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 constantes e evolutivas as ameaças ativas e passivas aos recursos computacionais e aos conteúdos transmitidos. Métodos para mineração de dados de forma robusta e contínua podem ser um aliado à segurança nesses casos. Particularmente em ambientes distribuídos e nos quais busca-se manter os fluxos de informação próximos a onde eles são gerados, como nas bordas das redes IoT, e na computação em névoa de maneira geral, a detecção de ameaças é essencial e não trivial. Além disso, a evolução constante dos tipos de dispositivos e de tráfegos nessas redes favorece que as ferramentas de detecção de ameaças sejam beneficiadas por algoritmos de detecção de novidades. MINAS é um exemplo de algoritmo de detecção de novidades em fluxos de dados com potencial para aplicação na computação em névoa. 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 maneira distribuída num contexto de detecção de intrusão e computação em névoa. Esta proposta foi implementada e é avaliada neste trabalho. Os resultados obtidos mostram a viabilidade do modelo de detecção de novidades distribuído em ambiente de computação em névoa. Mesmo sem uma análise de tráfego centralizada, a detecção de novidades apresentou eficácia equivalente à do algoritmo original, ainda que executando em dispositivos de capacidade computacional não elevada, como é previsto em cenários de IoT.

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

Figura 1 – Modelo de processos do sistema M-FOG. 25
Figura 2 – Diagrama UML de Sequência do sistema M-FOG: visão geral das linhas de vida. 46
Figura 3 – Parâmetros do algoritmo MINAS. 52
Figura 4 – Serial Implementation 55
Figura 5 – Parallel single-node 55
Figura 6 – Parallel multi-node 55
Figura 7 – Stream hits and novelties visualization 55

Lista de tabelas

Tabela 1 – Sumário dos trabalhos relacionados 32
Tabela 2 – Reference implementation 53
Tabela 3 – Serial implementation 53
Tabela 4 – Parallel single-node 53
Tabela 5 – Parallel multi-node 53
Tabela 6 – Sumário das métricas. 54

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*)

SPMD Único Programa, Múltiplos Dados (*Single Program, Multiple Data*)

Lista de Algoritmos

1 Fase *online* interpretada do algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016). 30
2 Detecção de Novidades interpretada do algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016). 31

3 Sistema M-FOG: ponto de entrada. 44
4 Função `Classificador` do nó folha do sistema M-FOG. 44
5 Função `AtualizaModelo` do nó folha do sistema M-FOG. 45
6 Função `Fonte` do nó raiz do sistema M-FOG. 45
7 Função `Detector` do nó raiz do sistema M-FOG. 46

Sumário

1 INTRODUÇÃO 13
1.1 Motivação 15
1.2 Objetivos 16
1.3 Proposta Metodológica 17
1.4 Organização do trabalho 17

2 FUNDAMENTOS CIENTÍFICOS E TECNOLÓGICOS . . . 19
2.1 Ambientes de Computação Distribuída 19
2.2 Arquiteturas e Plataformas de Processamento de Fluxos de Dados 22
2.3 Interface de Troca de Mensagens 24
2.4 Fluxo de Dados e Mineração de Fluxo de Dados 26
2.4.1 Detecção de Novidade em Fluxo de Dados 26
2.4.2 O algoritmo MINAS 27

3 TRABALHOS RELACIONADOS 32
3.1 Ferramenta BigFlow 32
3.2 Ferramenta CATRACA 34
3.3 Arquitetura IDSA-IoT 35
3.4 Conclusão 37

4 PROPOSTA E METODOLOGIA 38
4.1 Políticas 40
4.2 Implementações Preliminares 41
4.2.1 Implementação com *Python* e *Apache Kafka* 41
4.2.2 Implementação com *Apache Flink* 42
4.3 Implementação com MPI 42

