Universidade Federal de São Carlos – UFSCar
Centro de Ciências Exatas e de Tecnologia – CCET
Departamento de Computação – DC
Programa de Pós-Graduação em Ciência da Computação – PPGCC

Luís Henrique Puhl de Souza

Uma Implementação distribuída em Névoa do algoritmo de Detecção de Novidade em Fluxos de Dados MINAS

São Carlos
2021

Luís Henrique Puhl de Souza

Uma Implementação distribuída em Névoa do algoritmo de Detecção de Novidade em Fluxos de Dados MINAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Sistemas de Computação

Orientador: Doutor Hermes Senger

São Carlos
2021

Puhl, Luis

Uma Implementação distribuída em Névoa do algoritmo de Detecção de Novidade em Fluxos de Dados MINAS / Luis Puhl -- 2021.
65f.

Dissertação (Mestrado) - Universidade Federal de São Carlos, campus São Carlos, São Carlos
Orientador (a): Hermes Senger
Banca Examinadora: Helio Guardia
Bibliografia

1. Detecção de Intrusão. 2. Internet das Coisas. I. Puhl, Luis. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Ronildo Santos Prado - CRB/8 7325

Universidade Federal de São Carlos – UFSCar
Centro de Ciências Exatas e de Tecnologia – CCET
Departamento de Computação – DC
Programa de Pós-Graduação em Ciência da Computação – PPGCC

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Luís Henrique Puhl de Souza, realizada em 5 de Julho de 2021:

Doutor Hermes Senger
Presidente e Orientador

Doutor Kelton Augusto Pontara da Costa
Convidado

Doutor Paulo Sérgio Lopes de Souza
Convidado

São Carlos
2021

Agradecimentos

Agradeço à constante compania, incentivo e ensinamentos dos colegas e agradeço especialmente à CNPq pelo suporte financiero (contrato 167345/2018-4).

Resumo

Em um cenário de crescente número de dispositivos na Internet das Coisas (IoT), gerando proporcional crescimento no volume dos fluxos de dados gerados, são necessários métodos robustos para a mineração de fluxos contínuos de dados. Uma das áreas afetadas pelo crescimento vertiginoso do número de dispositivos e os fluxos associados a eles é a área de segurança da informação, onde são necessárias ferramentas de detecção de intrusão em redes que operem em ambientes de computação em névoa, devido aos custos de comunicação associados a operar estas ferramentas somente em ambiente de nuvem. As ferramentas de detecção de intrusão utilizam extensivamente algoritmos de detecção de novidade em fluxos de dados para identificar padrões no tráfego da rede. Porém, os algoritmos que tratam adequadamente dos desafios de detecção de novidade em fluxos de dados, como mudança e evolução de conceito e atualização contínua do modelo de classificação sem interferência de especialistas, ainda são pouco utilizados. O algoritmo de detecção de novidade em fluxo de dados MINAS tem recebido atenção de pesquisas recentes por tratar desses desafios de detecção de novidade em fluxos de dados. No entanto, apesar de sua divisão em três partes semi-independentes, este algoritmo ainda não foi adaptado para processar grandes volumes de fluxos reais em ambiente de computação em névoa. O presente trabalho aborda essa lacuna, propondo um sistema que implementa o algoritmo MINAS de maneiraiot distribuída num contexto de detecção de intrusão e computação em névoa. Experimentos mostram que o algoritmo MINAS pode ser paralelizado e distribuído utilizando plataformas de processamento de fluxos como *Apache Flink*.

Palavras-chave: Detecção de Novidades. Detecção de Intrusão. Fluxos de Dados. Computação Distribuída. Computação em Névoa. Internet das Coisas.

Abstract

The ongoing implementation of the Internet of Things (IoT) is sharply increasing the number and variety of small devices on edge networks. Likewise, the attack opportunities for hostile agents also increases, requiring more effort from network administrators and strategies to detect and react to those threats. For a network security system to operate in the context of edge and IoT, it has to comply with processing, storage, and energy requirements alongside traditional requirements for stream and network analysis like accuracy and scalability. Using a previously defined architecture (IDSA-IoT), we address the construction and evaluation of a support mechanism for distributed Network Intrusion Detection Systems (NIDS) based on the MINAS Data Stream Novelty Detection algorithm. We discuss the algorithm steps, how it can be deployed in a distributed environment, the impacts on the accuracy and evaluate performance and scalability using a cluster of constrained devices commonly found in IoT scenarios. The obtained results show a negligible accuracy loss in the distributed version but also a small reduction in the execution time using low profile devices. Although not efficient, the parallel version showed to be viable as the proposed granularity provides equivalent accuracy and viable response times.

Keywords: Novelty Detection. Intrusion Detection. Data Streams. Distributed Computing. Fog Computing. IoT devices.

Lista de ilustrações

Lista de tabelas

Lista de siglas

ND-DS Detecção de Novidades em Fluxos de Dados (*Novelty Detection in Data Stream*)

IoT Internet das Coisas (*Internet of Things*)

MPI Interface de Troca de Mensagens (*Message Passing Interface*)

ML Aprendizado de Máquina (*Machine Learning*)

NIDS Sistema de Detecção de Intrusão em Redes (*Network Intrusion Detection System*)

Sumário

Capítulo 1

Introdução

A Internet das Coisas (*Internet of Things*) (IoT) conecta globalmente variados dispositivos, incluindo dispositivos móveis, *wearables*, eletrônicos domésticos, automóveis e sensores industriais. Estes dispositivos podem, através da Internet, ser acessados, conectar-se a outros dispositivos, servidores ou aplicações, tudo com mínima intervenção ou supervisão humana (TAHSIEN; KARIMIPOUR; SPACHOS, 2020; ABANE et al., 2019; HADDADPAJOUH et al., 2019; SHANBHAG; SHANKARMANI, 2015). Outra característica de dispositivos IoT são os recursos computacionais dimensionados, para propósitos específicos, que limitam a capacidade de computar outras funções muito além da função original do dispositivo.

Segurança e privacidade são uma grande preocupação em IoT, especialmente em relação aos dados pessoais como localização e saúde, aos quais dispositivos podem ter acesso (SENGUPTA; RUJ; BIT, 2020). Contudo, além dos dados de sensores e atuadores que gerenciam, esses dispositivos, se subvertidos, podem gerar tráfego maligno, como o gerado pela *mirai botnet* em 2016 (KAMBOURAKIS; KOLIAS; STAVROU, 2017; KOLIAS et al., 2017). Nesse cenário, fatores que podem favorecer a subversão dos dispositivos incluem a falta de controle sobre a origem do hardware e software embarcado nos dispositivos bem como a menor frequência de atualizações deste software. Além disso, estes dispositivos têm longa vida e, após implantação, convivem com ampla diversidade de outros dispositivos, complicando a manutenção da rede que os hospeda, aumentando sua superfície de ataque.

No contexto de segurança de redes IoT, ferramentas que facilitem a detecção e resposta a ataques são necessárias. Como a maioria dos dispositivos IoT tem recursos limitados (como bateria, processamento, memória e comunicação), técnicas de segurança tradicionais baseadas em algoritmos configuráveis não são usuais, com isso restam as técnicas de observação de rede (ZHOU et al., 2017). Ferramentas como Sistema de Detecção de Intru-

são em Redes (*Network Intrusion Detection System*) (NIDS) observam o comportamento da rede e de seus dispositivos e detectam possíveis ataques.

Para implementação de NIDS, técnicas de Aprendizado de Máquina (*Machine Learning*) (ML) tem sido empregadas na detecção de ataques a partir de ataques conhecidos ou descobrir novos ataques o mais cedo possível (BUCZAK; GUVEN, 2016; MITCHELL; CHEN, 2014). Apesar do uso promissor de ML para segurança para sistemas IoT, muitos estudos na literatura (BUCZAK; GUVEN, 2016; MITCHELL; CHEN, 2014; TAHSIEN; KARIMIPOUR; SPACHOS, 2020) são limitados à métodos tradicionais de ML. Estes métodos tradicionais utilizam modelos estáticos para descrever e prever o comportamento da rede que, por serem estáticos, não mantém confiabilidade quando confrontados com a evolução de ataques (VIEGAS et al., 2019; LOPEZ; DUARTE; PUJOLLE, 2019).

Além das complicações de confiabilidade, a grande quantidade de dispositivos em redes distantes gerando dados em volumes e velocidades consideráveis, diretamente ligados às suas funções originais ou metadados produzidos como subproduto, técnicas tradicionais que tratam grandes lotes não são aplicáveis. Considerando fluxos contínuos de dados (*Data Stream*), técnicas de mineração de fluxos de dados (*Data Stream Mining*) são amplamente necessárias. Nesses cenários, essas técnicas são aplicadas, por exemplo, em problemas de monitoramento e classificação de valores originários de sensores para tomada de decisão tanto em nível micro, como na modificação de atuadores remotos, ou macro, na otimização de processos industriais.

Dentre as técnicas de mineração de fluxo de dados, classificadores podem ser utilizados para detectar padrões conhecidos e, em conjunto com algoritmos de Detecção de Novidades em Fluxos de Dados (*Novelty Detection in Data Stream*) (ND-DS), detectar novos padrões. Essa capacidade de detectar novos padrões é relevante para NIDS, onde novidades na rede podem representar novas funcionalidades ou ataques por agentes maliciosos, sem assinaturas existentes em bancos de dados de ataques conhecidos. Outras características que fazem ND-DS atraente para NIDS são a produção de respostas imediatas e a detecção de novidades e mudança de conceitos já conhecidos.

Análises como *Data Stream Mining* e ND-DS são geralmente implementadas sobre o paradigma de computação na nuvem (*Cloud Computing*) e, recentemente, sobre paradigmas como computação em névoa (*Fog Computing*). Para *fog*, além dos recursos em *cloud*, são explorados os recursos distribuídos pela rede desde o nó remoto até a *cloud*. Processos que dependem desses recursos são distribuídos de acordo com características como sensibilidade à latência, privacidade, consumo computacional ou energético.

Como elaborado, a aplicação de ND-DS para detecção de ameaças em fluxos de dados originários de redes IoT dentro de NIDS tem sido um ponto de interesse (VIEGAS et al., 2019; LOPEZ; DUARTE; PUJOLLE, 2019; COSTA et al., 2019). Este trabalho foca explora as características de implementação destas técnicas em conjunto, focando serviços localizados na borda da rede, de maneira distribuída, para uso em ambientes IoT.

1.1 Motivação

Um problema recente que une, em um único contexto, os métodos de computação em névoa, processamento de fluxo de dados e detecção de novidades nesses fluxos é a detecção de intrusão em redes de dispositivos IoT. Para tratar esse problema, a arquitetura IDSA-IoT, recentemente proposta por Cassales et al. (2019), aplica ao problema algoritmos atuais de detecção de novidades em fluxos, executando esses algoritmos em ambiente próximo aos dispositivos e avaliando-os quanto à detecção de intrusão.

Na arquitetura proposta, Cassales et al. (2019) avaliou os algoritmos ECSMiner (MASUD et al., 2011), AnyNovel (ABDALLAH et al., 2016) e MINAS (FARIA; CARVALHO; GAMA, 2016), sendo que o último mostrou resultados promissores.

Contudo, o algoritmo MINAS ainda não foi implementado e avaliado com paralelismo, multi-processamento ou distribuição computacional, que são necessários para tratar fluxos de dados com grandes volumes e velocidades. O tratamento de distribuição em ambiente *fog computing* é essencial para aplicação deste algoritmo ao problema de detecção de intrusão em redes IoT, pois esta aplicação requer tempo de resposta mínimo e mínima comunicação entre nós distantes, como aqueles na borda e na nuvem. Ainda observando o algoritmo MINAS, destaca-se a possível divisão em três partes semi-independentes, sendo elas treinamento, classificação e detecção de novidade; a classificação é o elemento central cujos resultados são utilizados para a identificação de intrusões.

Ainda no contexto de ND-DS como método de detecção de intrusão, outras propostas tratam do caso de fluxos com grandes volumes e velocidades, como é o caso de Viegas et al. (2019), que apresenta o *BigFlow* no intuito de detectar intrusão em redes do tipo *10 Gigabit Ethernet*, que podem produzir um volume considerável. Essa implementação foi feita sobre uma plataforma distribuída processadora de fluxos (*Apache Flink*) executada em um cluster com até 10 nós de trabalho, cada um com 4 núcleos de processamento, totalizando 40 núcleos, para atingir taxas de até $10,72$ *Gbps*.

Os trabalhos de Cassales et al. (2019) e Viegas et al. (2019) abordam detecção de intrusão em redes utilizando algoritmos de ND em DS, porém com perspectivas diferentes. O primeiro investiga *IoT* e processamento em *fog* e baseia-se em um algoritmo genérico de detecção de novidade. O segundo trabalho trata de *backbones* e processamento em *cloud* e implementa o próprio algoritmo de detecção de novidade. Essas diferenças deixam uma lacuna onde, de um lado, tem-se uma arquitetura mais adequada para o ambiente *fog* com um algoritmo estado da arte de detecção de novidades, porém sem paralelismo e. Do outro lado da lacuna, tem-se um sistema escalável de alto desempenho porém almejando outro ambiente (*cloud*) e com um algoritmo menos preparado para os desafios de detecção de novidades.

A proposta, aqui chamada sistema M-FOG, adapta a arquitetura IDSA-IoT (CASSALES et al., 2019) empregando o algoritmo de ND-DS algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016), tornando-o capaz de ser executado em um sistema distribuído

composto de pequenos computadores com recursos limitados, alocados na borda da rede próximos dos dispositivos IoT. Utilizando a nova implementação do algoritmo algoritmo MINAS, avalia-se experimentalmente como a distribuição afeta a capacidade do sistema de detectar mudanças (novidades) nos padrões de tráfego e o impacto na eficiência computacional. Por fim, algumas estratégias e políticas para configuração do sistema de detecção de novidades em fluxo de dados são discutidas.

1.2 Objetivos

Como estabelecido na Seção 1.1, a lacuna no estado da arte observada é a ausência de uma implementação de algoritmo de detecção de novidades no contexto de computação distribuída na borda. Neste sentido, um algoritmo que trate adequadamente os desafios de fluxo de dados contínuos (como volume e velocidade do fluxo, evolução e mudança de conceito) e considere o ambiente de computação em névoa aplicada à detecção de intrusão.

