servidor o sistema em sí, e os quadrados azuis as máquinas que estão interligadas com o sistema principal. Cada parte da estrutura possui suas respectivas funções, sendo essas:

Requisições de Cadeias de ML. O sistema recebe como entrada uma requisição de execução de função de ML, essa pode estar organizada de maneira paralela ou sequencial. Cada requisição pode conter uma ou mais funções a serem executadas e o conjunto do resultado dessas funções será a resposta do servidor ao cliente. Cada requisição corresponde a uma função de ML escrita em Python, que será executada pelo sistema.

Orquestrador. O orquestrador é responsável por armazenar os dados de cada instância conectada ao sistema, assim como deve cadastrar e remover os nós acessíveis. Além disso, também é o responsável por receber e repartir as requisições dos clientes e decidir em qual instância a requisição será executada. A tomada de decisão é feita com a utilização do algoritmo de Round Robin. Posteriormente essa decisão deve ser passada ao Gerenciador.

Hosts Disponíveis. É um banco de dados em memória que armazena as informações dos hosts associados ao sistema, tais como: bibliotecas disponíveis, número de execuções, memória, CPU, etc. Apenas o orquestrador tem acesso ao banco de dados.

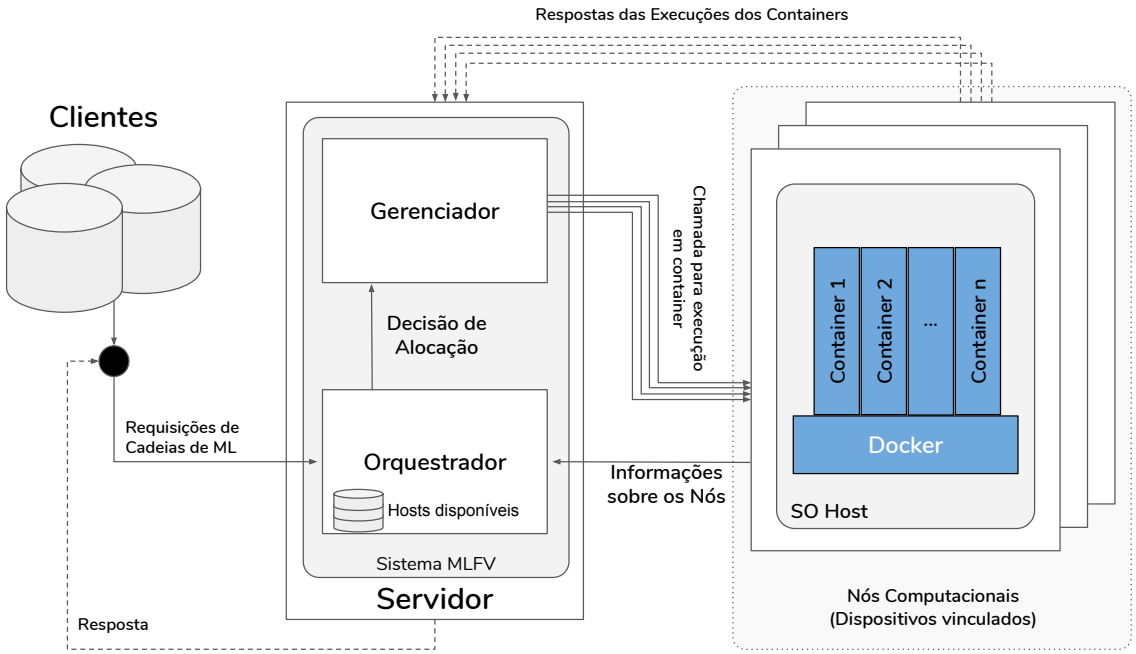
Gerenciador. O gerenciador é o responsável por realizar a coordenação das execuções após o Orquestrador decidir em qual máquina deve ocorrer o processamento e o Gerenciador encaminha para o devido container Docker a função correspondente. Além disso, caso o container não seja compatível com a execução de determinada chamada, o Gerenciador é quem emite o sinal para instalação da biblioteca necessária, e só depois libera a função para executar. Também é o encarregado de agrupar a resposta e retornar ao cliente que requisitou a execução.

Sistema MLFV. É o coordenador dos módulos **Orquestrador** e **Gerenciador**. Na visão do cliente que requisita uma execução, a comunicação ocorre de maneira direta com

o Sistema, sem que este necessariamente conheça os módulos e os hosts.

Hosts. São os provedores de processamento do sistema MLFV. São neles que os containers conectados ao sistema estão instanciados e esses serão os responsáveis por executar as funções que forem requisitadas pelo gerenciador. Em um mesmo computador host pode existir uma ou mais instâncias docker do sistema host, que irá se comunicar com o Sistema MLFV.