4.4 Metodologia de Avaliação 47

5 EXPERIMENTOS E RESULTADOS 51
5.1 Ambiente de Teste 51
5.2 Métricas e Visualizações 53
5.3 Conclusão 54

6 CONCLUSÃO 56

REFERÊNCIAS 58

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). Além dos dados de sensores e atuadores que esses dispositivos gerenciam, se esses dispositivos forem subvertidos podem gerar tráfego maligno, como o produzido 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, o que torna complexa 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 energia, processamento, memória e comunicação), técnicas de segurança tradicionais baseadas em algoritmos configuráveis não são usuais, restando as técnicas de obser-

vação de rede (ZHOU et al., 2017). Ferramentas como Sistema de Detecção de Intrusão 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) têm sido empregadas na detecção de ataques a partir de características de ataques conhecidos ou pela tentativa de descoberta de 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 a métodos tradicionais de ML. Estes métodos comumente utilizam modelos estáticos para descrever e prever o comportamento da rede que não mantêm a confiabilidade frente à 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 elevadas as técnicas tradicionais que tratam grandes lotes em *datacenters* não são aplicáveis. Para esses fluxos contínuos de dados (*Data Stream*), técnicas de mineração de fluxos de dados (*Data Stream Mining*) são promissoras. 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 têm sido implementadas sobre o paradigma de computação na nuvem (*Cloud Computing*) e, recentemente, também sobre paradigmas como computação em névoa (*Fog Computing*). Para *fog*, além dos recursos em *cloud*, são explorados os recursos distribuídos pela rede desde nós remotos até a *cloud*. Processos que dependem desses recursos são distribuídos de acordo com características como sensibilidade à latência, privacidade, consumo computacional ou energético.

De maneira geral, 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 explora as características de implementação destas técnicas em conjunto, concentrando-se em serviços localizados na borda da rede, de maneira distribuída, para uso em ambientes

IoT.

FALTA na INTRODUCAO 1 OU 2 PARAGRAFOS EXPLICANDO O QUE VC FEZ/ E QUAL É UM RESUMO DOS RESULTADOS, SO PRA SETAR EXPECTATIVA DO LEITO

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 em ambientes distribuídos, como em cenários IoT e *fog computing*. 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. 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 deste trabalho, aqui chamada sistema M-FOG, adapta a arquitetura IDSA-IoT (CASSALES et al., 2019) empregando o algoritmo de ND-DS 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 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.

Ler aqui –> sera que motivacao tá clara, vc tá implementando isso pq o MINAS é um algoritmo promissor, IoT e subversao de dispositivos pode ser perigoso, e tem

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 em névoa. 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 foi o estabelecimento do estado da arte desses assuntos, de forma que alguns desses trabalhos servissem para comparações e relacionamentos. Além disso, desses trabalhos buscou-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 foi focada em plataformas, ferramentas e técnicas para realizar a implementação proposta. Portanto, foram selecionadas plataformas de processamento distribuído de DS e técnicas de aprendizado de máquina associadas a elas. Dessa revisão também foram identificadas 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 foi a experimentação. Nesse passo, foi desenvolvida uma aplicação na plataforma escolhida que, com base no algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016), classifica e detecta novidades em DS. Também nesse passo, a implementação foi validada comparando os resultados de classificação obtidos com os resultados de classificação do algoritmo original MINAS. Posteriormente, foram 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 foram organizados para publicação em 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 **??** os desafios

e resultados preliminares encontrados durante o desenvolvimento do trabalho. Capítulo 5 aborda os ambientes de teste, descreve experimentos realizados e discute os resultados obtidos. Capítulo 6 adiciona considerações gerais sobre o trabalho e seus resultados.

finalizar organizacao do trabalho versao final

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: computação em nuvem, computação de borda, e computação em névoa. A computação em nuvem (*cloud computing*) é aplicada a vários problemas, e para sistemas IoT fornece vastos recursos como computação e armazenamento para onde geralmente os 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 utilizam 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. 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. Entretanto, 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 Arquiteturas e Plataformas de Processamento de Fluxos de Dados

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. Entretanto, 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 o *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 (Apache Software 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 Interface de Troca de Mensagens