Portanto, seguindo os trabalhos do Grupo de Sistemas Distribuídos e Redes (GSDR) da Universidade Federal de São Carlos (UFSCar), propõem-se a construção de uma aplicação que implemente o algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016) de maneira escalável e distribuível para ambientes de computação em névoa, seguindo a arquitetura IDSA-IoT (CASSALES et al., 2019).

Além disso, propõem-se também a avaliação dessa implementação com experimentos baseados na literatura usando conjunto de dados públicos relevantes. O resultado esperado é uma implementação compatível em qualidade de classificação ao algoritmo MINAS e passível de ser distribuída em um ambiente de computação em névoa aplicado à detecção de intrusão.

Com foco no objetivo geral, alguns objetivos específicos são propostos: Implementar o algoritmo MINAS de maneira distribuída sobre uma plataforma de processamento distribuída de fluxos de dados; Arquitetar e implementar um mecanismo de execução e avaliação de qualidade e desempenho para os ambientes escolhidos; Avaliar e comparar a qualidade de detecção de intrusão da nova implementação; Avaliar a qualidade de detecção de intrusão em ambiente distribuído conforme a arquitetura IDSA-IoT em ambiente de computação em névoa.

1.3 Proposta Metodológica

Para cumprir os objetivos citados na Seção 1.2, foi identificada a necessidade de um processo exploratório seguido de experimentação. Tal processo inclui a revisão da literatura, tanto acadêmica quanto técnica, seguida da experimentação através de implementação de aplicação e testes.

O foco da revisão da literatura acadêmica é em trabalhos que abordem processamento de fluxos de dados, classificação de fluxo de dados, detecção de novidades em fluxo de dados e processamento distribuído de fluxo de dados. O objetivo da revisão é o estabelecimento do estado da arte desses assuntos, de forma que alguns desses trabalhos sirvam para comparações e relacionamentos. Além disso, desses trabalhos buscam-se métricas de qualidade de classificação (por exemplo, taxa de falso positivo e matriz de confusão) e métricas de escalabilidade (como taxa de mensagens por segundo e escalabilidade vertical ou horizontal).

A revisão da literatura técnica será focada em plataformas, ferramentas e técnicas para realizar a implementação proposta. Portanto, são selecionadas plataformas de processamento distribuído de DS e técnicas de aprendizado de máquina associadas a elas. Dessa revisão também serão obtidas técnicas ou ferramentas necessárias para extração das métricas de avaliação, bem como *data sets* públicos relevantes para detecção de novidades em DS.

Uma vez definidos o estado da arte, as ferramentas técnicas e os *data sets*, o passo seguinte é a experimentação. Nesse passo, será desenvolvida uma aplicação na plataforma escolhida que, com base no algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016), irá classificar e detectar novidades em DS. Também nesse passo, a implementação será validada comparando os resultados de classificação obtidos com os resultados de classificação do algoritmo original MINAS. Posteriormente, serão realizados experimentos com a implementação e variações em *data sets* e cenários de distribuição em *fog*, coletando as métricas de classificação e escalabilidade.

Ao final, a aplicação, resultados, comparações e discussões serão publicados nos meios e formatos adequados, como repositórios técnicos, eventos ou revistas acadêmicas.

1.4 Organização do trabalho

O restante desse trabalho segue a estrutura: Capítulo 2 aborda conceitos teóricos e técnicos que embasam esse trabalho; Capítulo 3 enumera e discute trabalhos relacionados e estabelece o estado da arte do tema detecção de novidade em fluxos de dados e seu processamento; Capítulo 4 descreve a proposta de implementação, discute as escolhas de plataformas e resultados esperados. Também são discutidos no Capítulo 5 os desafios e resultados preliminares encontrados durante o desenvolvimento do trabalho. Capítulo 6 aborda os ambientes de teste, descreve experimentos realizados e discute os resultados obtidos. Capítulo 7 adiciona considerações gerais sobre o trabalho e seus resultados.

Capítulo 2

Fundamentos Científicos e Tecnológicos

Este Capítulo aborda conceitos que embasam esse trabalho, conceitos teóricos de ambientes e arquiteturas de computação distribuída e detecção de novidade e conceitos técnicos, como plataformas de processamento distribuído de fluxo de dados e o algoritmo MINAS.

2.1 Ambientes de Computação Distribuída

Esta Seção relaciona três ambientes de computação distribuída habitualmente utilizados para o processamento de dados massivos relacionados a redes de dispositivos IoT, entre outras aplicações. A computação em nuvem (*cloud computing*) é aplicada a vários problemas e neste trabalho seu papel em sistemas IoT é fornecer vastos recursos e garantias e em que dispositivos enviam todos dados relevantes ao sistema. O segundo e terceiro ambiente são computação de borda (*edge computing*) e a computação em névoa (*fog computing*), que utiliza os recursos computacionais distribuídos presentes em nós localizados entre os dispositivos de borda e a nuvem, com diversas intenções, desde privacidade até redução de latência.

A computação em nuvem (*cloud computing*), ou simplesmente nuvem (*cloud*), habilita o acesso através da rede a um grupo compartilhado de recursos de computação configuráveis, como servidores, redes, aplicações, armazenamento, etc. Tais recursos podem ser provisionados ou liberados sob demanda rapidamente com o mínimo esforço de gerenciamento e mínima interação com o provedor destes recursos (MELL; GRANCE, 2012).

As principais características do ambiente *cloud computing*, segundo Mell e Grance (2012) são: Serviço sob Demanda, Amplo acesso à rede, Agrupamento de recursos, Elasticidade e Serviço mensurado. Segundo, Mell e Grance (2012), a implantação da Computação em Nuvem pode ocorrer através dos seguintes modelos: privada, comunitária, pública, híbrida. Das implantações, a pública é a mais comum, sendo gerenciada e operada por um provedor de nuvem e a infraestrutura é provisionada e oferecida para uso público.

A computação de borda (*edge computing*) refere-se às tecnologias que permitem que a computação seja executada na borda da rede. Define-se borda ou *edge* como qualquer recurso de computação e de rede ao longo do caminho entre as fontes de dados e os data centers da nuvem (SHI et al., 2016). Na borda, é possível fazer armazenamento, processamento e descarregamento de dados, assim como distribuir as requisições e entregar os serviços das nuvens aos usuários. Shi et al. (2016) ressalta que essas capacidades (dentre outras) dos nós da borda (*edge nodes*) possibilitam que a computação de borda reduza a latência na resposta da nuvem, pré-processando os dados nos nós da borda, aproveitando melhor a banda e a transmissão de dados, e também consumindo menos recursos de computação na nuvem. Além disso, o autor ainda acrescenta que a computação de borda pode aumentar a privacidade dos dados, uma vez que eles podem ser processados no próprio dispositivo final.

A computação de borda tenta trazer a computação mais próxima das fontes de dados. Como é observado na figura, os componentes desse tipo de computação podem ser tanto produtores como consumidores, não só requisitando serviços e conteúdo da nuvem, mas também realizando tarefas da nuvem. Algumas aplicações da computação de borda incluem: análise de vídeo; em sistemas críticos para redução de latência; descarregar a nuvem de parte da computação; privacidade dos dados produzidos, mantendo-os fora de ambientes públicos; redução das cargas de dados na rede e processamento distribuído de sensoriamento massivo em cidades inteligentes (SHI et al., 2016).

Dastjerdi e Buyya (2016) e IEEE Communications Society (2018) mencionam que a enorme massa de dados gerados por ambientes IoT pode ser processada em nuvem, entretanto a latência produzida pela transferência desses dados para a nuvem e o retorno do resultado pode não ser toleradas por sistemas críticos que sejam sensíveis a latência (monitoramento de saúde e resposta a emergências). IEEE Communications Society (2018) ainda acrescenta que enviar tantos dados à nuvem para processamento e armazenamento pode ser ineficiente e não escalável, devido à saturação de dados na rede. O ambiente *edge computing* foi proposto para trazer o processamento e armazenamento para os dispositivos de borda tentando solucionar esses problemas. Porém, dispositivos de borda comumente não podem lidar com várias aplicações IoT competindo pelos seus recursos limitados, o que poderia causar a contenção dos recursos e o aumento na latência do processamento (DASTJERDI; BUYYA, 2016). Portanto, para solucionar estas questões de latência e capacidade limitada dos dispositivos de borda, a computação em névoa foi proposta.

A computação em névoa (*fog computing*) é um paradigma que distribui as capacidades de computação, armazenamento e rede entre os nós próximos das fontes dados e dos dispositivos finais, mas não necessariamente localizados na borda, dando a esses nós características de uma nuvem (BONOMI et al., 2012; DASTJERDI; BUYYA, 2016; IEEE Communications Society, 2018). Esse tipo de computação evita a sobrecarga dos dispositivos de borda. Bonomi et al. (2012) e Dastjerdi e Buyya (2016) consideram computação em névoa como complementar da computação em borda, podendo a computação em névoa aproveitar os recursos da nuvem e da borda. IEEE Communications Society (2018) considera que a principal diferença entre esses dois tipos de computação está no número de camadas. Enquanto *edge computing* tem camadas menores, pois atua só nos dispositivos de borda, *fog computing* tem mais camadas e um modelo hierárquico, pois não atua só na camada de borda.

Segundo Bonomi et al. (2012) e Dastjerdi e Buyya (2016), as principais características da computação em névoa são:

- **Mobilidade:** é essencial que as aplicações *fog* sejam capazes de se comunicar com dispositivos móveis, por exemplo, utilizando protocolos que considerem a mobilidade dos nós;
- **Heterogeneidade:** os nós nesse tipo de paradigma possuem configurações e formatos diferentes e podem estar implantados em ambientes distintos;
- **Baixa Latência:** computação em névoa foi proposta para atender aplicações que requeiram baixa latência (monitoramento de saúde, jogos, realidade aumentada, etc.);
- **Distribuição geográfica:** computação em névoa pode possuir milhares de sensores e dispositivos distribuídos geograficamente, com consciência de suas localizações (*location awareness*);
- **Alto número de nós:** seguindo os ambientes IoT, a computação em névoa pode ser composta por milhares de nós;
- **Interoperabilidade e federação:** os componentes da computação em névoa devem ser capazes de interoperar, e o serviços devem ser federados ao longo de diferentes domínios;
- **Uso de fluxo de dados e aplicações em tempo real:** a computação em névoa pode envolver aplicações que processam em lote, mas na maior parte das vezes envolve aplicações com requisito de processamento em tempo real, e para isso fazem o uso de fluxo de dados. Por exemplo, os sensores de um rede IoT escrevem a informação no fluxo de dados, a informação é processada, ações são inferidas e traduzidos em ações nos componentes atuadores.

Algumas aplicações para computação em névoa são: cidades inteligentes e semáforos inteligentes que enviam sinais de alerta aos veículos e coordenam os sinais verdes com outros semáforos através de sensores (veículos, pedestres, ciclistas); na área de saúde, para monitorar e prever situações de pacientes que estão conectados a sensores; em prédios inteligentes, que são dotados de sensores de umidade, temperatura, qualidade do ar, ocupação, sendo que a partir das informações deles, é possível alertar os ocupantes do prédio em algum caso de emergência.

2.2 Mineração de Dados e Fluxo de Dados

A Mineração de Dados é o processo de descoberta de padrões em conjuntos de dados utilizando métodos derivados de aprendizagem de máquina, estatística e banco de dados (GABER; ZASLAVSKY; KRISHNASWAMY, 2005). Além de mineração de dados tradicional, *Big Data* trata de conjuntos de dados que não podem ser processados em tempo viável, devido a limitações como memória ou armazenamento principal.

Definição 1. *Um Fluxo de Dados S é uma sequência massiva, potencialmente ilimitada de exemplos multi-dimensionais $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n, \ldots$ recebida em instantes $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_n, \ldots$ (AGGARWAL et al., 2003).*

Além da dimensão de armazenamento, outra dimensão que afeta a maneira como dados são modelados e manipulados é o tempo. Técnicas e algoritmos de mineração de fluxo de dados atendem a esses desafios utilizando restrições como apenas uma leitura do conjunto de dados e baixo tempo de processamento na construção de seus algoritmos (GAMA; RODRIGUES, 2007; GABER; ZASLAVSKY; KRISHNASWAMY, 2005).

As características de fluxos de dados e mineração de dados e os requisitos de seu processamento regularmente superam as capacidades computacionais de um único nó computacional convencional, de forma que a distribuição dos requisitos em múltiplos nós computacionais em um sistema distribuído pode ser necessária (GABER; ZASLAVSKY; KRISHNASWAMY, 2005).

Para a construção de sistemas que apliquem técnicas de mineração de fluxos de dados são necessárias bibliotecas e plataformas (*frameworks*) que são abordadas na Seção 2.3.

2.3 Arquiteturas e Plataformas de Processamento de Fluxos

Tradicionalmente, aplicações foram construídas com um sistema gerenciador de banco de dados (SGBD) relacional ou não-relacional associado. Essa arquitetura, nomeada de "arquitetura totalmente incremental" por Marz e Warren (2015), foi evoluída e simplificada iterativamente durante décadas de uso, porém ela não é adequada para sistemas em

tempo real, como os sistema de fluxo de dados. O volume e a velocidade de dados em um *Data Stream* leva à necessidade de distribuir o processamento, acrescentando poder computacional a cada nó adicionado. Porém, desafios como comunicação eficiente e sincronização de estado entre os nós, assim como tolerância a falhas, aumentam a complexidade de construção de um sistema distribuído em relação a um sistema tradicional.

Para mitigar problemas associados à construção de sistemas *Big Data* e *Data Streams*, arquiteturas de processamento de fluxo de dados distribuído foram propostas, como a arquitetura *Lambda* (MARZ; WARREN, 2015) e *Kappa* (KREPS, 2014), além de diversas plataformas, tanto de *Big Data* com características de tempo real, como especializadas em fluxo de dados.

MapReduce é a primeira plataforma de processamento de conjuntos massivos de dados que atingiu uso generalizado. Nessa implementação, uma biblioteca gerencia a distribuição, paralelização, tolerância a falhas e balanceamento de carga. Ao usuário da biblioteca resta implementar duas funções: *Map*, que recebe um par ordenado $(chave, valor)$ e emite um conjunto de pares intermediários na mesma estrutura; *Reduce*, que recebe uma chave e um conjunto de valores gerado pelo agrupamento de pares com essa mesma chave (DEAN; GHEMAWAT, 2004).