Figura 4.1 – Arquitetura Proposta



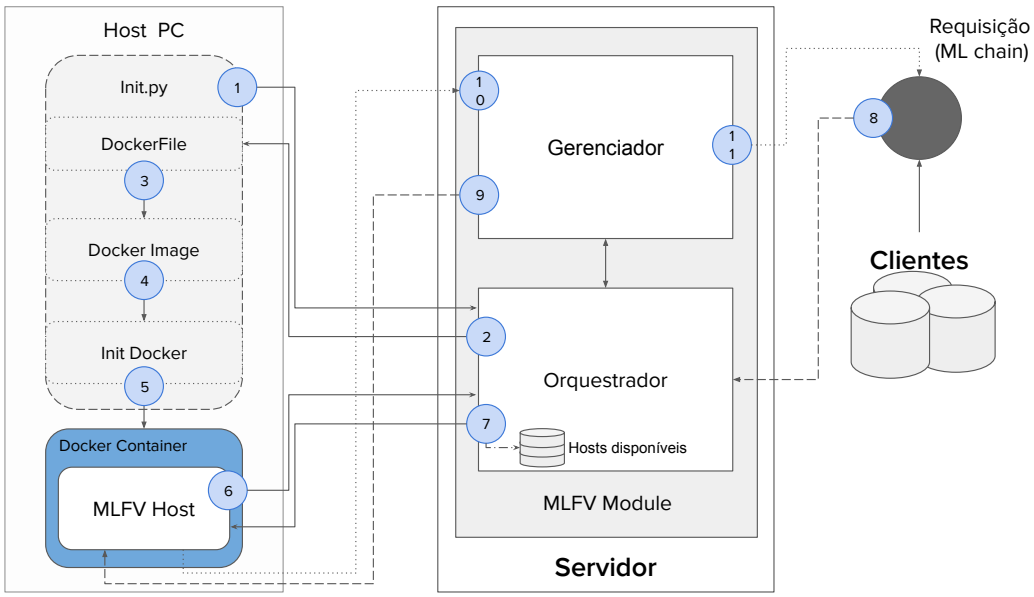
Fonte: O próprio autor

O Docker está sendo utilizado para prevenir erros de incompatibilidade de bibliotecas. O dockerfile para instanciação da virtualização por padrão vem com o mínimo de pacotes possíveis, justamente para diminuir a sobrecarga. A proposta é que o usuário passe ao sistema as bibliotecas que sua função irá utilizar e assim o gerenciador instale os pacotes em uma imagem docker em execução. A imagem é salva no sistema para diminuir a sobrecarga do host para que não seja necessário a geração de outro arquivo. Fora isso, o cliente também poderá passar ao sistema um dockerfile, caso queira executar a partir da que ele definiu. O mesmo é válido para os hosts, uma vez que estiverem executando uma determinada imagem, essa imagem fica salva em sua máquina e pode ser usada posteriormente em outra requisição que seja compatível.

4.2 DO REGISTRO À EXECUÇÃO

Para o funcionamento do sistema, tanto as máquinas hosts como o servidor, precisam estar executando corretamente. Nesta seção será apresentado o que ocorre desde o início do Sistema no Servidor até um pedido de execução por um cliente. O passo é dado pela imagem 4.2.

Figura 4.2 – Passo a passo do registro de um cliente até a execução de uma requisição



Fonte: O próprio autor

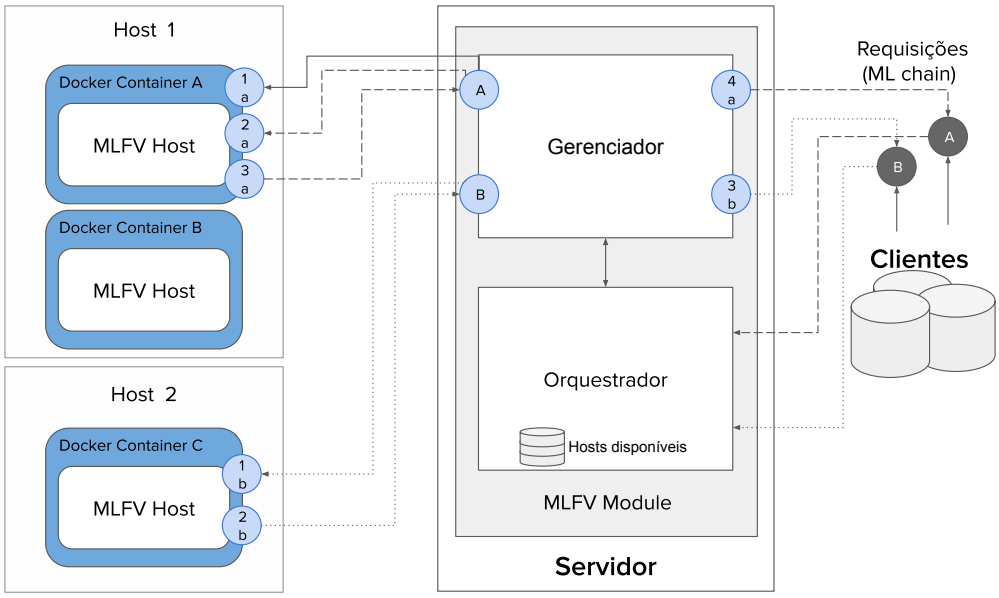
Considerando então o Módulo principal do servidor inicializado, o orquestrador possui seu banco de dados vazio e um processo rodando de fundo na porta 8886, que escuta solicitações de inscrição no sistema. Assim o primeiro passo no host pode ser inciado. **(1)** O programa inicia com um broadcast na rede solicitando a inscrição no servidor de hosts. **(2)** O servidor retorna pela porta 8887, uma confirmação de inicialização, junto com o seu endereço IP e um dockerfile para ser executado. **(3)** Assim a máquina host gera uma imagem docker a partir dos arquivos recebidos do servidor e então **(4)** inicializa a imagem docker. **(5)** A imagem por sua vez inicializa o programa que irá se inscrever no servidor com base nos dados recebidos no passo **(2)**. Finalizada a inicialização, o MLFV Hosts automaticamente irá requisitar uma **(6)** inscrição no banco de dados do orquestrador, e se não ocorrer nenhum erro, **(7)** o Orquestrador retorna uma confirmação. Assim a instanciação e inscrição da maquina virtual é dada pelos 7 passos anteriores