Em um sistema distribuído multiprocessado, a memória é distribuída entre múltiplos nós e processos, tendo cada processo acesso direto somente a sua memória local. Para esse tipo de sistema, o paradigma de programação paralela Único Programa, Múltiplos Dados (*Single Program, Multiple Data*) (SPMD) pode ser implementado utilizando o conceito de memória distribuída compartilhada através de troca de mensagens. Nesse modelo, cada processo tem sua memória local e se comunica com outros processos através da troca de mensagens (GROPP; LUSK; SKJELLUM, 2014). Observando boas práticas e melhores funcionalidades ao longo dos anos, a comunidade e a indústria padronizaram esse modelo de troca de mensagens no padrão Interface de Troca de Mensagens (*Message Passing Interface*) (MPI).

O 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 distribuída compartilhada. Esse modelo pode ser implementado desde computadores multiprocessados de memória compartilhada até supercomputadores com centenas de nós e milhares de processadores. O MPI, por meio de alguma implementação como OpenMPI e MPICH, permite a construção de um sistema distribuído com um executável único (monólito) e sua execução em um ambiente gerenciado.

O MPI tem duas primitivas básicas para comunicação dos processos que são o enviar (*send*) e o receber (*receive*) (GROPP; LUSK; SKJELLUM, 2014). A primitiva de envio é composta pelos argumentos: conteúdo da mensagem, tamanho da mensagem, processo destino, e *tag* é um argumento para diferenciar mensagens (ex: ordem e conteúdo). Associado à cada primitiva enviar, a primitiva receber possui os argumentos: conteúdo da mensagem, tamanho máximo da mensagem, remetente da mensagem, tamanho real da mensagem, e *tag*. Além de operações de envio e recebimento, pode-se fazer envio de mensagens de uma fonte para múltiplos destinos, e um destino receber de múltiplas fontes numa única operação.

Um dos detalhes importantes quando se utiliza o MPI com o TCP/IP, é que as garantias de envio são fornecidas pelo protocolo TCP, ou seja, feito um envio com a operação *send* o processo que enviou é desbloqueado logo após que a mensagem é escrita no buffer de saída do TCP do sistema operacional. Portanto, existe tempo de voo (*time of flight*) entre o envio e recebimento.

O MPI possui um ambiente de execução e gerenciamento chamado *mpirun* que recebe como entrada a configuração do cluster e o programa a ser executado, e se encarrega de

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

iniciar os processos em todos os nós, geralmente por *ssh*, repassando os parâmetros que recebeu a cada processo e o identificador do processo para que ele saiba qual dado ler e qual função executar.

Este trabalho utiliza o OpenMPI, que é uma implementação aberta de MPI, para a comunicação entre os processos de cada nó do cluster. Cada nó do cluster possui um conjunto de processos, e cada processo possui seu espaço de endereçamento e é *multithread*, onde as threads se comunicam através da memória compartilhada, como mostra a Figura 1. O OpenMPI permite a comunicação entre os processos independente se estão ou não no mesmo nó.

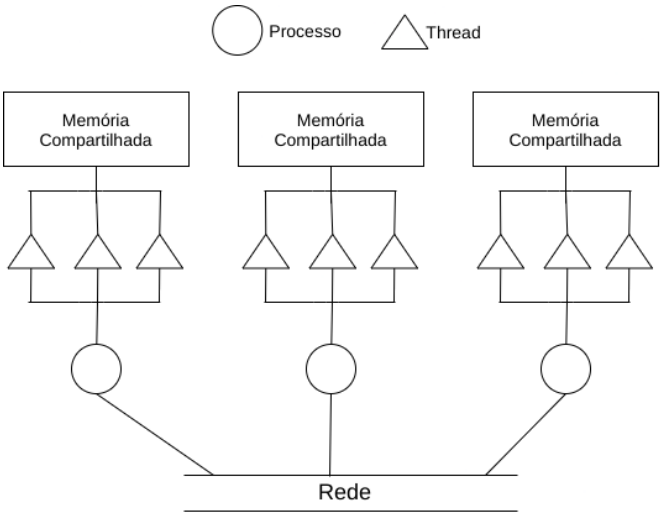


Figura 1 – Modelo de processos do sistema M-FOG.