Em prática, um *cluster MapReduce* tem centenas de processadores e o conjunto de dados é armazenado em um sistema de arquivos distribuído que é lido pela plataforma com programas escritos por usuários sendo executados sob supervisão de um nó mestre. Essa implementação tem esquema geral de processamento em lotes que não atende o requisito de baixa latência. *MapReduce* é uma das principais influências na criação da arquitetura *Lambda* (MARZ; WARREN, 2015).

Apache Hadoop é uma coleção de ferramentas, incluindo: *Hadoop Distributed File System* (HDFS, um sistema de arquivos distribuído), *Hadoop YARN* um gerenciador de recursos em cluster e escalonador de trabalhos e, *Hadoop MapReduce*, um sistema baseado em *YARN*, implementando o modelo *MapReduce* (Apache Hadoop, 2020).

Apache Spark, analogamente ao *Hadoop*, é um *framework* para construção de sistemas de computação distribuída em *cluster*, com garantias de tolerância a falhas. No entanto, o modelo de processamento diverge significativamente do tradicional *MapReduce*, utilizando em lugar do HDFS um multiconjunto imutável distribuído (*Resilient Distributed Dataset* - RDD) com um escalonador de trabalhos representados por grafos acíclicos direcionados (*directed acyclic graph* - DAG), otimizador de consultas e motor de execução (Apache Spark, 2020).

Uma das extensões de *Apache Spark* é *Spark Streaming*, que é um sistema de processamento de fluxo de dados escalável e tolerante a falhas (ZAHARIA et al., 2016a; ZAHARIA et al., 2016b). *Spark Streaming* implementa processamento incremental de fluxo de dados usando o modelo de fluxos discretizados em que dividem-se os dados de entrada em micro-lotes (ex: a cada 100 milissegundos) e combinam-se regularmente com o estado nos RDDs

para produzir novos resultados (ZAHARIA et al., 2016a). Essa estratégia traz benefícios sobre os sistemas de fluxos de dados distribuídos tradicionais, pois permite a consistência e recuperação de falhas rapidamente, devido à linhagem de RDD (*RDD lineage*) e à combinação do fluxo de dados com consultas em lotes e interativas (ZAHARIA et al., 2016b; LOPEZ, 2018).

Apache Storm é um sistema de computação tolerante a falhas em tempo real que facilita o processamento de fluxo de dados (FOUNDATION, 2020; LOPEZ, 2018). Ao invés de executar trabalhos (*jobs*) como algumas ferramentas citadas anteriormente, *Apache Storm* executa topologias. Os *jobs* eventualmente finalizam, e as topologias executam continuamente até serem finalizadas por comandos. Uma topologia constitui-se de processos trabalhadores (*workers*) sendo executados em um *cluster* de nós que são gerenciados pelo nó mestre que além de coordenar e distribuir execução, monitora falhas. Uma topologia pode ser representada por um grafo de computação direcionado acíclico (DAG).

O *Apache Flink* é uma plataforma de processamento distribuído para computação com estado gerenciado (*stateful*) sobre fluxo de dados limitados (têm início e fim) e ilimitados (não têm fim definido) (Apache Flink, 2020). Essa plataforma segue um paradigma que abrange o processamento de fluxos de dados contínuos e o processamento em lote (CARBONE et al., 2015; LOPEZ, 2018). O *Apache Flink* pode ser integrado a vários gerenciadores de *cluster* comuns, como *Hadoop Yarn*, *Apache Mesos*, e *Kubernetes*, mas também pode ser configurado para ser executado como um *cluster stand-alone*. Já o acesso programático a essa plataforma pode ser feito através das linguagens Java, Scala ou Python.

2.3.1 Interface de Troca de Mensagens

Interface de Troca de Mensagens (*Message Passing Interface*) (MPI)[1] é um padrão que estabelece um protocolo de comunicação e define sintaxe e semântica para bibliotecas de troca de mensagens em ambientes de memória compartilhada. Este padrão, por meio de alguma implementação como OpenMPI e MPICH, permitem a construção de um sistema distribuído com um executável único (monólito) utilizando com abstração a passagem de mensagens.

O modelo de programação para programas MPI Single Program Multiple Data (SPMD) vs Single instruction, multiple data (SIMD) multiple instruction, multiple data (MIMD)

, modelo de comunicação (empacotamento, localização de dados).

Configuração de execução.

Computation over Communication (CCR).

[1] Disponível em <https://www.mpi-forum.org/>.

2.4 Detecção de Novidade

No âmbito de classificação de dados, parte da área de aprendizado de máquina, os métodos de Detecção de Novidades em Fluxos de Dados (*Novelty Detection in Data Stream*) (ND-DS) lidam com o reconhecimento e a classificação de exemplos que diferem de exemplos anteriores (MARKOU; SINGH, 2003; PERNER, 2009; GAMA; RODRIGUES, 2010). Esses métodos tratam da classificação em fluxos de dados que evoluem com o tempo, levando em consideração as características desse tipo de fluxos.

Tratando-se de fluxos de dados contínuos, são características dos padrões observados: evolução de conceito (*Concept Evolution*) em que novos padrões podem surgir; desaparecimento ou recorrência de conceito, em que padrões podem desaparecer e também podem reaparecer; mudança de conceito (*Concept Drift*, também nomeado deriva ou desvio) onde um padrão gradualmente se transforma; presença de ruído e *outliers* (GAMA; RODRIGUES, 2010).

Os métodos de ND-DS são aplicados a diversos problemas como detecção de intrusos (COULL et al., 2003; SPINOSA; CARVALHO; GAMA, 2008; VIEGAS et al., 2019; CASSALES et al., 2019), detecção de falhas (ZHANG et al., 2006), diagnósticos médicos (PERNER, 2009), detecção de regiões de interesse em imagens (SINGH; MARKOU, 2004), detecção de fraudes (WANG et al., 2003; ABDALLAH; MAAROF; ZAINAL, 2016), filtros de spam (HAYAT; HASHEMI, 2010) e detecção de variações comportamentais em um jogador (VALLIM et al., 2013).

Alguns métodos de ND-DS utilizam tratam de novidades como uma classificação de uma ou duas classes (binariamente) onde um conceito representa a classe normal e as anomalias são representadas pela falta de conceito no modelo ou como um segundo conceito no modelo. Além da abordagem de classificação binária, múltiplos conceitos em um mesmo conjunto de dados, para isso é necessário abordar ND-DS como classificação multi-classe. Alguns métodos que abordam ND-DS como classificação multi-classe não atendem completamente características de conjuntos com evolução temporal, como *Concept Evolution* e *Concept Drift*, deixando de detectar múltiplos padrões que surgem simultaneamente num intervalo de avaliação (FARIA et al., 2016; GAMA; RODRIGUES, 2010).

A maioria dos métodos de ND-DS são construídos seguindo a abordagem de aprendizado *Offline-Online*. Essa abordagem estabelece que o método seja dividido em duas fases: a primeira fase (*Offline*) usa um conjunto de exemplos rotulados para deles extrair conceitos conhecidos e gerar um modelo; a segunda fase (*Online*) consome um conjunto ou fluxo de exemplos não rotulados e detecta padrões-novidade. Além de detectar padrões-novidade, alguns algoritmos classificam cada exemplo em um dos conceitos do modelo, ou marca o exemplo como desconhecido. Ainda na segunda fase, para atualizar o modelo, os exemplos marcados como desconhecidos são utilizados para a extração de novos conceitos ou variações em conceitos conhecidos (GAMA; RODRIGUES, 2010).

Dentre os métodos de ND-DS que baseiam-se em aprendizado *Offline-Online*, muitos

são baseados em algoritmos de agrupamento não supervisionados, tanto para construção do modelo inicial como na extração de novos conceitos dos exemplos não explicados pelo modelo marcados como desconhecidos (SPINOSA; CARVALHO; GAMA, 2009; MASUD et al., 2011; FARIA et al., 2013).

2.4.1 O algoritmo MINAS

Um algoritmo de ND-DS que tem recebido atenção nos últimos anos é o algoritmo MINAS, originalmente proposto por Faria et al. (2013), refinado por Faria, Carvalho e Gama (2016) e recentemente aprimorado por Silva (2018), com o uso de conceitos *Fuzzy*, e expandido por Costa (2019), para tratar problemas multi-rótulo além dos problemas multi-classe já tratados na versão original. Esse algoritmo segue a abordagem de duas fases no modelo *Offline-Online* e emprega algoritmos de agrupamento (*clustering*) não supervisionados como *K-means* e *CluStream*.

O algoritmo MINAS em sua fase *Offline* consome um conjunto de treinamento contendo exemplos etiquetados. Esse conjunto de treinamento é dividido em grupos usando como chave a etiqueta, e para cada grupo de exemplos o método de agrupamento é executado. O método de agrupamento objetiva resumir um conjunto maior de exemplos em um conjunto menor de *micro-clusters*.

Um *micro-cluster* é uma tupla de quatro components $(N, \mathbf{LS}, \mathbf{SS}, T)$ derivados dos exemplos representados por este *micro-cluster* onde: N é o número de exemplos, **LS** é o vetor soma linear dos exemplos, **SS** é o vetor quadrada dos exemplos, T é o instante de chegada do último exemplo adicionado ao *micro-cluster*. Deste sumário extrai-se, entre outras estatísticas, o centro e raio que são utilizados na operação de classificação da fase *Online*. A cada *micro-cluster* é adicionada a etiqueta do grupo original e todos *micro-clusters* são arranjados em um único conjunto formando o modelo de decisão.

Na fase *Online*, listada no Algoritmo 1, o algoritmo MINAS executa três operações: classificação de novos exemplos, detecção de padrões-novidade e atualização do modelo de decisão (FARIA; CARVALHO; GAMA, 2016).

A primeira operação é a classificação, onde exemplos do fluxo de dados são consumidos e avaliados pelo modelo de decisão. O modelo de decisão avalia cada exemplo calculando a distância euclidiana entre o exemplo e todos *micro-clusters* do modelo, selecionando o *micro-cluster* de menor distância. Se a distância entre o exemplo e o centro do *micro-cluster* for menor que o raio do *micro-cluster*, o exemplo é classificado com a etiqueta do *micro-cluster* e o sumário estatístico do *micro-cluster* é atualizado. Caso a distância (mínima no modelo) seja maior que o raio, o exemplo é marcado como desconhecido e armazenado em conjunto próprio (FARIA; CARVALHO; GAMA, 2016).

A segunda operação da fase *Online* é a detecção de padrões novidade, que é executada quando o tamanho do conjunto de desconhecidos é maior que um parâmetro predefinido.

Entrada: Modelo, fluxoEntrada
Saída : fluxoSaída
Parâmetros: janelaLimpeza, gatilhoDetecçãoNovidade

```
1  Função MinasOnline(Modelo, fluxoEntrada):
2      Desconhecidos ← ∅, ModeloAntigo ← ∅ ;
3      últimaLimpeza ← 0 , proximaNovidade ← 0;
4      para cada exemplo_i ∈ fluxoEntrada faça
5          maisPróximo ← clusterMaisPróximo (exemplo, Modelo);
6          se maisPróximo.distância < maisPróximo.cluster.raio então
7              exemplo.etiqueta ← maisPróximo.cluster.etiqueta;
8              maisPróximo.cluster.últimoUso ← i ;
9          senão
10             exemplo.etiqueta ← "desconhecido";
11             Desconhecidos ← Desconhecidos ∪ exemplo;
12             se | Desconhecidos | ≥ gatilhoDetecçãoNovidade então
13                 novidades ← DetecçãoNovidade (Modelo ∪ ModeloAntigo, *Desconhecidos);
14                 Modelo ← Modelo ∪ novidades;
15             se i > ( últimaLimpeza + janelaLimpeza) então
16                 Modelo ← moveModeloAntigo (Modelo, *ModeloAntigo, últimaLimpeza);
17                 Desconhecidos ← removeExemplosAntigos (Desconhecidos, últimaLimpeza);
18                 últimaLimpeza ← i;
19         fluxoSaída.adicione(exemplo);
```

Algoritmo 1: Interpretação do algoritmo MINAS baseado em (FARIA; CARVALHO; GAMA, 2016)

Esta operação utiliza o agrupamento (*clustering* descrito na fase *Offline*) e valida os *micro-clusters* gerados verificando sua representatividade e coesão.

Para atribuição de etiquetas aos *micro-clusters* gerados, o algoritmo MINAS encontra no modelo atual o *micro-cluster* mais próximo pela distância euclidiana e classifica em dois tipos de conceitos. Se a distância é menor que um parâmetro predefinido, o novo *micro-cluster* gerado recebe como etiqueta o valor de extensão de conceito conhecido. Caso contrário, se o novo *micro-cluster* está mais distante, um novo conceito foi encontrado e a etiqueta marca um padrão novidade. Após a atribuição da etiqueta do novo *micro-cluster*, ele é adicionado ao modelo de decisão.

The Novelty Detection function, illustrated in Algorithm 2, groups the instances to form new clusters, and each new cluster is validated to discard the non-cohesive or unrepresentative ones. Valid clusters are analyzed to decide if they represent an extension of a known pattern or a completely new pattern. In both cases, the model absorbs the valid clusters and starts using them to classify new instances.

O algoritmo MINAS, como já foi discutido na Seção 2.4.1, classifica exemplos e detecta novidades em DS e considera em sua composição *concept drift* e *concept evolution*, sendo capaz de classificar como extensão de classe conhecida e identificar padrões novidade sem intervenção de especialista (FARIA; CARVALHO; GAMA, 2016). Neste trabalho,

Parâmetros: minExemplos, fatorNovidade
Função `DetecçãoNovidade(`*Modelo, Desconhecidos*`):`
novoModelo ← ∅;
para cada *novoCluster in* `agrupamento` *(Desconhecidos)* **faça**
  se | *novoCluster.exemplos* | ≥ minExemplos **então**
    maisPróximo ← `clusterMaisPróximo` (novoCluster, Modelo);
    se *maisPróximo.distância* < *(maisPróximo.cluster.raio* × fatorNovidade*)* **então**
      novoCluster.etiqueta ← maisPróximo.cluster.etiqueta;
      novoCluster.tipo ← "extensão";
    senão
      novoCluster.etiqueta ← proximaNovidade;
      proximaNovidade ← proximaNovidade +1;
      novoCluster.tipo ← "novidade";
    Desconhecidos ← Desconhecidos − novoCluster.exemplos;
    novoModelo ← novoModelo ∪ novoCluster;
retorna *novoModelo*;

Algoritmo 2: Detecção de Novidades interpretada do algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016).

consideram-se algoritmos derivados do algoritmo MINAS aqueles apresentados em trabalhos publicados após 2016, que estendem a implementação original seguindo sua estrutura básica.