Uma vez que uma requisição de um cliente chega ao servidor **(8)**, ela é automaticamente redirecionada ao orquestrador, que verifica em sua base o host que teve menor carga de trabalho até o momento e encaminha para o gerenciador esses dados, que por sua vez **(9)** envia para o container os dados necessários para execução e a função que será executada. Após o término da execução, **(10)** o container host retorna o valor da exe-

cução para o gerenciador, caso seja apenas uma execução a ser feita, o processamento **(11)** retorna ao cliente que efetuou a requisição, caso contrário, repete-se os passos **(9 e 10)**, até que não hajam mais funções a serem executadas.

Agora supondo outro caso, dessa vez possuímos duas requisições e dois hosts que estão provendo serviço ao servidor, o host 1 possui duas instâncias do container docker em sua máquina, já o host 2 tem apenas uma. A requisição **A** possui um conjunto de instruções a serem executadas que dependem de algumas bibliotecas específicas, nesse caso, nenhum nó possuía essas bibliotecas. Já a **B** possui um grupo de instruções que apenas o host 2 possuía a capacidade de executar. Como pode ser visto na 4.3, ambas requisições entram no orquestrador, e são repassadas para o gerenciador. Uma vez que o orquestrador verificou as bibliotecas e viu que a instrução **B** só poderia ser executada no host 2, a requisição foi encaminhada para o "Container C" do host 2. Contudo, a requisição **A** não pode ser executada em nenhum host até o momento já que as bibliotecas não são compatíveis, então o gerenciador envia ao container escolhido pelo orquestrador um pedido de instalação de bibliotecas, representado na 4.3 como 1a, após a instalação dos devidos pacotes, o "Container A" recebe a cadeia para execução no passo 2a, e retorna o resultado no passo 3a. Para a cadeia **B**, as funções a serem executadas chegam no passo 1b e seu resultado é retornado para o servidor no passo 2b. Por fim, os passos 4a e 3b retornam os resultados das execuções aos devidos clientes.

Figura 4.3 – Exemplo de execução de duas requisições com necessidade de instalação de bibliotecas



Fonte: O próprio autor

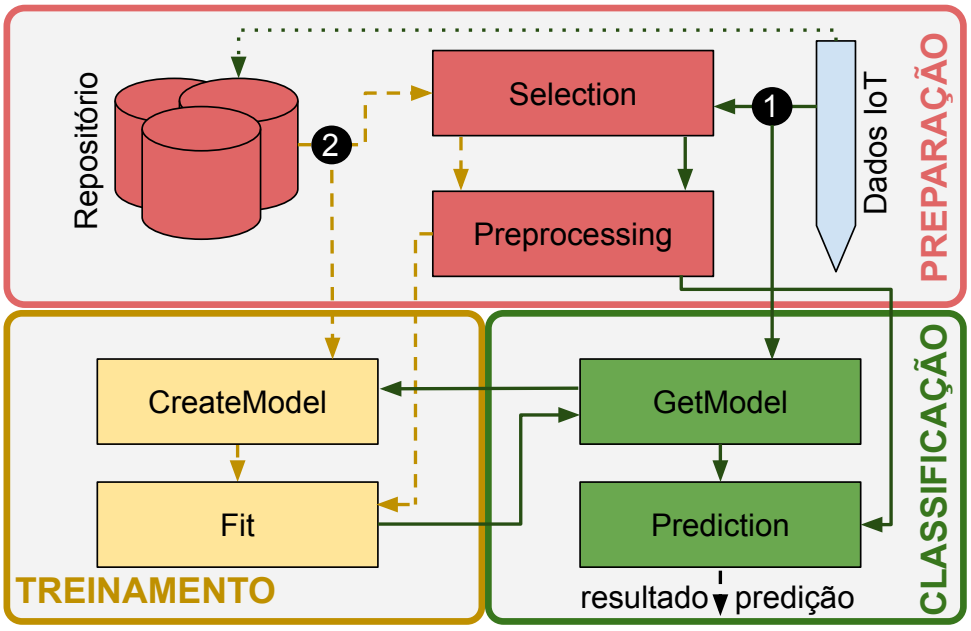
5 TESTES E RESULTADOS

Com base nos testes realizados por Souza (SOUZA, 2019), a abordagem será testada com o mesmo método baseado na extração de conhecimento de base dados (KDD), uma vez que esse garante a padronização e um fluxo de execução de atividades bem definido, flexível e otimizado. Contudo, não é obrigatório a utilização desse procedimento, pois serve mais como um guia de testes e comparativo para avaliação do sistema.