O amplo uso de MPI para programação paralela e distribuída está principalmente pautado no desempenho. Processadores modernos possuem memórias hierárquicas que beneficiam programas que fazem o melhor uso de cache. O modelo de passagem de mensagens permite o particionamento dos dados em fatias menores otimizando o uso de cache nesses processadores. Portanto, para aplicações limitados por memória pode haver acelerações superlineares (*superlinear speedup*) (GROPP; LUSK; SKJELLUM, 2014).

Apesar do MPI não ter sido idealizado para o processamento de fluxo de dados, ele permite o controle sobre a distribuição de tarefas de maneira semelhante ao Apache Flink, porém dispensando o uso de um gerenciador. Essa dispensa de gerenciador em experimentos preliminares mostrou crucial para implementação do sistema M-FOG em dispositivos IoT, pois não excede o limite de memória do nó.

Em resumo, o MPI é utilizado neste trabalho dentro de um programa monólito escrito em C, e utilizando a biblioteca do MPI. Este monólito é executado por múltiplos processos dentro de cada nó. Cada processo recebe um conjunto de dados diferente. De acordo com o tipo de dado recebido, o processo executa funções diferentes. Se for um processo raiz, ele lê e distribui os dados de entrada entre os nós folha. Já os nós folha executam todo cálculo de distância e detecção de novidades, portanto execução de forma paralela e distribuída.

2.4 Fluxo de Dados e Mineração de 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). A mineração de *Big Data* trata de enormes conjuntos de dados que podem não ser processados em tempo viável devido a limitações como memória ou armazenamento principal. Uma abordagem para minerar esse tipo dado é a mineração de fluxo de dados. Os fluxos de dados são definidos na Definição 1.

Definição 1. *Um Fluxo de Dados S é uma sequência massiva, potencialmente ilimitada de exemplos $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n, \ldots$, recebida em instantes $T_1, T_2, \ldots, T_n, \ldots$, onde cada exemplo $\mathbf{X}$ tem dimensão d (AGGARWAL et al., 2003).*

O volume dos dados em *Big Data* e a velocidade com qual eles são produzidos afeta como os dados de um problema são modelados e manipulados. Técnicas e algoritmos de mineração de fluxos de dados atendem esses desafios utilizando restrições como apenas uma leitura do conjunto de dados e baixo complexidade na construção de seus algoritmos (GAMA; RODRIGUES, 2007; GABER; ZASLAVSKY; KRISHNASWAMY, 2005). O processamento requerido para a mineração de um enorme fluxos de dados pode não ser atendido por recursos computacionais de um único computador (nó), de forma que pode ser necessária a distribuição dos dados e do processamento em múltiplos nós computacionais em um sistema distribuído (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 2.2.

2.4.1 Detecção de Novidade em Fluxo de Dados

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.

As características de evolução dos padrões observados em fluxos de dados contínuos são: 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 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, podem existir 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; **??**).

2.4.2 O algoritmo MINAS

Um algoritmo de ND-DS que tem recebido atenção nos últimos anos é o algoritmo MINAS, originalmente proposto por **??**), 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 rotulados. Esse conjunto de treinamento é dividido em grupos usando como chave o rótulo, 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* (Definição 2). A cada *micro-cluster* é adicionada o rótulo do grupo original e todos *micro-clusters* são arranjados em um único conjunto formando o modelo de decisão.

Definição 2. *Para um grupo de exemplos* $\mathbf{X} \in \mathcal{MX}$*, um* micro-cluster $\mathcal{M}$ *é uma tupla de quatro componentes* $(n, \mathbf{LS}, \mathbf{SS}, t)$ *derivados dos exemplos representados por este* micro-cluster *onde (AGGARWAL et al., 2003; FARIA; CARVALHO; GAMA, 2016):*

- n *é a contagem de exemplos,*
- **LS** *é o vetor soma linear dos exemplos sendo* $\mathbf{LS}_d = \sum_{i=0}^{n} \mathbf{X}_{i,d}$*,*
- **SS** *é o vetor da soma dos quadrados dos exemplos sendo* $\mathbf{SS}_d = \sum_{i=0}^{n} (\mathbf{X}_{i,d})^2$*,*
- t *é o instante de chegada do último exemplo adicionado ao* micro-cluster.