FIX-ME

O algoritmo MINAS foi relevante e motivou extensões como (DA SILVA et al., 2018; JÚNIOR et al., 2019)...

O algoritmo FuzzyND, derivado do algoritmo MINAS foi proposto por DA SILVA et al. (2018). FuzzyND incrementa o algoritmo original, aplicando a ele teorias de conjuntos *fuzzy* pela modificação da representação dos *clusters*. O algoritmo MINAS-LC foi proposto por Costa (2019) e trata a classificação multi-rótulo, porém não trata evoluções de conceito (*Concept Evolution*). Estes trabalhos foram importantes para entender MINAS e ajudaram a construir uma ideia geral do algoritmo e como proceder para criação de uma versão distribuída.

Algumas propostas de modificação do algoritmo MINAS notáveis discutidas durante o desenvolvimento deste trabalho porém alguns exemplos mencionados não abordados são (DA SILVA et al., 2018; SILVA, 2018; JÚNIOR et al., 2019):

a) diferentes métodos de cálculo de distância entre pontos além da distância euclidiana;

b) a mudança de representação de *clusters*, atualmente hiper-esferas (COSTA, 2019), para hiper-cubos tratando *data sets* onde as características representadas pelas dimensões são completamente independentes;

c) um modo interativo onde o *cluster* é formado, mostrado ao especialista que o classifica como inválido (ruído ou não representativo) ou válido, podendo conter uma ou

mais classes e, se contiver mais que uma classe corte em grupos menores até conter somente uma classe;

d) ainda considerando interação com especialista, a possibilidade de injetar um exemplo não pertencente a uma classe, ou seja, marcar o exemplo como não pertencente a uma classe para mantê-lo na memória de desconhecidos e, eventualmente forçar criação de um *cluster* que represente uma classe geometricamente próxima mas semanticamente distinta;

e) na fase *offline* a verificação de sobreposição de *clusters* pertencentes a classes distintas e tratamento adequado.

Capítulo 3

Trabalhos Relacionados

Este Capítulo trata dos trabalhos relacionados e apresenta aspectos do estado da arte dos tópicos Detecção de Novidades em Fluxos de Dados, e Processamento Distribuído de Fluxos de Dados.

Nesta Capítulo, abordam-se trabalhos que aplicam algoritmos de detecção de novidades em ambiente de processamento distribuído de fluxo de dados em tempo real. Um sumário dos trabalhos abordados pode ser visto na Tabela 1.

trabalhos genéricos: duas linhas por trabalho, destaque, contribuição, relação.

Adicionar trabalhos fazem parte, sem fechar numa plataforma.

Adicionar considerações sobre: Distribuição, Paradigma (cloud, fog, edge), algoritmo (envelhecimento de modelo, atualização do modelo)

Tabela 1 – Sumário dos trabalhos relacionados

Trabalho	Plataforma	Técnica	Atualização de Modelo	Conjunto de dados	Métricas
Ferramenta BigFlow (VIEGAS et al., 2019)	*Python*, *flowtbag*, *Apache Kafka* e *Apache Flink*	*Hoeffding Tree*, *OzaBoosting*, *Leveraging Bag* e comitê	Semanal semi-automática	*MAWILab*	Acurácia (geral e por classe), Taxa de bytes
Ferramenta CATRACA (LOPEZ, 2018)	*Virtual Network Function*, *Apache Kafka* e *Apache Spark*	PCA, SFS, e SVM-RFE	Contínua semi-automática	NSL-KDD, GTA/UFRJ e NetOp	Acurácia, precisão, sensibilidade e F1-score
Arquitetura IDSA-IoT (CASSALES et al., 2019)	Java, *Apache Kafka* e *Python*	ECSMiner, AnyNovel e MINAS	Contínua Automática	*Kyoto 2006+*	*Fnew*, *Mnew* e *erro*

3.1 Ferramenta BigFlow

Proposta por Viegas et al. (2019), a ferramenta BigFlow é um sistema de detecção de intrusão em rede (*Network Intrusion Detection System*, NIDS) baseado em detecção de anomalias. Duas abordagens, detecção por assinatura e detecção por anomalia, são de uso frequente, como o mecanismo de detecção de intrusão na construção de NIDS. Para a detecção de novos tipos de ataque (*zero day*), a abordagem de detecção por anomalia é vantajosa, em contraste com a abordagem de detecção por assinatura, devido ao tempo de resposta (que envolve a identificação e criação de uma assinatura), grande demais para prevenir esse tipo de intrusão.

A ferramenta BigFlow é composta pelos módulos de extração de atributos e de aprendizado confiável. O módulo de extração de atributos é responsável por coletar pacotes da rede monitorada, transformar esses pacotes em fluxos com estatísticas de comunicação e enviar informações desses fluxos como exemplos para o módulo de aprendizado confiável. O módulo de aprendizado confiável, é composto pelos submódulos: submódulo classificador, responsável por classificar exemplos; submódulo de verificação, responsável por verificar o resultado de classificação; submódulo de exemplos rejeitados, responsável por requisitar a um especialista etiquetas para exemplos rejeitados e; submódulo de atualização incremental, que atualiza e distribui o modelo aos classificadores.

Viegas et al. (2019) destaca que *data sets* adequados para NIDS são poucos, devido ao conjunto de qualidades que os mesmos devem atender, como realismo, validade, etiquetamento, grande variabilidade e reprodutibilidade (disponibilidade pública).

Para avaliar o desempenho de NIDS, o *data set* MAWIFlow é proposto por Viegas et al. (2019). Este *data set* é derivado do dos traços de pacotes no *backbone* WIDE, ponto de amostras F *data set Packet traces from WIDE backbone, samplepoint-F*, composto por seções de captura de pacotes diárias de 15 minutos de um link de 1Gbps entre Japão e EUA, com início em 2006 continuamente até hoje, anonimizados e etiquetados por MAWILab (MAWI Working Group Traffic Archive, 2020; FONTUGNE et al., 2010). Desse *data set* original, o *data set* MAWIFlow utiliza apenas os eventos de 2016, dos quais 158 atributos são extraídos resultando em 7.9TB de captura de pacotes. Além disso, os dados são estratificados para redução de seu tamanho a um centésimo, mantendo as proporções de etiquetas (Ataque e Normal), facilitando o compartilhamento e avaliação de NIDS, além de atender às qualidades anteriormente mencionadas.

Com o *data set* MAWIFlow reduzido a 62 atributos, foram avaliados quatro classificadores da literatura em dois modos de operação. O primeiro modo de operação usa somente a primeira semana do ano como conjunto de treinamento e as demais como conjunto teste. O segundo modo usa o conjunto da semana anterior como treinamento e o conjunto da semana seguinte como teste. Comparando os resultados entre os modos de operação, os autores demonstram que a qualidade da classificação reduz-se com o tempo, quando não há atualização frequente do modelo classificador.

Com base na avaliação dos classificadores da literatura, para a ferramenta BigFlow é proposta a utilização de 4 algoritmos de classificação com capacidade de atualização, sendo todos variações de árvore de decisão *Hoeffding* (VIEGAS et al., 2019; DOMINGOS; HULTEN, 2000). A avaliação da ferramenta foi executada de maneira semelhante à avaliação dos algoritmos da literatura, onde o conjunto de dados da primeira semana foi usado para treinamento e o conjunto de dados do restante do ano como conjunto de teste. Além do conjunto de treinamento, o modelo é atualizado semanalmente com base nas instâncias rejeitadas pelo submódulo de verificação.

Quanto à distribuição do processamento, a ferramenta BigFlow faz uso das plataformas *Apache Flink* e *Apache Kafka*. Em especial, destaca-se o uso do serviço gerenciador de trabalhos (*Job Manager*) e as múltiplas instâncias do serviço gerenciador de tarefas (*Task Manager*).

Em conclusão, a ferramenta BigFlow demonstra capacidade de classificação e detecção de anomalias em fluxos de dados de alta velocidade no contexto de detecção de intrusão. No entanto, a atualização semanal e, mais importante, dependendo de avaliação de um especialista não é ideal para detecção de novidades e respectiva ação sobre a descoberta de novos padrões.

3.2 Ferramenta CATRACA

O trabalho de Lopez (2018) aborda a detecção de ameaças a redes de computadores em tempo real e, para atingir esse objetivo, propôs a ferramenta CATRACA[1]. A ferramenta CATRACA é composta de três camadas: captura, processamento e visualização.

Na camada de captura, pacotes são capturados da rede e são geradas informações sumário de fluxos por uma aplicação *Python* utilizando a biblioteca *flowtbag*[2]. Esses sumários são enviados para um tópico de um sistema de fila de mensagens (*Apache Kafka*) hospedado em nuvem. Essa aplicação *Python* é distribuída como uma função virtual de rede (*Network Function Virtualization*) executada em dispositivos de rede virtuais.

A camada de processamento consome o tópico de mensagens que contém os fluxos da camada de captura e extrai características dos fluxos, detecta e classifica ameaças, enriquece o resultado (com localização geográfica por exemplo) e envia para a próxima camada na arquitetura por meio de um banco de dados (SGBD). A última camada da ferramenta fornece uma interface gráfica que apresentada a visualização dos fluxos processados bem como os conhecimentos extraídos e armazenados no banco de dados (SGBD). Ambas as camadas de processamento e visualização são executadas em ambiente de computação em nuvem (*cloud computing*).

[1] A ferramenta e sua documentação estão disponíveis em <http://gta.ufrj.br/catraca> e <https://github.com/tinchoa/catraca>.

[2] Disponível em <https://github.com/danielarndt/flowtbag> e <https://dan.arndt.ca/projects/netmate-flowcalc/>.

Para o desenvolvimento da ferramenta CATRACA, Lopez (2018) avaliou e comparou as plataformas de processamento de fluxo de dados em tempo real disponíveis (*Apache Storm*, *Apache Flink*, *Apache Spark Streaming*). A avaliação extraiu a velocidade máxima, em mensagens por minuto, de cada plataforma, variando a configuração de paralelismo em dois programas. Ambos consumiam dados de um tópico de um sistema de fila de mensagens (*Apache Kafka*) e produziam para outro tópico. O primeiro programa consiste de um detector de ameças composto por uma rede neural classificadora escrito em *Java*, que foi testado com o conjunto de dados sintético UFRJ/GTA (LOPEZ, 2018). O segundo programa conta quantas repetições de uma palavra existem em um fluxo de dados, exemplo muito comum em tutoriais de plataformas desse gênero, e é avaliado com um conjunto de *Tweets*.

Para o modelo de classificação, a ferramenta CATRACA utiliza o método árvore de decisão, escolhido pelo rápido treinamento e pela alta precisão e acurácia[3]. O modelo é criado na fase *Offline* e utilizado na classificação binária (normal e ameaça) da fase *Online*, sendo recalculado quando uma ameaça é encontrada.

Pra avaliação da ferramenta CATRACA dois conjuntos de dados são utilizados. O primeiro conjunto, UFRJ/GTA, é sintético e foi criado por uma simulação de rede de computadores, contendo 214 200 fluxos de rede e totalizando 95GB de pacotes capturados, este *data set* é composto de 24 atributos e 16 classes. O outro conjunto, referido como NetOp, foi coletado de um operador de rede que atendia 373 residências na cidade do Rio de Janeiro em 2017. O conjunto NetOp é formado por 5 TB de pacotes capturados e etiquetados por um detector de intrusão comercial.

Também para a avaliação da ferramenta CATRACA, foram utilizadas as métricas de qualidade de classificação acurácia, precisão, sensibilidade e F1M, com intervalo de confiança de 95%. As métricas de qualidade, dependendo do tamanho do conjunto, foram extraídas por métodos de avaliação amplamente utilizados para avaliar modelos de aprendizado de máquina (*machine learning*) como validação cruzada com proporção 70% do conjunto base para treinamento e 30% para teste. Para as métricas de escalabilidade foram utilizadas a latência e fator de aceleração *speedup factor* (latência observada com paralelismo 1 dividida pela latência observada com paralelismo variável).

Em conclusão, a ferramenta CATRACA apresenta uma arquitetura dividida em camadas alocadas em ambientes de névoa (*fog computing*) e nuvem (*cloud computing*). Essa ferramenta foi avaliada com métricas de qualidade, métricas de escalabilidade e dois conjuntos de dados relevantes. No entanto, o algoritmo de detecção de anomalias desenvolvido para a ferramenta consiste de um modelo de classificação pelo método árvore de decisão e a atualização do modelo durante a fase *Online* depende de todos os exemplos do último intervalo de atualização. Esse tipo de algoritmo de detecção de anomalias não é capaz de

[3] A precisão e a acurácia do método árvore de decisão podem estar associadas à independência entre as características (*features*) de cada exemplo, típico de conjuntos derivados de pacotes de rede.

lidar adequadamente com as características de fluxos contínuos de dados, como os descritos na Seção 2.4 (*Concept Drift*, *Concept Evolution*, limitado a ler o conjunto somente uma vez), que são atendidos por algoritmos de detecção de novidade.

3.3 Arquitetura IDSA-IoT

A arquitetura IDSA-IoT, proposta por Cassales et al. (2019), tem por objetivo monitorar uma rede local com dispositivos IoT e detectar tentativas de intrusão e alguma subversão do comportamento das transmissões destes dispositivos. O principal destaque da arquitetura é a distribuição de tarefas do sistema de detecção de intrusão entre nós na rede de borda (*fog computing*) e nós em nuvem pública (*cloud computing*). O objetivo dessa distribuição é a redução de latência, que torna inviável a hospedagem de um sistema detector de intrusão somente em ambiente *cloud computing*, e também possibilitar a análise de grandes volumes de dados por algoritmos de maior complexidade, que são de custo computacional proibitivo para nós de borda.

A arquitetura proposta é avaliada com três algoritmos de detecção de novidade: ECS-Miner (MASUD et al., 2011), AnyNovel (ABDALLAH et al., 2016) e MINAS (FARIA; CARVALHO; GAMA, 2016). A avaliação foi feita com o *data set Kyoto 2006+*, composto de dados coletados de 348 *Honeypots* (máquinas isoladas, equipadas com diversos softwares com vulnerabilidades conhecidas e expostas à Internet, com propósito de atrair ataques) de 2006 até dezembro 2015. Esse *data set* tem as características desejáveis de um conjunto para detecção de novidades como: realismo, validade, etiquetas previamente definidas, alta variabilidade, reprodutibilidade e disponibilidade pública. O *data set Kyoto 2006+* contém 24 atributos, 3 etiquetas atribuídas por detectores de intrusão comerciais e uma etiqueta distinguindo o tráfego entre normal, ataque conhecido e ataque desconhecido.