5.1 METODOLOGIA

O sistema de testes é dividido em seis funções e três etapas. As etapas organizam as funções de acordo com suas características e representam o objetivo a ser alcançado com a execução destas. A função é uma rotina de execução, com um conjunto de algoritmos e atividades a serem executadas. As etapas são divididas em preparação, treinamento e classificação, onde possuímos duas cadeias de paralelismo: a de **(1) criação e treinamento** e a de **(2) predição dos dados**, onde essas serão distribuídas pelas funções de determinada etapa. Essa divisão pode ser visualizada na figura 5.1

Figura 5.1 – Visão global das Etapas e Funções de ML



Fonte: (SOUZA, 2019)

5.1.1 Etapa de preparação

A etapa de preparação possui duas funções responsáveis por automatizar o processo de preparação e manipulação de dados brutos. Seu principal objetivo é transformar esses dados brutos em um conjunto de dados adequado para ser utilizado por algoritmos de ML. Para isso, esta etapa é composta de duas funções.

Selection. A primeira função tem como objetivo agrupar esquemas de bases de dados em um mesmo formato. Essa organização permite que o acesso aos dados seja rápido, otimizando todos os processos seguintes. Uma vez executada, a função retorna como resultado de um novo conjunto de dados.

Preprocessing. Aplica diversos algoritmos, filtros e técnicas de normalização para realizar a limpeza ou remoção de inconsistências básicas nos dados. O objetivo é limpar os registros que podem atrapalhar no treinamento da inteligência artificial. Retorna, como resultado, o conjunto de dados da entrada devidamente pré-processado.

5.1.2 Etapa de treinamento

Essa etapa é responsável por criar, testar e validar o modelos de ML. Esse modelo é criado a partir da etapa anterior de extração de dados, e será utilizado na etapa de classificação para validar o resultado.

CreateModel. Função responsável por instanciar um modelo de ML. Para isso, essa função recebe como parâmetros de entrada o nome do classificador e as configurações desejadas. Uma vez terminada, a função retorna como resultado, uma instância de um modelo de ML escolhido.

Fit. Função responsável pelo treinamento do modelo de ML criado utilizando um conjunto de dados de treinamento. Recebe como parâmetros de entrada duas variáveis: (i) o modelo de ML criado anteriormente e o (ii) conjunto de dados para treinamento, provido pela função de preprocessamento. Ao final, a função retorna a instância de um modelo de ML treinado e ajustado com base nos dados de treinamento fornecidos.

5.1.3 Etapa de classificação

A última etapa é responsável por realizar a predição do conjunto de dados que utiliza como parâmetro o resultado das duas etapas anteriores e consiste em duas funções.

GetModel. Tem o objetivo de carregar o modelo de ML criado. Em geral, a função irá recriar esse modelo, o qual já transitou pelo processo de treinamento, e repassá-lo para a função ‘Prediction’ executar a tarefa de predição. Retorna, quando terminado esse

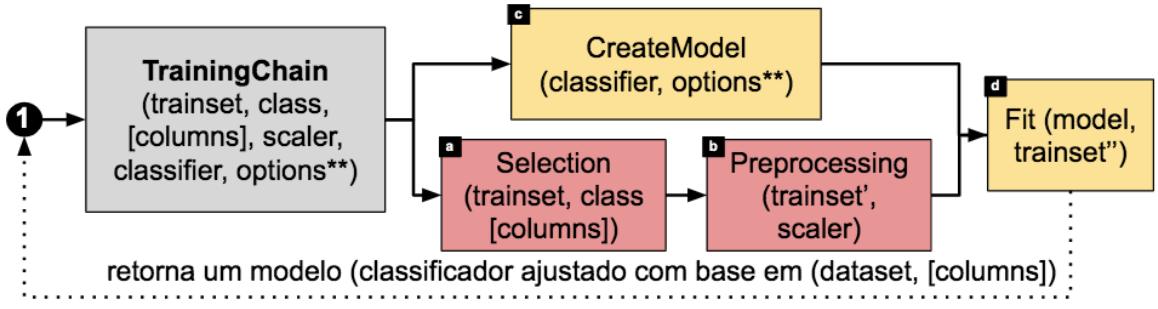
processo, como resultado o modelo para iniciar a tarefa de classificação na função 'Prediction'.

Prediction. Tem como tarefa realizar a classificação e predição de dados fornecidos. A função 'Prediction' recebe, como entrada, um modelo, carregado pela função 'getModel', e o conjunto de dados a ser classificado, resultado do 'preprocessing'. Uma vez finalizada a tarefa de predição, retorna o conjunto de dados devidamente classificado ao **Módulo MLFV**, finalizando a execução da cadeia de funções.

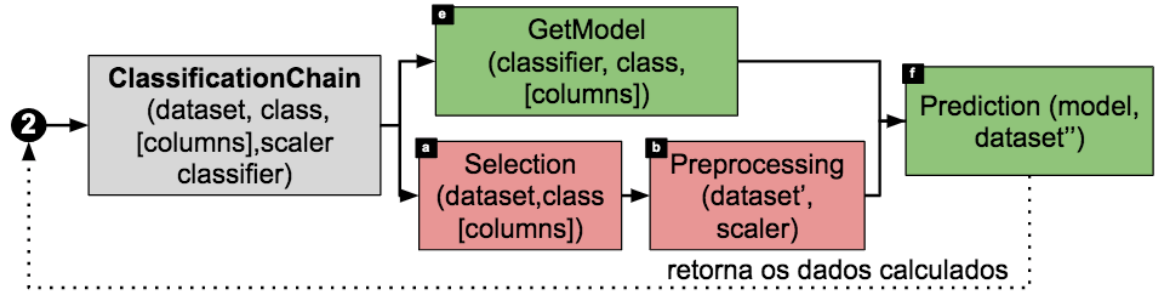
5.2 CADEIAS DE FUNÇÕES

As Machine Learning Function Chain (MLFC) estão representadas como círculos pretos com os números 1 e 2 da imagem 5.1. Essas baseiam-se no conceito de Encadeamento de funções de rede, porém propõem o encadeamento de funções de ML para criar serviços de extração e descoberta de conhecimento em dados. Ao contrário de uma sequência de funções de redes tradicional, uma MLFC é composta pela sequência de funções e atividades necessárias para criação e treinamento de modelos de ML ou para realizar a classificação e predição de dados usando esses modelos, como pode ser visualizado na figura 5.2

Figura 5.2 – Cadeias MLFV



(a) **Cadeia de Treinamento**.



(b) **Cadeia de Classificação**.

Fonte: (SOUZA, 2019)

De acordo com Souza (SOUZA, 2019), a **Cadeia de Treinamento** (Figura 5.2a) agrupa todo o processo hierárquico de execução das atividades para criação, preparação e treinamento, através de aprendizado supervisionado de modelos de ML. A **Cadeia de Classificação** (Figura 5.2b), por sua vez, apresenta a sequência de atividades necessárias para realização do processo de predição, em dados não classificados, utilizando um modelo de ML. Esse modelo é resultante da cadeia de Treinamento. É possível notar ainda que ambas MLFC permitem que funções sejam organizadas em serial e paralelo, dependendo das suas dependências.

5.3 IMPLEMENTAÇÃO

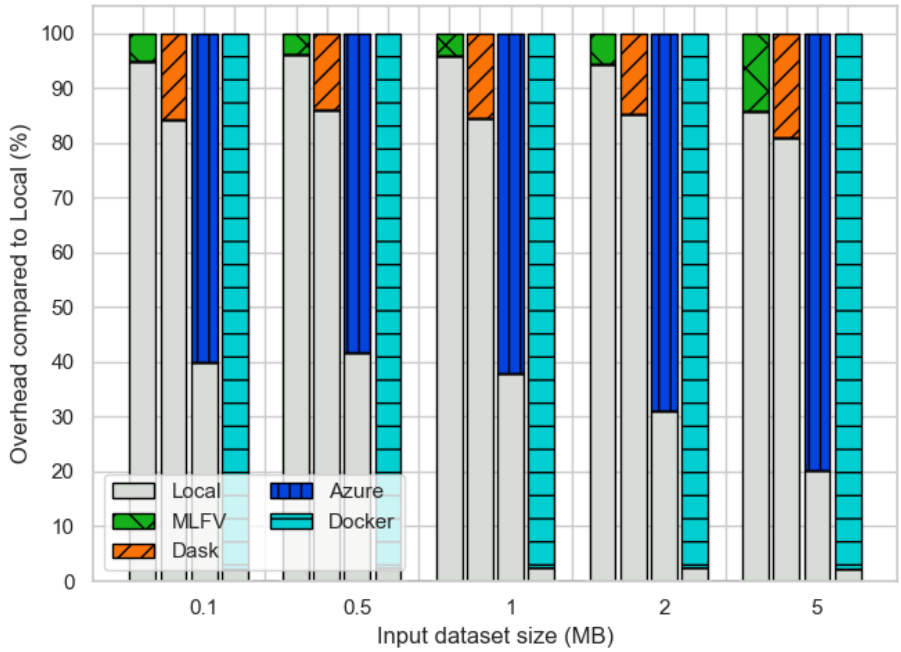
A implementação foi realizada em cima da primeira versão do MLFV (SOUZA, 2019), com mudanças na arquitetura do sistema a fim de realizar a virtualização em Docker. Foi criado também o sistema de cadastro de usuários e descoberta na rede. A implementação foi feita em Python, com a utilização de bibliotecas que contém métodos disponíveis para implementação de algoritmos (Scikit-Learn (v0.20.0)), cálculos matemáticos (numpy (v1.15.2)), manipulação de dados e análises estatísticas (Pandas (v0.23.4)) [1].

5.4 TESTES

Foram realizados dois experimentos em dois ambientes de testes, comparando a proposta Docker com outras quatro diferentes abordagens semelhantes. A primeira abordagem é o MLFV, que serve como base para comparar a sobrecarga do sistema antigo em relação ao proposto. A segunda dessas abordagens é o Microsoft Azure MLaaS Studio [2], o qual utiliza a nuvem do Azure para executar a criação, treinamento e implantação de modelos de ML. A terceira é a biblioteca Python Dask.distributed [3], que apresenta uma abordagem semelhante à atual onde um servidor central, chamado de **dask-scheduler**, coordena as ações de diversos trabalhadores (escravos), chamados de **dask-workers**, que, espalhados por vários computadores, realizam as tarefas que compõem o processo de ML. Por fim, a última abordagem se refere a funções de ML criadas localmente, a serem executadas em apenas um cliente (PC), utilizando bibliotecas padrões de Python, como Scikit-Learn, Pandas e etc. Todos os experimentos relatados nesta seção foram executados 30 vezes, onde os gráficos, para cada experimento, apresentam a média e o desvio padrão desses valores.