A avaliação da arquitetura foi realizada utilizando as métricas de qualidade *Fnew*, *Mnew* e *erro*. A métrica *Fnew* (ou Falso Positivo) é a fração dos exemplos de uma classe normal classificados com etiqueta novidade ou etiqueta extensão. A métrica *Mnew* (ou Falso Negativo) é a fração dos exemplos de uma classe novidade classificados com etiqueta normal. A métrica *erro* é a soma dos valores falso positivo e falso negativo dividida pelo número de exemplos classificados. Além das métricas de qualidade de classificação tradicionais, também foi medida a quantidade de requisições de classificação por especialista.

Outra avaliação dos algoritmos foi a extração de métricas de uso de recursos computacionais e tempo total de processamento em dispositivos limitados. Essa avaliação envolveu dois computadores. Para tanto, um computador pessoal com recursos convencionais produzia exemplos e adicionava como mensagens em um tópico no sistema de fila de mensagens *Apache Kafka*; já o outro computador, com recursos limitados, consumia as mensagens do tópico e classificava os exemplos.

Ambas as avaliações demonstraram o equilíbrio entre qualidade de classificação e velocidade ou uso de recursos. O algoritmo ECSMiner mostrou melhor qualidade de classificação, porém com velocidade inferior e maior consumo de recursos comparado aos outros algoritmos. Já o algoritmo MINAS, apesar de maiores valores na métrica *erro*, mostrou-se adequado para dispositivos limitados com baixo consumo de recursos computacionais e manteve a métrica *Fnew* constante e baixa. O algoritmo AnyNovel não apresentou consistência nos resultados e o consumo de recursos computacionais (memória) foi elevado.

A distribuição das tarefas em serviços proposta abre oportunidades para a discussão de diferentes métodos de distribuição dessas tarefas em diferentes ambientes computacionais. Contudo, o algoritmo MINAS ainda não foi implementado e avaliado com paralelismo, multi-processamento ou distribuição computacional, que são necessários para tratar fluxos de dados com grandes volumes e velocidades.

3.4 Conclusão

Os trabalhos discutidos nesse Capítulo têm temas complementares em áreas distintas. A área de aprendizado de máquina, com o tema detecção de novidades em fluxos de dados, preocupa-se em fornecer melhores previsões através de algoritmos classificadores que atendam as características de cada problema. A área de computação distribuída aborda os temas de processamento distribuído de fluxos contínuos em ambientes de computação em nuvem e em névoa, fornecendo métodos para processar grandes volume de dados com mínima latência.

Capítulo 4

Proposta e metodologia

Este Capítulo apresenta a proposta deste trabalho e a metodologia elegida para atingir os objetivos.

In this work, we investigate an appropriate architecture for performing ND-DS at the edge, as a means of allowing small IoT devices to filter and detect undesirable network behavior. Our approach is based on the IDSA-IoT architecture (CASSALES et al., 2019) and ND-DS techniques provide by the algoritmo MINAS algorithm (FARIA; CARVALHO; GAMA, 2016). Named sistema M-FOG, our distributed algorithm explores load balancing to enable low profile devices at the edge of the internet to also work on the classification and detection of unwanted traffic.

In this work, we propose and assess sistema M-FOG, a distributed data stream novelty detection system based on the algorithm algoritmo MINAS for securing IoT networks. sistema M-FOG implements a distributed version of algoritmo MINAS according to the IDSA-IoT architecture proposed in a previous work (CASSALES et al., 2019), to execute in the edge where small devices and constrained resources may be prevalent.

However, given the distributed nature and the typical use of small computing devices in IoT scenarios, new challenges arise:

(*i*) the classification phase of the algorithm must occur in parallel at different nodes;

(*ii*) the novelty detection phase, which provides the model evolution, must also be asynchronous;

(*iii*) the algorithm complexity (time and space) must allow it to be processed by modest computing devices (i.e., small memory and low processor performance).

NIDS monitor network traffic, and analyze the characteristics of each flow to identify any intrusion or misbehavior. However, this problem requires both fast and accurate response (COSTA et al., 2019): fast response is needed to have a proper reaction before harm can be cast to the network and to cope with the traffic without imposing loss or delay in the NIDS or observed network; accurate response is required as not to misidentify, especially the case of false positive that leads to false alarms. To achieve those goals, we leverage fog computing.

In common IoT scenarios, data is captured by small devices and sent to the cloud for any compute or storage tasks, but this is not feasible in a NIDS scenario. Fog computing infrastructure aims to offload processing from the cloud providers by placing edge devices closer to end-users and/or data sources.

In our proposal, fog and cloud computing resources are combined to minimize the time elapsed between a flow descriptor ingestion and intrusion alarm, performing the classification step of algoritmo MINAS running multiple classifier instances. After the initial classification, the resulting label can be used immediately, but if the sample is labeled as *unknown*, this sample must be stored and the novelty detection step will be triggered.

The overall sistema M-FOG architecture has two main modules, Classification and Novelty Detection, which implement the algoritmo MINAS main tasks. The Classification Module performs the same task of the algoritmo MINAS Online phase and is the focal point for parallelism and distribution in our proposal. It is replicated in the fog and runs on each cluster node, using a configurable number of threads (limited to the node CPU core count).

The Novelty Detection Module can also be replicated, the choice being one instance per local network, one global cloud instance, or both. This module also handles the homonymous task of algoritmo MINAS Online phase, receiving all the samples labeled with *unknown*, storing them in an internal *unknown-buffer*, and, when this buffer is full, performing the algoritmo MINAS Novelty Detection task (clustering followed by validation).

4.1 Políticas

O projeto desta arquitetura de detecção de novidades

O design de nossa arquitetura ND-DS distribuída inclui o particionamento do funcionalidades de algoritmo MINAS e estabelecendo os fluxos de dados apropriados entre diferentes atores. Mudanças no posicionamento e no comportamento podem ter impactos diferentes e devem ser escolhido com cuidado. As decisões após essas discussões podem ser organizadas em várias políticas, alguns deles eram recorrentes durante nossas discussões de implementação e são:

The design of our distributed ND-DS architecture includes partitioning the functionalities of algoritmo MINAS and establishing the appropriate data flows between different actors. Changes to placement and behavior can have different impacts and should be chosen with care. The decisions following these discussions can be organized in several policies, some of them were recurring during our implementation discussions and are:

- Regarding the allocation of the Novelty Detection Module:
  - At each fog node: patterns will be only detected if sufficient samples of them occur in the local observed network, use of the local node processing power, and a model synchronization mechanism between networks must be added;
  - In the cloud: detect patterns even when scattered on each local network, each sample with *unknown* label must be sent from edge to cloud implying increased internet link usage and increased delay between the appearance of a pattern, its detection and propagation to fog classifiers;
  - On both: local *unknown* buffer is maintained and novelty detection is local as well, once a sample is considered as noise or outlier it shall be sent to the cloud where the process repeats but with global data. This choice needs an even more complex model synchronization mechanism.
- Regarding the model cleanup (forget mechanism): Even when a global novelty detection is used, local models can be optimized for faster classification using the local model statistics by sorting by (or removing) least used clusters;
- Lastly, reclassification of *unknowns*: In the novelty detection task in algoritmo MINAS, the *unknown* sample buffer is effectively classified using the new set of clusters. In Algorithm 2, at the line 12, the new cluster valid (novelty or extension) includes the set of samples composing that cluster, thus, if this new label assignment was put forth to the system output it would introduce delayed outputs, more recent and perhaps more accurate. Also, it would change the system data stream behavior from a *map* (meaning each input has one output) to a *flatMap* (each input can have many outputs).

A Internet das Coisas (IoT) é composta por vastas quantidades de dispositivos conectados à Internet e distribuídos geograficamente. Com capacidades diversas providas por elementos como sensores e atuadores, esses dispositivos produzem e consomem Fluxos Contínuos de Dados (*data streams*) com diversos objetivos. Alguns cenários de IoT envolvem a mineração desses fluxos (*data stream mining*) em busca de padrões para tomada de decisão e, por vezes requerem também baixa latência. Para casos de baixa latência ou alta vazão, conexões adequadas para processamento em nuvem nem sempre são possíveis ou desejáveis; para esses casos, a computação em névoa (*fog computing*) é uma solução.

O tema de *data stream mining* envolve a classificação de novos elementos, que podem tanto estar relacionados aos dados ou aos metadados das comunicações, com base em um modelo. Porém, como *data streams* variam temporalmente e são ilimitados, as classes contidas em um *data stream* não são todas previamente conhecidas. A identificação e classificação de novas classes em *data streams* é denominada Detecção de Novidades (*Novelty Detection*, ND-DS) em *data streams*.

Sistemas que utilizam ND-DS para *data streams* gerados por dispositivos IoT devem utilizar algoritmos que considerem os desafios inerentes a fluxos de dados (*Concept Evolution* e *Concept Drift*) para adequada detecção de novidades e, para tanto, requerem processamento em arquiteturas que atendam os requisitos de volume de mensagens e latência de detecção. O algoritmo MINAS é adequado para esse caso, pois trata os desafios de *data stream mining*, porém não tem ainda implementação que atenda os requisitos de volume e latência, especialmente para aplicações IoT onde um ambiente de *fog computing* é atrativo.

Para preencher a lacuna de algoritmo de ND-DS em ambiente *fog computing*, propõem-se então o sistema M-FOG, uma implementação do algoritmo MINAS sobre a plataforma *Apache Flink*, que considera distribuição em um ambiente de *fog computing*. O sistema M-FOG descrito neste documento foi refinado com os resultados dos experimentos descritos na Seção 5.1 e poderá ser revisado ao longo da pesquisa conforme os resultados de outros experimentos evidenciarem obstáculos ou oportunidades de melhoria.

4.2 Descrição da Arquitetura Proposta

O sistema M-FOG utiliza em seus módulos a distribuição oferecida pela plataforma *Apache Flink* como paralelização, ou seja, utiliza uma instância de trabalho (*job*) por dispositivo de classificação, sendo que cada instância de trabalho aloca um gerenciador de tarefas por processador. Dessa forma, busca-se a escalabilidade no ambiente de *fog computing* para o módulo classificador. O módulo treinamento, por ser utilizado somente uma vez para gerar o modelo de classificação inicial, não tem impacto na escalabilidade geral do sistema. O módulo detector de novidades também é implementado na plataforma *Apache Flink* e, por ser hospedado em ambiente de *cloud computing*, herda as qualidades desse ambiente incluindo escalabilidade. O restante do sistema (módulo auxiliar *source*, módulo auxiliar *sink*, armazenamento de modelo) não é foco deste estudo e sua escalabilidade, desde que não afete a escalabilidade do módulo classificador e módulo detector de novidades.

4.3 Metodologia de Avaliação

A avaliação da proposta apresentada é feita por meio de métricas extraídas da literatura, divididas em duas partes: métricas de qualidade de classificação e métricas de escalabilidade. Métricas tradicionais de qualidade de classificação estabelecidas por trabalhos de aprendizado de máquina não são adequadas para avaliar detecção de novidades em *data streams* sem tratamento inicial. Felizmente, o tratamento necessário é estabelecido por Faria et al. (2013) e expandido por DA SILVA et al. (2018), Silva (2018), Júnior et al. (2019), Costa (2019). Além do tratamento estabelecido, as métricas tradicionais não são calculadas somente para o conjunto completo, e sim para cada exemplo classificado. Portanto, as métricas têm como índice o instante (n nas equações à seguir), informando a posição do exemplo em relação ao fluxo.

O tratamento estabelecido das métricas de qualidade para *data stream mining* define que as métricas sejam extraídas de uma matriz de erro de classificação multi-classe $\mathbf{E}_n$ (Equação 3), adaptada para detecção de novidade. A matriz de erro é preenchida com o número de eventos da classe c_i classificados com etiqueta l_j até o instante n. A Equação 1 representa o conjunto de classes presentes nos eventos do fluxo até o instante n e a Equação 2 representa o conjunto de etiquetas atribuídas pelo classificador a eventos até o mesmo instante.

$$\mathbf{C}_n = \{c_1, c_2, \cdots, c_M\} \tag{1}$$

$$\mathbf{L}_n = \{l_1, l_2, \cdots, l_J\} \tag{2}$$

$$\mathbf{E}_n = \begin{pmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,J} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ e_{M,1} & e_{M,2} & \cdots & e_{M,J} \end{pmatrix} \tag{3}$$

As métricas de qualidade de classificação selecionadas para avaliar a implementação do sistema M-FOG serão taxa de desconhecidos ($UnkR$ na Equação 4) (FARIA et al., 2013), acurácia média (acc na Equação 5) e Macro F-score ($Fscore$ na Equação 9, também referido na literatura por F1M) (SOKOLOVA; LAPALME, 2009; SILVA, 2018). As métricas são extraídas para todos os exemplos classificados (instantes n) da respectiva matriz de erro $\mathbf{E}_n$.

$$UnkR_n = \frac{1}{M}\sum_{i=1}^{M}\frac{\#Unk_i}{\#ExC_i} \tag{4}$$

$$acc_n = \frac{1}{M}\sum_{i=1}^{M}\frac{tp_i + tn_i}{tp_i + fn_i + fp_i + tn_i} = \frac{1}{M}\sum_{i=1}^{M}\frac{\#Acc_i}{\#ExC_i} \tag{5}$$

$$Precision_n = \frac{1}{M}\sum_{i=1}^{M}\frac{tp_i}{tp_i + fp_i} \tag{6}$$

$$Recall_n = \frac{1}{M}\sum_{i=1}^{M}\frac{tp_i}{tp_i + fn_i} \tag{7}$$

$$Fscore\beta_n = (\beta^2 + 1)\cdot\frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall} \tag{8}$$

$$Fscore1_n = 2\cdot\frac{Precision \cdot Recall}{Precision + Recall} \tag{9}$$

A transformação do fluxo de saída em uma matriz de erro é realizada no módulo auxiliar *sink*, onde tem-se disponível o fluxo original com as etiquetas corretas e o fluxo resultante da classificação. Esse módulo deve levar em consideração que pode haver reclassificação de um evento, previamente rotulado como desconhecido, em padrões oriundos de classe novidade ou extensão devido ao processo de detecção de novidades executado posteriormente ao surgimento do padrão em questão.