[1] <https://scikit-learn.org> | <https://pandas.pydata.org> | <http://www.numpy.org>
[2] <https://azure.microsoft.com/en-us/services/machine-learning-studio>
[3] <http://distributed.dask.org>

Figura 5.3 – Sobrecarga imposta pelas abordagens em comparação com um ambiente Local com docker inicializando do zero



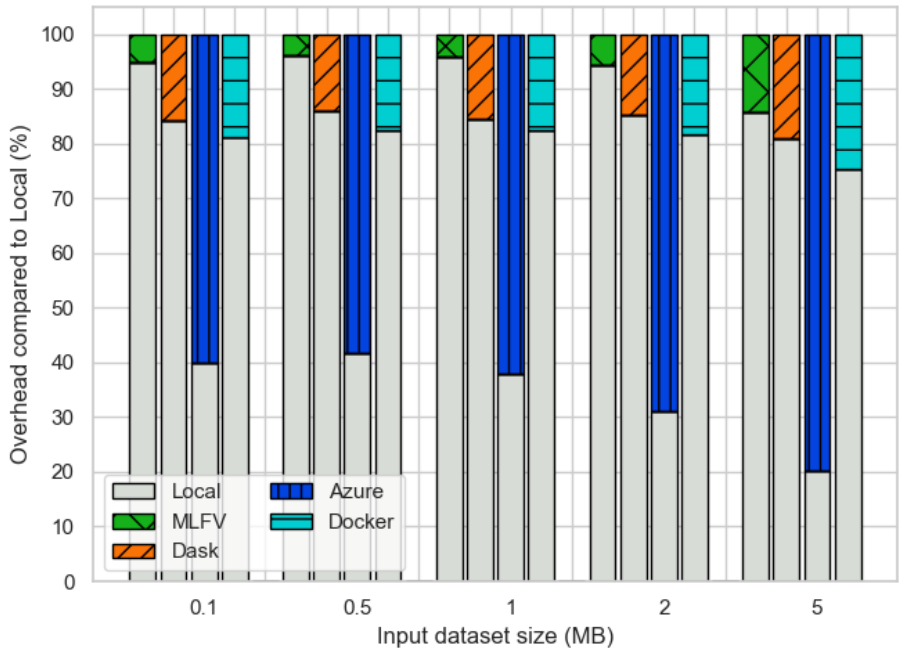
Fonte: O próprio autor

O primeiro experimento foi dividido em três partes. Na primeira o cliente deveria se inscrever e inicializar o docker do zero para vermos, assim, a sobrecarga causada pela primeira execução do serviço. A segunda parte foi realizada apenas com a inicialização do docker no cliente, assim, a única sobrecarga do sistema seria na hora de levantar a instância do docker. E a última representa a mesma execução porém com o docker já instanciado nas máquinas hosts. Os gráficos com os resultados podem ser conferidos nas figuras 5.3, 5.4 e 5.5, respectivamente .

Os valores de tempo foram calculados com base em um cronômetro disparado pelo servidor. Na primeira parte do primeiro experimento foi considerado no tempo final, o tempo de comunicação entre as máquinas locais, a instanciação e download dos pacotes e execução de uma requisição de tamanho variável de acordo com a figura 5.3. Foram utilizadas 4 máquinas para a realização dos testes, uma delas é o servidor, a outra o cliente que envia as requisições, e as outras duas, as máquinas hosts rodando uma instância de docker.

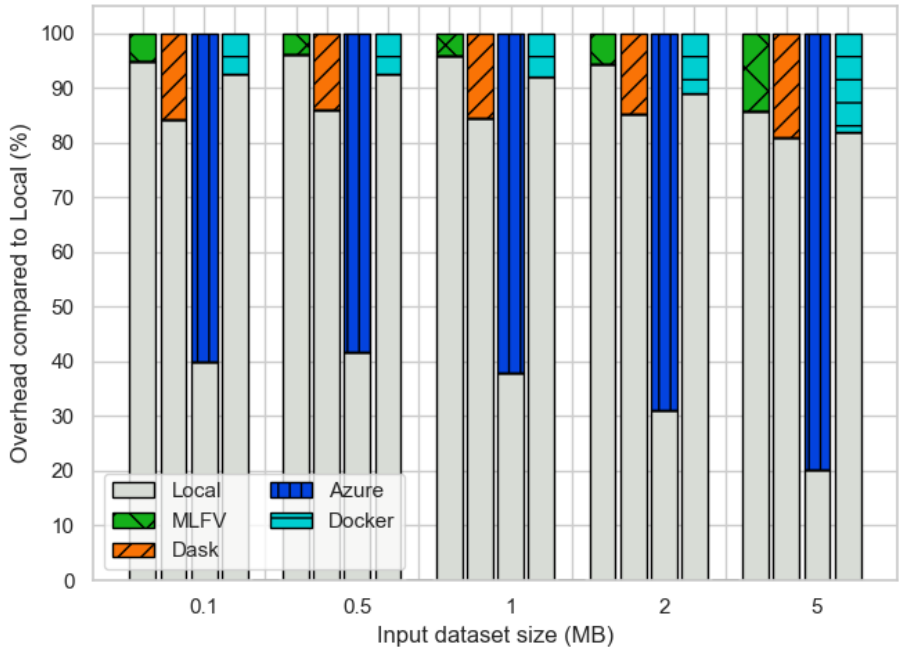
No primeiro experimento pode-se avaliar que é muito caro para o host recarregar o sistema toda vez que este o for utilizar. Já que o host irá realizar um download dos dados requisitados pela imagem dokcer, a sobrecarga gerada pela construção e download da imagem junto com a comunicação e instanciação da máquina acaba sendo muito custosas para o sistema. A reinicialização completa do serviço teve um tempo médio de 175 segundos com conexão cabeada e limite de download de 100MBPS.