As métricas de escalabilidade selecionadas são: número de nós processadores, tipo de processadores, uso de memória, tempo de processamento, taxa de eventos processados e latência entre a produção e classificação de um evento.

Da implementação do sistema M-FOG é prevista a execução de experimentos com *data sets* diversos, em especial os *data sets* reais como *Kyoto 2006+*, que contenham evolução de conceitos. Os resultados desses experimentos irão conter as seguintes métricas:

a) Qualidade de classificação (taxa de desconhecidos, F1M);

b) Escalabilidade (número de processadores, volume processado, tempo decorrido);

c) Recursos computacionais utilizados (memória, tempo de processamento, operações de leitura e escrita).

Para a validação da corretude da implementação do sistema M-FOG com relação ao algoritmo MINAS original, as métricas de qualidade de classificação serão extraídas de ambas as Implementação e comparadas.

Capítulo 5

Implementação

5.1 Resultados preliminares

No desenvolvimento parcial desta pesquisa, algumas experimentações e algumas ferramentas de teste já foram desenvolvidas. Aspectos desses desenvolvimentos são descritos a seguir.

5.1.1 Implementação com *Python* e *Apache Kafka*

A primeira implementação e avaliação do sistema M-FOG realizada foi construída sobre a linguagem *Python* com o sistema de fila de mensagens *Apache Kafka* e a respectiva biblioteca de conexão. A escolha desse conjunto para a implementação ocorreu devido à ampla disponibilidade de bibliotecas de aprendizagem de máquina no ecossistema *Python* e, à simplicidade geral da linguagem. Na implementação desenvolvida, o sistema *Apache Kafka* recebe mensagens e as armazena em tópicos distribuídos em partições replicadas em nós de um *cluster*, gerenciados por um nó mestre e suportados pelo serviço de gerenciamento de configuração distribuída *Apache ZooKeeper*. A aplicação *Python* consome eventos através da interface *Consumer API*, que expõe a distribuição através da associação de um consumidor às partições mantidas pelo *Apache Kafka*.

Para essa implementação, havia a hipótese de que a distribuição de mensagens gerenciada pelo *Apache Kafka* se estenderia a processos consumidores, efetivamente distribuindo o volume de mensagens entre eles igualmente. No entanto, a hipótese foi refutada nos experimentos realizados. Os experimentos em questão foram compostos de 8 processos consumidores, um processo produtor, uma instância *Apache Kafka* com 8 partições em seu tópico principal e uma instância *Apache ZooKeeper* associada à instância *Apache*

Kafka. A hipótese foi refutada quando observou-se que o número de mensagens consumidas por um dos 8 processos representava a maioria (mais de 80%) do volume introduzido no sistema, o restante sendo distribuído entre outros 3 processos e o restante dos processos não recebia nenhuma mensagem. Portanto, a iniciativa de implementar o algoritmo MINAS em *Python* com *Apache Kafka* e atingir os objetivos de distribuição falhou, o que levou à reconsideração das plataformas escolhidas.

5.1.2 Implementação com *Apache Flink*

A segunda alternativa explorada teve por inspiração o trabalho de Viegas et al. (2019) e, como outro grupo de pesquisa já estava explorando o algoritmo na plataforma *Apache Spark*, a segunda implementação foi baseada na plataforma *Apache Flink*.

A plataforma *Apache Flink* tem modelos de processamento tanto de fluxos como em lotes. O modelo em lotes é implementado como extensão do modelo de fluxos e, apesar de não ser foco desse trabalho, mostrou-se útil para a construção do módulo treinamento, já que o conjunto consumido por esse módulo é limitado.

Um desafio encontrado durante o desenvolvimento da implementação do sistema M-FOG foi a falta de bibliotecas na plataforma *Apache Flink* que disponibilizem versões adaptadas à plataforma de algoritmos base para o algoritmo MINAS. Em especial, a ausência dos algoritmos *K-means* e *CluStream* gerou carga imprevista sobre o processo de desenvolvimento resultando no atraso do processo de desenvolvimento.

Esta implementação segue a arquitetura descrita na Seção 4.2 e as avaliações e resultados esperados descritos neste Capítulo 4 referem-se à implementação do sistema M-FOG na plataforma *Apache Flink*.

5.2 Implementação com MPI

The original MINAS algorithm has a companion unpublished implementation (*Ref*) written in Java using MOA library base algorithms such as K-means and CluStream, but our implementation only used K-means. Another difference between *Ref* and sistema M-FOG is the calculus of the cluster radius from the distances of elements forming the cluster and the cluster's center. *Ref* uses the maximum distance while sistema M-FOG uses the standard deviation of all distances as described in (FARIA; CARVALHO; GAMA, 2016).

The stream formats for input and output are also of note. As input, the algorithm takes samples ($\vec{v}$), which are a sequence of numbers with dimension d. In addition to $\vec{v}$, for both training and evaluation, the class identifier is provided as a single character, along with a unique item identifier (*uid*), which can otherwise be determined from the sample index in the stream.

As its output, the algorithm returns the original sample $\vec{v}$ followed by the assigned label. Adjustments can easily be made to provide the output results as a tuple containing *uid* and the assigned label.

Parameters: mpiNodeRank as mpiRank
Entrada: ModelSet, Sample Stream
Function `Mfog(`*ModelStream, InputStream, OutputStream*`):`
  ModelSet = ∅;
  ModelSetLock = **new** `Lock` ();
  se mpiRank == *0* **então** *root*
    new `Thread` (`Detector`, [OutputStream, ModelSet, ModelSetLock]);
    `Sampler` (InputStream, ModelSet, ModelSetLock);
  senão *leaf*
    new `Thread` (`modelReceiver`, [ModelSet, ModelSetLock]);
    `Classifier` (ModelSet, ModelSetLock);

Algoritmo 3: MFOG: main MPI entry-point.

Function `Classifier(`*ModelSet, ModelSetLock*`):`
  enquanto *True* **faça**
    sampe = `receive` (SampleType, root);
    se *sample* == EndOfStream **então break**;
    sample.label = unknown;
    with `readLock` *(ModelSetLock)*
      (distance, cluster) = `nearestCluster` (sample, ModelSet);
    se *distance* < *cluster.radius* **então**
      sample.label = cluster.label;
    `send` (root, SampleType, sample);
Function `modelReceiver(`*ModelSet, ModelSetLock*`):`
  enquanto *True* **faça**
    cl = `receive` (ClusterType, root);
    se *cl* == EndOfStream **então break**;
    with *writeLock(ModelSetLock)*
      ModelSet = ModelSet ∪ cl;

Algoritmo 4: MFOG Leaf Tasks: Model Receiver and Classifier.

For evaluation purposes, an sistema M-FOG implementation was made using MPI (*Open MPI 4.0.4*). The program is organized in a single program multiple data (SPMD) programming model, so a single version of the sistema M-FOG program was initiated on all nodes, being that one of them would perform the root role, while the others ran as leaves, the program entry point is illustrated on Algorithm 3. On the root process, a sampler thread is responsible for distributing the sampled flow information ($\vec{v}$) to the classifier nodes, using a round-robin load balancing scheme. The other thread on the root process is responsible for receiving the classification results and for processing the unknown samples in the search for novelties. The root process functions are illustrated

```
   Parameters: mpiClusterSize as mpiSize
 1 Function Sampler(InputStream, ModelSet, ModelSetLock):
 2 |  dest = 1;
 3 |  para cada  sample from InputStream faça
 4 |  |  se typeOf (sample) is Cluster então
 5 |  |  |  broadcast (ClusterType, sample, root);
 6 |  |  |  with writeLock (ModelSetLock)
 7 |  |  |  |  ModelSet = ModelSet ∪ sample;
 8 |  |  |  continue;
 9 |  |  send (dest, SampleType, sample);
10 |  |  dest = dest +1;
11 |  |  se dest > mpiSize então dest = 1;
   Parameters: cleaningWindow, noveltyDetectionTrigger
12 Function Detector(OutputStream, ModelSet, ModelSetLock):
13 |  lastCleanup ← 0;
14 |  enquanto  True faça
15 |  |  sampe = receive (SampleType, any);
16 |  |  se sample == EndOfStream então break;
17 |  |  OutputStream.append(sample);
18 |  |  se sample.label == unknown então
19 |  |  |  UnknownSet = UnknownSet ∪ sample;
20 |  |  |  se | UnknownSet | ≥ noveltyDetectionTrigger então
21 |  |  |  |  novelties = NoveltyDetection (p, ModelSet, *UnknownSet);
22 |  |  |  |  with writeLock (ModelSetLock)
23 |  |  |  |  |  ModelSet = ModelSet ∪ novelties;
24 |  |  |  |  para cada  cl in novelties faça
25 |  |  |  |  |  broadcast (ClusterType, cl, root);
26 |  |  |  se  sampe.uid > ( lastCleanup + cleaningWindow) então
27 |  |  |  |  UnknownSet ← removeOldSamples (UnknownSet, lastCleanup);
28 |  |  |  |  lastCleanup ← sampe.uid;
```

Algoritmo 5: MFOG Root Tasks: Sampler and Detector.

in Algorithm 5. Each leaf node runs a model adjustment thread and multiple (up to the number of cores) classifier threads. The leaf tasks are illustrated in Algorithm 4. The overall sequence of interactions is shown in Figure 1.

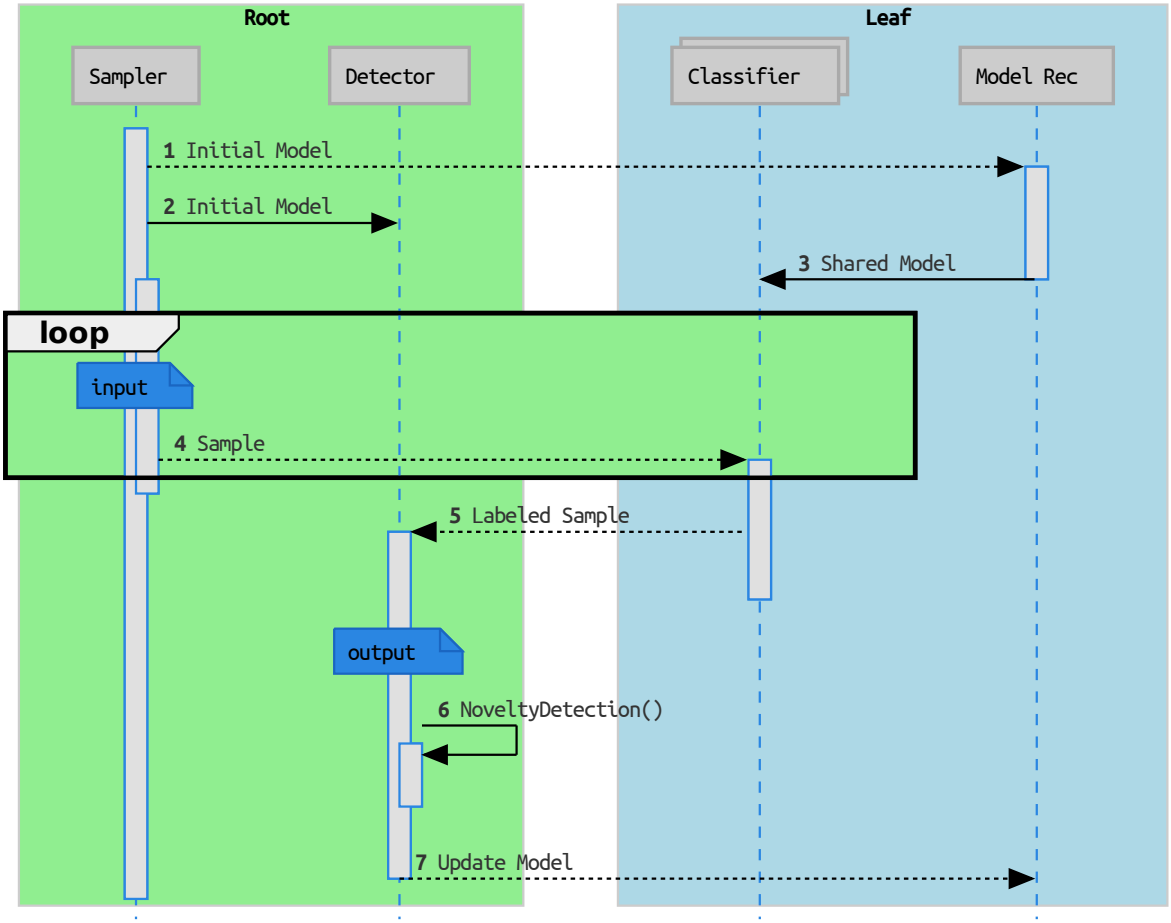


Figura 1 – sistema M-FOG life line overview.

Capítulo 6

Experimentos e Resultados

6.1 Ambiente de Teste

Aiming to evaluate our proposal for the effects of distributed novelty detection in a IoT NIDS scenario, we implemented an experimental setup, composed of three Raspberry Pi 3 model B single board computers connected via Ethernet Switch. The idea was to create a simple cluster simulating an IoT network with constrained resources at the edge of the network. This cluster stored all source code, binaries (compiled and linked in place) and data sets. In our setup, the data set is stored in the root's node SD card and is read for each experiment. All experiments were executed in this cluster for isolation of otherwise unforeseen variations and for safe software comparison with constant hardware.

The data set used is the December 2015 segment of Kyoto 2006+ data set[1] (Traffic Data from Kyoto University's Honeypots) (SONG et al., 2011a) containing 7 865 245 samples. From the original data set, we filtered only samples associated with normal traffic or known attack types identified by existing NIDS, and attack types with more than 10 000 samples for significance, as previously done by (CASSALES et al., 2019). The remaining samples then were normalized so each feature value space (e.g., IP Address, Duration, Service) is translated to the Real interval $[0, 1]$.

The resulting derived data set is then stored in two sets, training set and test set, using the holdout technique. However, for the training set we filter in only normal class resulting in 72 000 instances. For the test set we use 653 457 instances with 206 278 instances with "N" (normal) class and 447 179 instances with "A" (attack) class. Note that this choice results in possible overfitting for the normal class and, under-fitting for the attack class as the system first needs to detect a novel class and then add it to the model.