Figura 5.4 – Sobrecarga imposta pelas abordagens em comparação com um ambiente Local com docker não instanciado na máquina do host



Fonte: O próprio autor

Figura 5.5 – Sobrecarga imposta pelas abordagens em comparação com um ambiente Local com docker já instanciado na máquina do host



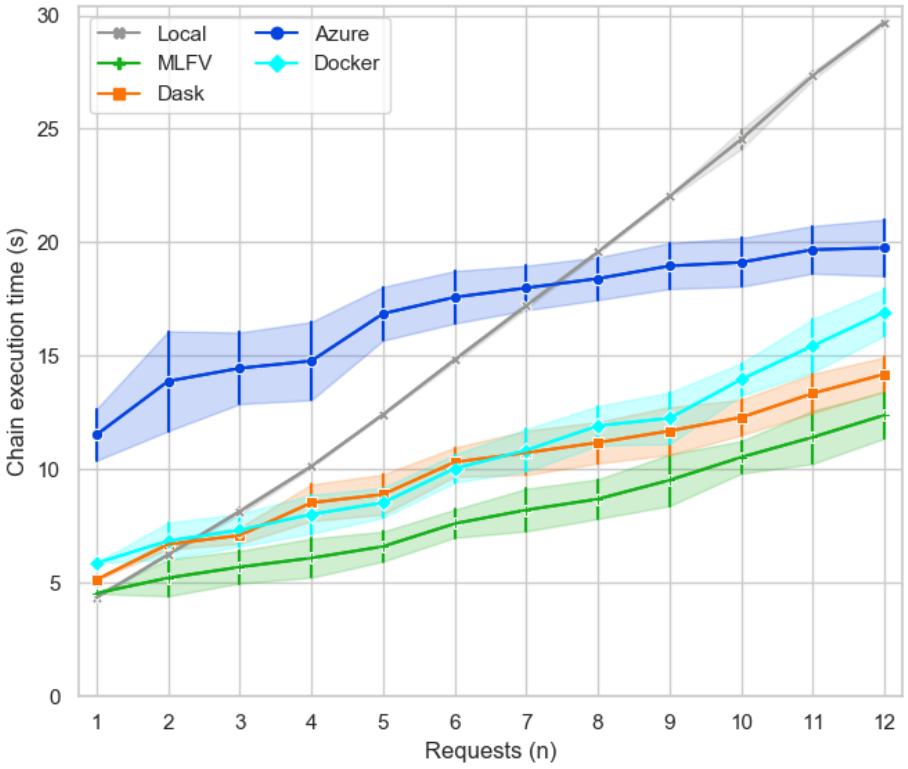
Fonte: O próprio autor

Na segunda parte do primeiro experimento foi considerado que o docker já havia sido criado no host, e deveria ser levantado para a execução, assim o tempo final a ser considerado soma o tempo de comunicação entre as máquinas locais, o tempo de instanciação do docker e a execução de uma requisição de tamanho variável. Com um tempo médio de 73,5 milissegundos para instanciação de um cliente, o resultado final gera uma sobrecarga de 20% em média em comparação com a abordagem local, como pode ser visualizado na 5.4.

Na última parte do primeiro teste, o docker já havia sido inicializado, e o único tempo que soma para o cálculo é o de comunicação entre as máquinas locais e a execução da chain de tamanho variável. Sendo assim os resultados foram melhorados chegando a apenas 10% de média em sobrecarga, como pode ser visto na figura 5.5

O segundo experimento foi para testar como o sistema e as outras abordagens se comportavam em relação a múltiplas chamadas simultâneas. Para esse teste, o tamanho do conjunto de dados foi fixado em 1MB e o número das requisições simultâneas alternando entre 1 até 12 chamadas. Por fim, foi medido o tempo de execução de cada abordagem para realizar a cadeia de classificação completa, tendo os resultados apresentados pela Figura 5.6.

Figura 5.6 – Múltiplas requisições simultâneas



Fonte: O próprio autor

Na figura 5.6 as linhas representam o tempo médio da execução das múltiplas chamadas em segundos, as barras que saem das linhas representam o desvio padrão em relação as múltiplas execuções que foram realizadas.

Com a análise dos resultados apresentados, percebemos que as abordagens de processamento distribuído não apresentam um abrupto aumento no tempo de execução, uma vez que conseguem manter de maneira eficiente a divisão de tarefas pelo sistema. Além disso, pode-se analisar também, que a abordagem proposta é computacionalmente mais cara, e acaba consumindo mais tempo das máquinas. Isso se deve ao fato da utilização de virtualização, que acaba consumindo recursos do hospedeiro e acaba influenciando parcialmente nas execuções. Contudo, os problemas de inconsistência e incompatibilidade nos nós do sistema são solucionados.

6 CONCLUSÃO

Dentro dos algoritmos disponíveis para processamento distribuídos, possuímos alguns focados em aprendizado de máquina. A proposta desse trabalho foi desenvolver uma abordagem que seja capaz de compartilhar dispositivos para processamento de ML e ao mesmo tempo pudesse prever erros de incompatibilidade, através da técnica de virtualização em nível de sistema operacional.