[1] Available at <http://www.takakura.com/Kyoto_data/>

Para realização dos experimentos, diversas configurações de ambientes são propostas. Os ambientes selecionados são: local, nuvem e névoa. As configurações consistem na distribuição de módulos da implementação sistema M-FOG sendo executadas em combinações de ambientes nuvem e névoa com variada quantidade de nós.

O ambiente local é composto por um único nó computacional, consistindo de um computador pessoal equipado com um processador de 8 núcleos, $16GB$ de memória e armazenamento em estado sólido (SSD) usado para o desenvolvimento e referência em comparações. O ambiente nuvem é provido pela utilização da infraestrutura de nuvem da Universidade Federal de São Carlos (Cloud@UFSCar[2]). O ambiente de névoa (*fog computing*) é composto por computadores de única placa (*Single Board Computer*) equipados com processador de arquitetura ARM de 4 núcleos, $1GB$ de memória, armazenamento em cartão SD (*SD-card*) e conectados por rede sem fio.

A combinação de diferentes distribuições tem por objetivo demonstrar padrões de latência e qualidade que podem afetar implantações em ambientes reais que não são geralmente destacados quando os experimentos são realizados em um único nó ou ambiente.

Faz parte também do ambiente de teste os conjuntos de dados (*data sets*) *KDD99* e *Kyoto 2006+* que foram selecionados por motivos distintos.

O *data set Kyoto 2006+* é o foco deste trabalho, pois contém dados ainda representativos (até 2015) e as característica desejáveis de um conjunto de dados (realismo, validade, etiquetas previamente definidas, alta variabilidade, reprodutibilidade e disponibilidade pública) são atendidas (SONG; TAKAKURA; OKABE, 2020; SONG et al., 2011b).

O *data set KDD99* é amplamente utilizado em trabalhos de detecção de anomalia. Porém, como não possui mais a característica de realismo, uma vez que foi construído em 1998, neste trabalho o *data set KDD99* é utilizado somente para que o leitor possa comparar com outros trabalhos (TAVALLAEE et al., 2009; PROTIĆ, 2018).

Os dois *data sets* mencionados e outros abordados em discussão e avaliados como relevantes são

6.2 Métricas e Visualizações

We have used two types of evaluation measurements for each experiment: a measure of the full experiment execution time and, a set of qualitative measurements extracted by a Python script.

Our evaluation script was build following reference techniques like multi-class confusion matrix with label-class association (FARIA; CARVALHO; GAMA, 2016) to extract classification quality measurements. This script takes two inputs, the test data set and the captured output stream, and outputs the confusion matrix, label-class association, final

[2] Disponível em <http://portalcloud.ufscar.br/servicos>

Tabela 2 – Sumário dos conjuntos de dados

Nome	Origem	Descrição	Acesso Público
KDD99 (TAVALLAEE et al., 2009; PROTIĆ, 2018)	Captura de Fluxos de rede com ataques simulados	41 atributos (sumário de fluxo), 23 classes, 4 898 431 instâncias, 709 MB	<https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>
Kyoto 2006+ (SONG et al., 2011b; PROTIĆ, 2018)	Captura de Fluxos de rede com HoneyPot	23 atributos (sumário de fluxo), 3 classes, 7 865 245 instâncias e 1.3 GB (dez-2015)	<https://www.takakura.com/Kyoto_data/new_data201704/>
CICIDS2017 (SHARAFALDIN; LASHKARI; GHORBANI, 2018)	Captura de Fluxos de rede com ataques simulados com perfil de trafego de 25 usuários normais e de 6 perfis de ataques durante 5 dias (1º dia sem ataque)	80 atributos (sumário de fluxo extraído de CICFlowMeter), 15 classes, 2 830 751 instâncias e 1.2GB em arquivos *pcap* e *csv*	<https://www.unb.ca/cic/datasets/ids-2017.html>
Radial Basis Function (RBF) da biblioteca *Massive Online Analysis* (MOA) *4CRE-V2*	Sintético gerado por função RBF da biblioteca MOA com características de mudança e evolução de conceito	Atributos ($\mathbb{R}$), exemplos, classes, evoluções e mudanças configuráveis	<https://sites.google.com/site/nonstationaryarchive/home>

quality summary with: *Hits* (true positive), *Misses* (Err), *Unknowns* (UnkR); and stream visualization chart with per example instance summary with novelty label markers.

In the confusion matrix $M = m_{ij} \in \mathbb{N}^{c \times l}$, computed by our evaluation script, each row denotes the actual class c and each column denotes the predicted label l present in the captured output stream. Thus, each cell $M_{c,l}$ contains the count of examples from the test data set of class c found in the output stream with the label l assigned by the under evaluation experiment.

For the data set under use, original classes are $c \in \{N, A\}$, and for the labels we have the training class *"N"*, *unknown* label *"-"* and the novelties $i \in \mathbb{N}$ so $l \in \{N, -\} \cup \mathbb{N}$.

Added to the original confusion matrix M are the rows *Assigned* and *Hits*. *Assigned* row represents which original class c (or if *unknown*, *"-"*) the label l is assigned to, this is computed by using the original class if $c = l$ or by associated novelty label to original class as described in (FARIA et al., 2015) section 4.1 (class from where the most samples came from). *Hits* row shows the true positive count for each label l with assigned class c, being the same value as cell $M_{c,l}$. The *Hits* row is also used to compute the overall true positive in the summary table and stream visualization chart. One complete matrix is shown in Tab. 3.

Tabela 3 – Reference implementation

Labels	-	N	1	2	3	4	5	6	7	8	9	10	11	12
Classes														
A	3774	438750	123	145	368	8	52	165	1	1046	161	2489	71	26
N	8206	193030	0	79	44	0	0	0	229	181	154	4066	289	0
Assigned	-	N	A	A	A	A	A	A	N	A	A	N	N	A
Hits	0	193030	123	145	368	8	52	165	229	1046	161	4066	289	26

Tabela 4 – Serial implementation

Labels	-	N	0	1	2	4	5	6	7	8	10
Classes											
A	16086	429765	94	995	104	0	23	3	29	46	34
N	12481	193642	3	94	0	47	0	0	0	11	0
Assigned	-	N	A	A	A	N	A	A	A	A	A
Hits	0	193642	94	995	104	47	23	3	29	46	34

Tabela 5 – Parallel single-node

Lab.	-	N	0	1	2	3	4
Cla.							
A	12282	433797	147	952	0	0	1
N	3088	203019	40	99	27	5	0
Ass.	-	N	A	A	N	N	A
Hits	0	203019	147	952	27	5	1

Tabela 6 – Parallel multi-node

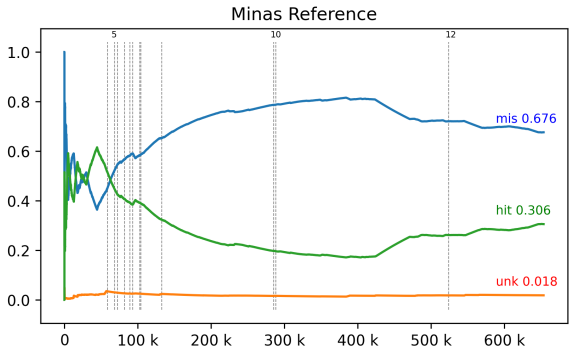
Lab.	-	N	0	1	2	3	4
Cla.							
A	12378	433631	117	886	0	162	5
N	3121	202916	40	96	105	0	0
Ass.	-	N	A	A	N	A	A
Hits	0	202916	117	886	105	162	5

For the measurements summary table, six measurements from two sources are displayed. Three measures *Hits*, *Unknowns* and *Misses* represented as ratio of the captured output stream, extracted from the evaluation python program, computed as follows: *Hits* (true positive rate) is the sum of the *Hits* row in the extended confusion matrix; *Unknowns* is the count of examples in the captured output stream marked with the *unknown* label (*"-"*); *Misses* is the count of all examples in the captured output stream marked with a label distinct from the *Assigned* original class and are not marked as unknown.

Furthermore in the measurement summary table, *Time*, *System* and *Elapsed* represented in seconds, are extracted from *GNU Time 1.9*. *Time* is the amount of CPU seconds expended in user-mode (indicates time used doing CPU intensive computing, e.g., math); *System* is the amount of CPU seconds expended in kernel-mode (for our case, it indicates time doing input or output); *Elapsed* is the real-world (wall clock) elapsed time and indicates how long the program took to complete. The lower the times, the better. Our four main experiments are shown in Tab. **??**.

Lastly, the stream visualization chart shows the summary quality measurement (*Hits*, *Unknowns*, *Misses*) computed for each example in the captured output stream. This summary is computed for each example, but it uses the *Assigned* row computed previously to evaluate *Hits*; the other measurements are derived as described before. The Horizontal axis (x, domain) plots the index of the example and the vertical axis (y, image) shows the measurement computed until that example index on the captured output stream.

Adding to the stream visualization chart, novelty label markers are represented as vertical lines indicating *when* in the captured output stream a new label first appeared. Some of the novelty label markers include the label itself ($l \in \mathbb{N}$) for reference (showing every label would turn this feature unreadable due to overlapping). Figure 5 shows complete stream visualization charts.



Reference Implementation

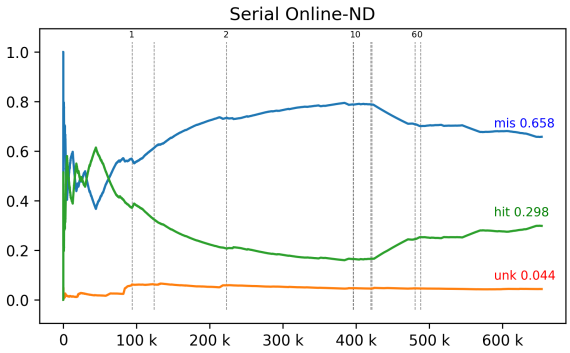


Figura 2 – Serial Implementation

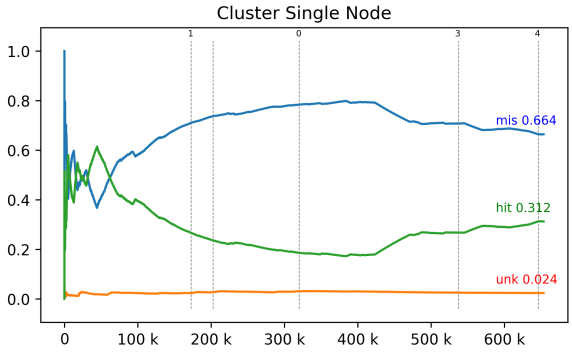


Figura 3 – Parallel single-node

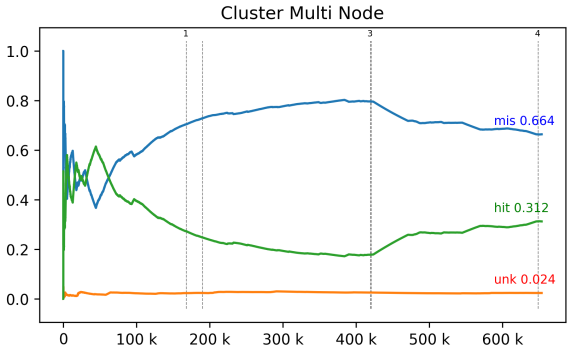


Figura 4 – Parallel multi-node

Figura 5 – Stream hits and novelties visualization

6.3 Conclusão

Portanto ...

Capítulo 7

Conclusão

Este trabalho reúne conceitos de aprendizado de máquina com ênfase em detecção de novidades em fluxos contínuos de dados e conceitos de processamento distribuído de fluxos contínuos, com o objetivo de unir a lacuna no estado da arte desses conceitos à luz de uma implementação e avaliação no cenário de detecção de intrusão em redes de dispositivos da Internet das Coisas (IoT) em ambiente de computação em névoa (*fog computing*).

O objeto central desse trabalho (sistema M-FOG) trata da implementação do algoritmo MINAS na plataforma de processamento de fluxos *Apache Flink*, em três módulos que podem ser distribuídos em um ambiente de *fog computing*. Sua distribuição permite selecionar o nó que tem os recursos computacionais mais adequados para cada tarefa. A avaliação do sistema M-FOG será feita por meio de métricas de qualidade de classificação e métricas de escalabilidade.

Dando continuidade a este trabalho, segue-se com o desenvolvimento da implementação objeto (sistema M-FOG) bem como a contínua avaliação comparativa dos resultados produzidos pelo sistema M-FOG com seu algoritmo base, MINAS. Também será dada continuidade nos experimentos com os conjuntos de dados (*data sets*) diversos e configurações variadas de distribuição de processamento em *fog computing* extraindo desses experimentos as métricas previamente discutidas.

Dessa forma, o sistema M-FOG pode contribuir com adição de uma ferramenta para os interessados em sistemas de detecção de intrusão de redes de dispositivos IoT ou outros sistemas que tratam de fluxos contínuos que tradicionalmente sofrem com os ônus de latência e largura de banda na comunicação entre borda e nuvem. Além disso, o sistema M-FOG objetiva contribuir com a adição de uma implementação distribuída de um algoritmo cujo modelo é estado da arte em detecção de novidades em fluxos contínuos de dados.

Data Stream Novelty Detection (ND-DS) can be a useful mechanism for Network

Intrusion Detection (NIDS) in IoT environments. It can also serve other related applications of ND-DS using continuous network or system behavior monitoring and analysis. Regarding the tremendous amount of data that must be processed in the flow analysis for ND-DS, it is relevant that this processing takes place at the edge of the network. However, one relevant shortcoming of the IoT, in this case, is the reduced processing capacity of such edge devices.

In this sense, we have put together and evaluated a distributed architecture for performing ND-DS in network flows at the edge. Our proposal, sistema M-FOG is a distributed ND-DS implementation based on the ND-DS algorithm algoritmo MINAS.

The main goal of this work is to observe the effects of our approach to a previously serial only algorithm, especially in regards to time and quality metrics.

While there is some impact on the predictive metrics, this is not reflected on overall classification quality metrics indicating that distribution of algoritmo MINAS shows a negligible loss of accuracy. In regards to time and scale, our distributed executions was faster than the previous sequential implementation of algoritmo MINAS, but efficient data distribution was not achieved as the observed time with each added node remained constant.

Overall, sistema M-FOG and the idea of using distributed flow classification and novelty detection while minimizing memory usage to fit in smaller devices at the edge of the network is a viable and promising solution. Further work include the investigation of other ND-DS algorithms, other clustering algorithms in algoritmo MINAS and analysis of varying load balancing strategies.

Referências

ABANE, A. et al. Modeling and improving named data networking over ieee 802.15.4. In: **2019 8th International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)**. [S.l.: s.n.], 2019. p. 1–6.

ABDALLAH, A.; MAAROF, M. A.; ZAINAL, A. Fraud detection system: A survey. **Journal of Network and Computer Applications**, v. 68, p. 90 – 113, 2016. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804516300571>.

ABDALLAH, Z. S. et al. Anynovel: detection of novel concepts in evolving data streams: An application for activity recognition. **Evolving Systems**, v. 7, n. 2, p. 73–93, 2016. ISSN 18686486.

AGGARWAL, C. C. et al. A framework for clustering evolving data streams. **Proceedings - 29th International Conference on Very Large Data Bases,VLDB 2003**, p. 81–92, 2003.

Apache Flink. **Apache Flink**. 2020. Disponível em: <https://flink.apache.org/>.

Apache Hadoop. **The Apache™ Hadoop® project develops open-source software for reliable,scalable,distributed computing.** 2020. Disponível em: <https://hadoop.apache.org/>.

Apache Spark. **Apache Spark™ - Unified Analytics Engine for Big Data**. 2020. Disponível em: <https://spark.apache.org/>.

BONOMI, F. et al. Fog computing and its role in the internet of things. In: **Proceedings of the first edition of the MCC workshop on Mobile cloud computing**. [s.n.], 2012. p. 13–16. ISBN 9781450315197. Disponível em: <http://www.lispmob.org>.

BUCZAK, A. L.; GUVEN, E. A survey of data mining and machine learning methods for cyber security intrusion detection. **IEEE Communications Surveys & Tutorials**, IEEE, v. 18, n. 2, p. 1153–1176, 2016.

CARBONE, P. et al. **Apache Flink™: Stream and Batch Processing in a Single Engine**. [S.l.], 2015. v. 36, n. 4. Disponível em: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-198940>.

CASSALES, G. W. et al. Idsa-iot: An intrusion detection system architecture for iot networks. In: **2019 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2019. p. 1–7. ISBN 978-1-7281-2999-0. ISSN 1530-1346. Disponível em: <https://ieeexplore.ieee.org/document/8969609/>.

COSTA, J. D. **Detecção De Novidade Em Fluxos Contínuos De Dados Multirrótulo**. 127 p. Tese (Master) — UFSCar - Universidade Federal de São Carlos, 2019. Disponível em: <https://repositorio.ufscar.br/handle/ufscar/12197>.

COSTA, K. A. da et al. Internet of things: A survey on machine learning-based intrusion detection approaches. **Computer Networks**, v. 151, p. 147–157, 2019. ISSN 13891286.

COULL, S. et al. Intrusion detection: A bioinformatics approach. In: IEEE. **19th Annual Computer Security Applications Conference,2003. Proceedings.** [S.l.], 2003. p. 24–33.

DA SILVA, T. P. et al. A fuzzy multiclass novelty detector for data streams. **IEEE International Conference on Fuzzy Systems**, IEEE, v. 2018-July, p. 1–8, 2018. ISSN 10987584.

DASTJERDI, A. V.; BUYYA, R. Fog computing: Helping the internet of things realize its potential. **Computer**, IEEE, v. 49, n. 8, p. 112–116, Aug 2016. ISSN 1558-0814.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified data processing on large clusters. **OSDI 2004 - 6th Symposium on Operating Systems Design and Implementation**, p. 137–149, 2004. ISSN 23487852.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: **Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York,NY,USA: Association for Computing Machinery, 2000. (KDD '00), p. 71–80. ISBN 1581132336. Disponível em: <https://doi.org/10.1145/347090.347107>.

FARIA, E. R. et al. Novelty detection in data streams. **Artificial Intelligence Review**, v. 45, n. 2, p. 235–269, Feb 2016. ISSN 1573-7462. Disponível em: <https://doi.org/10.1007/s10462-015-9444-8>.

_____. Evaluation methodology for multiclass novelty detection algorithms. In: **2013 Brazilian Conference on Intelligent Systems**. [S.l.: s.n.], 2013. p. 19–25. ISBN 9780769550923.

FARIA, E. R. de; CARVALHO, A. C. Ponce de L. F.; GAMA, J. Minas: multiclass learning algorithm for novelty detection in data streams. **Data Mining and Knowledge Discovery**, v. 30, n. 3, p. 640–680, May 2016. ISSN 1573-756X. Disponível em: <https://doi.org/10.1007/s10618-015-0433-y>.

FARIA, E. R. de et al. Evaluation of multiclass novelty detection algorithms for data streams. **IEEE Transactions on Knowledge and Data Engineering**, v. 27, n. 11, p. 2961–2973, nov 2015. ISSN 1041-4347. Disponível em: <http://ieeexplore.ieee.org/document/7118190/>.

FONTUGNE, R. et al. Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In: **ACM CoNEXT '10**. Philadelphia,PA: [s.n.], 2010. p. 1–12.

FOUNDATION, A. S. **Apache Storm**. 2020. Disponível em: <https://storm.apache.org/>.

GABER, M. M.; ZASLAVSKY, A.; KRISHNASWAMY, S. Mining data streams: A review. **SIGMOD Rec.**, Association for Computing Machinery, New York,NY,USA, v. 34, n. 2, p. 18–26, jun 2005. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/1083784.1083789>.

GAMA, J.; RODRIGUES, P. P. Data stream processing. In: _____. **Learning from Data Streams: Processing Techniques in Sensor Networks**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 25–39. ISBN 978-3-540-73679-0. Disponível em: <https://doi.org/10.1007/3-540-73679-4_3>.

_____. **Knowledge Discovery from Data Streams**. [S.l.]: Chapman and Hall/CRC, 2010. ISBN 9781439826119.

HADDADPAJOUH, H. et al. A survey on internet of things security: Requirements, challenges, and solutions. **Internet of Things**, Elsevier, p. 100129, 2019.

HAYAT, M. Z.; HASHEMI, M. R. A dct based approach for detecting novelty and concept drift in data streams. In: IEEE. **2010 International Conference of Soft Computing and Pattern Recognition,SoCPaR 2010**. [S.l.], 2010. p. 373–378. ISBN 9781424478958.

IEEE Communications Society. **IEEE Std 1934-2018: IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing.** IEEE, 2018. 176 p. ISBN 9781504450171. Disponível em: <https://ieeexplore.ieee.org/document/8423800>.

JÚNIOR, J. D. C. et al. Novelty detection for multi-label stream classification. In: IEEE. **2019 8th Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.], 2019. p. 144–149. ISBN 9781728142531.

KAMBOURAKIS, G.; KOLIAS, C.; STAVROU, A. The Mirai botnet and the IoT Zombie Armies. In: **MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)**. IEEE, 2017. v. 2017-Octob, p. 267–272. ISBN 978-1-5386-0595-0. Disponível em: <http://ieeexplore.ieee.org/document/8170867/>.

KOLIAS, C. et al. DDoS in the IoT: Mirai and Other Botnets. **Computer**, IEEE, v. 50, n. 7, p. 80–84, 2017. ISSN 00189162. Disponível em: <http://ieeexplore.ieee.org/document/7971869/>.

KREPS, J. **Questioning the Lambda Architecture – O'Reilly**. 2014. 10 p. Disponível em: <https://www.oreilly.com/radar/questioning-the-lambda-architecture/>.

LOPEZ, M. A.; DUARTE, O. C. M. B.; PUJOLLE, G. A monitoring and threat detection system using stream processing as a virtual function for big data. In: **Anais Estendidos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. Porto Alegre, RS, Brasil: SBC, 2019. p. 209–216. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc_estendido/article/view/7789>.

LOPEZ, M. E. A. **A monitoring and threat detection system using stream processing as a virtual function for Big Data**. Tese (Theses) — Sorbonne Université ; Universidade federal do Rio de Janeiro, Jun 2018. Disponível em: <https://tel.archives-ouvertes.fr/tel-02111017>.

MARKOU, M.; SINGH, S. Novelty detection: A review - Part 1: Statistical approaches. **Signal Processing**, Elsevier, v. 83, n. 12, p. 2481–2497, dec 2003. ISSN 01651684.

MARZ, N.; WARREN, J. **Big Data: Principles and best practices of scalable real-time data systems**. [S.l.]: New York; Manning Publications Co., 2015.

MASUD, M. et al. Classification and novel class detection in concept-drifting data streams under time constraints. **IEEE Trans. on Knowledge and Data Engineering**, IEEE, v. 23, n. 6, p. 859–874, June 2011. ISSN 1041-4347.

MAWI Working Group Traffic Archive. **Index of /mawi/samplepoint-F**. 2020. Disponível em: <http://mawi.wide.ad.jp/mawi/samplepoint-F/>.

MELL, P.; GRANCE, T. The NIST definition of cloud computing: Recommendations of the National Institute of Standards and Technology. In: NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Public Cloud Computing: Security and Privacy Guidelines**. National Institute of Standards and Technology, 2012. p. 97–101. ISBN 9781620819821. Disponível em: <http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>.

MITCHELL, R.; CHEN, I.-R. A survey of intrusion detection techniques for cyber-physical systems. **ACM Computing Surveys (CSUR)**, ACM, v. 46, n. 4, p. 55, 2014.

PERNER, P. Concepts for novelty detection and handling based on a case-based reasoning process scheme. **Engineering Applications of Artificial Intelligence**, v. 22, n. 1, p. 86–91, 2009. ISSN 0952-1976. Disponível em: <https://www.sciencedirect.com/science/article/pii/S095219760800105X>.

PROTIĆ, D. D. Review of KDD Cup '99,NSL-KDD and Kyoto 2006+ datasets. **Vojnotehnicki glasnik**, v. 66, n. 3, p. 580–596, 2018. ISSN 0042-8469. Disponível em: <http://orcid.org/0000-0003-0827-2863>.

SENGUPTA, J.; RUJ, S.; BIT, S. D. A comprehensive survey on attacks, security issues and blockchain solutions for iot and iiot. **Journal of Network and Computer Applications**, Elsevier, v. 149, p. 102481, 2020.

SHANBHAG, R.; SHANKARMANI, R. Architecture for internet of things to minimize human intervention. **2015 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2015**, IEEE, p. 2348–2353, 2015.

SHARAFALDIN, I.; LASHKARI, A. H.; GHORBANI, A. A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: INSTICC. **Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,**. SciTePress, 2018. p. 108–116. ISBN 978-989-758-282-0. Disponível em: <https://www.doi.org/10.5220/0006639801080116>.

SHI, W. et al. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, Institute of Electrical and Electronics Engineers Inc., v. 3, n. 5, p. 637–646, oct 2016. ISSN 23274662. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7488250>.

SILVA, T. P. da. **Abordagem Fuzzy para Detecção de Novidade em Fluxo Contínuo de Dados**. 89 p. Tese (Master) — Universidade Federal de São Carlos, 2018. Disponível em: <https://repositorio.ufscar.br/handle/ufscar/10544>.

SINGH, S.; MARKOU, M. An approach to novelty detection applied to the classification of image regions. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 16, n. 4, p. 396–407, 2004.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. **Inf. Process. Manage.**, Pergamon Press,Inc., USA, v. 45, n. 4, p. 427–437, jul. 2009. ISSN 0306-4573. Disponível em: <https://doi.org/10.1016/j.ipm.2009.03.002>.

SONG, J.; TAKAKURA, H.; OKABE, Y. **Kyoto 2006+ New version data**. 2020. Disponível em: <http://www.takakura.com/Kyoto_data/new_data201704/>.

SONG, J. et al. Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. **Proceedings of the 1st Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2011**, p. 29–36, 2011.

_____. Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. In: **Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security**. New York,NY,USA: Association for Computing Machinery, 2011. (BADGERS '11), p. 29–36. ISBN 9781450307680. Disponível em: <https://doi.org/10.1145/1978672.1978676>.

SPINOSA, E. J.; CARVALHO, A. P. de Leon F. de; GAMA, J. a. Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks. In: **Proceedings of the 2008 ACM Symposium on Applied Computing**. New York,NY,USA: Association for Computing Machinery, 2008. (SAC '08), p. 976–980. ISBN 9781595937537. Disponível em: <https://doi.org/10.1145/1363686.1363912>.

_____. Novelty detection with application to data streams. **Intell. Data Anal.**, IOS Press, NLD, v. 13, n. 3, p. 405–422, ago. 2009. ISSN 1088-467X.

TAHSIEN, S. M.; KARIMIPOUR, H.; SPACHOS, P. Machine learning based solutions for security of internet of things (iot): A survey. **Journal of Network and Computer Applications**, v. 161, n. November 2019, 2020. ISSN 10958592.

TAVALLAEE, M. et al. A detailed analysis of the kdd cup 99 data set. In: IEEE. **2009 IEEE symposium on computational intelligence for security and defense applications**. 2009. p. 1–6. ISBN 9781424437641. Disponível em: <https://doi.org/10.1109/CISDA.2009.5356528>.

VALLIM, R. M. et al. Online behavior change detection in computer games. **Expert Systems with Applications**, v. 40, n. 16, p. 6258 – 6265, 2013. ISSN

0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417413003576>.

VIEGAS, E. et al. Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. **Future Generation Computer Systems**, Elsevier, v. 93, p. 473 – 485, 2019. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X18307635>.

WANG, H. et al. Mining concept-drifting data streams using ensemble classifiers. In: **Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York,NY,USA: Association for Computing Machinery, 2003. (KDD '03), p. 226–235. ISBN 1581137370. Disponível em: <https://doi.org/10.1145/956750.956778>.

ZAHARIA, M. et al. Apache spark: A unified engine for big data processing. **Communications of the ACM**, v. 59, p. 56–65, 11 2016.

_____. Apache spark: A unified engine for big data processing. **Communications of the ACM**, v. 59, n. 11, p. 56–65, 2016. ISSN 15577317.

ZHANG, J. et al. Novel fault class detection based on novelty detection methods. In: _____. **Intelligent Computing in Signal Processing and Pattern Recognition: International Conference on Intelligent Computing,ICIC 2006 Kunming,China,August 16–19,2006**. Berlin,Heidelberg: Springer Berlin Heidelberg, 2006. p. 982–987. ISBN 978-3-540-37258-5. Disponível em: <https://doi.org/10.1007/978-3-540-37258-5_124>.

ZHOU, J. et al. Security and privacy for cloud-based iot: Challenges. **IEEE Communications Magazine**, v. 55, n. 1, p. 26–33, 2017.