Uma vez que o sistema foi implementado e estava funcional, foram realizados os testes. Como esperado, é perceptível que a técnica de virtualização em sua primeira execução cria uma sobrecarga ao sistema, uma vez que é necessário a instalação de todos os pacotes e bibliotecas. Porém, após isso, a virtualização possui um ótimo desempenho em relação as outras abordagens se mantendo consistente sem grandes perdas. Tendo em vista que o problema de inconsistência de pacotes foi resolvido e a distribuição de processamento ocorreu de maneira correta, pode-se afirmar que esse trabalho atingiu os objetivos esperados.

6.1 TRABALHOS FUTUROS

Como trabalho futuro esperamos adicionar suporte a novos dispositivos e funcionalidades, tais como suporte a execução em Unidade gráfica de processamento (GPU) e execução do sistema distribuído em dispositivos de IOT, como micro-computadores.

REFERÊNCIAS BIBLIOGRÁFICAS

BERRY, M. J.; LINOFF, G. S. **Data mining techniques: for marketing, sales, and customer relationship management**. [S.l.]: John Wiley & Sons, 2004.

CHAN, S. et al. Predictionio: a distributed machine learning server for practical software development. In: **Proceedings of the 22nd ACM CIKM**. [S.l.: s.n.], 2013. p. 2493–2496.

CONTAINERS, L. Linux containers. **Linux Containers:< http://lxc. sourceforge. net**, 2009.

CREASY, R. J. The origin of the vm/370 time-sharing system. In: . [S.l.]: IBM, 1981. v. 25, n. 5, p. 483–490.

DATAREPORTAL. **DIGITAL 2018: GLOBAL DIGITAL OVERVIEW**. 2018. Acessado em 09 out 2019. Disponível em: <https://datareportal.com/reports/digital-2018-global-digital-overview>.

DEUTSCH, P.; GAILLY, J. Rfc 1050: Zlib 3.3 specification. In: . [S.l.: s.n.], 1996.

DOCKER Inc. **Why Docker**. 2013. Acessado em 09 out 2019. Disponível em: <https://www.docker.com/why-docker>.

Fundação Getúlio Vargas. **Pesquisa Anual do Uso de TI**. 2019. Acessado em 09 out 2019. Disponível em: <https://eaesp.fgv.br/ensinoeconhecimento/centros/cia/pesquisa>.

GOLDBERG, R. P. Survey of virtual machine research. **Computer**, IEEE, v. 7, n. 6, p. 34–45, 1974.

HEBB, D. O. **The organization of behavior: a neuropsychological theory**. Wiley, NJ: [s.n.], 1949.

IBM. **RPC Model**. 2019. Acessado em 09 out 2019. Disponível em: <https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/commprogramming/rpc_mod.html>.

Inovex Blog. **The Edge is Near: An Introduction to Edge Computing!** 2019. Acessado em 09 out 2019. Disponível em: <https://www.inovex.de/blog/edge-computing-introduction/>.

Internet Live Stats. **Internet Live Stats**. 2019. Acessado em 09 out 2019. Disponível em: <https://www.internetlivestats.com/one-second/#traffic-band>.

JOSEP, A. D. et al. A view of cloud computing. In: . [S.l.: s.n.], 2010. v. 53, n. 4.

JOY, A. M. Performance comparison between linux containers and virtual machines. In: IEEE. **2015 International Conference on Advances in Computer Engineering and Applications**. [S.l.], 2015. p. 342–346.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

NILSSON, N. J. **Introduction to machine learning: An early draft of a proposed textbook**. [S.l.]: USA; Stanford University, 1996.

ODLYZKO, A. M. Internet traffic growth: Sources and implications. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. **Optical transmission systems and equipment for WDM networking II**. [S.l.], 2003. v. 5247, p. 1–15.

RIBEIRO, M.; GROLINGER, K.; CAPRETZ, M. A. Mlaas: Machine learning as a service. In: IEEE. **2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)**. [S.l.], 2015. p. 896–902.

ROCKLIN, M. Dask: Parallel computation with blocked algorithms and task scheduling. In: CITESEER. **Proceedings of the 14th python in science conference**. [S.l.], 2015.

SMITH, J. E.; NAIR, R. An overview of virtual machine architectures. In: . [S.l.: s.n.], 2001. v. 26, p. 1–20.

SOUZA, R. **MLFV: Uma Abordagem Consciente do Estado da Rede para Orquestração de Cadeias de Funções de Aprendizado de Máquina**. 2019. 81 p. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Maria, Santa Maria, 2019.

SRINIVASAN, R. **XDR: External Data Representation Standard**. RFC Editor, 1995. RFC 1832. (Request for Comments, 1832). Disponível em: <https://rfc-editor.org/rfc/rfc1832.txt>.

TAURION, C. **Cloud computing-computação em nuvem**. [S.l.]: Brasport, 2009.

THURLOW, R. Rpc: Remote procedure call protocol specification version 2. In: . [S.l.: s.n.], 2009.

TU, J. V. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. **Journal of clinical epidemiology**, Elsevier, v. 49, n. 11, p. 1225–1231, 1996.

WHITE, J. E. High-level framework for network-based resource sharing. In: . [S.l.: s.n.], 1975.

YAO, Y. et al. Complexity vs. performance: Empirical analysis of machine learning as a service. In: **ACM Internet Measurement Conference (IMC)**. [S.l.: s.n.], 2017.

ZHAO, J. et al. Privacy-preserving machine learning based data analytics on edge devices. In: ACM. **Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society**. [S.l.], 2018. p. 341–346.