Deste sumário extrai-se o centro e o raio que são utilizados na operação de classificação da fase *Online* (FARIA; CARVALHO; GAMA, 2016). O vetor centro é definido pelo algoritmo de agrupamento e corresponde ao ponto médio dos exemplos contidos no grupo, visto na Equação 1. O raio pode ser definido de duas maneiras e ambas dependem no conjunto das distâncias $D_{\mathcal{M}}$ entre centro e cada elemento contido no grupo. A primeira definição de raio é o desvio padrão das distâncias em $D_{\mathcal{M}}$ multiplicado por um fator parametrizado (FARIA; CARVALHO; GAMA, 2016), visto na Equação 2, a segunda definição é o valor máximo das distâncias em $D_{\mathcal{M}}$ (FARIA, 2013), visto na Equação 3.

Definição 3. *O vetor centro é o ponto médio dos exemplos contidos no grupo:*

$$centro = 1/n \cdot \mathbf{LS} \tag{1}$$

O conjunto das distâncias $D_{\mathcal{M}}$ *entre centro e cada elemento contido no grupo:*

$$D_{\mathcal{M}} = \{d_i : d_i = dist(\mathbf{X}_i, centro)\} \qquad \mathbf{X}_i \in \mathcal{MX}$$

O escalar raio é o desvio padrão ou o máximo do conjunto $D_{\mathcal{M}}$:

$$\mu = 1/n \cdot \sum_{i=1}^{n} d_i \qquad d_i \in D_{\mathcal{M}}$$

$$\sigma^2 = 1/n \cdot \sum_{i=1}^{n} (d_i - \mu)^2 \qquad d_i \in D_{\mathcal{M}}$$

$$\sigma = \sqrt{\sigma^2}$$

$$raio_1 = f_{raio} \cdot \sigma \tag{2}$$

$$raio_2 = max\{d_i \in D_{\mathcal{M}}\} \tag{3}$$

A distinção entre as duas definições de raio é importante, pois ela impacta imensamente nos resultados. Para cada *data set* ou problema abordado uma configuração de parâmetros afinada é essencial, pois as características da distribuição dos valores e sua associação às classes são geralmente únicas para cada problema. Em testes preliminares, a implementação do algoritmo MINAS com raio definido pela distância máxima (Eq. 3) mostrou facilidade na configuração quando comparado com a mesma implementação utilizando a definição que utiliza desvio padrão (Eq. 2), pois pequenas variações no parâmetro f_{raio} resultavam em alta taxa de desconhecidos ou alta taxa de erro.

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 o rótulo 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 no conjunto de desconhecidos (FARIA; CARVALHO; GAMA, 2016).

A segunda operação da fase *Online* é a detecção de padrões novidade listada no Algoritmo 2 que é executada quando o tamanho do conjunto de desconhecidos é maior que um parâmetro predefinido. Esta operação utiliza o agrupamento (*clustering* descrito na fase *Offline*) e valida os *micro-clusters* gerados verificando sua representatividade e coesão. A representatividade é avaliada com o número de exemplos do conjunto de desconhecidos que é englobado pelo novo *micro-cluster*, onde o valor mínimo de exemplos é parametrizado.

Um padrão é considerado coeso se o valor *silhueta*, Equação 4, é maior que 0. O valor *silhueta* por sua vez é calculado com os valores: b a distância do novo *micro-cluster* até o mais próximo contido no modelo atual; a o desvio padrão das distâncias entre o centro do novo *micro-cluster* e cada exemplo representado no novo *micro-cluster*.

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

Função `MinasOnline(`*Modelo, fluxoEntrada*`):`
  Desconhecidos ← ∅, ModeloAntigo ← ∅ ;
  últimaLimpeza ← 0 , proximaNovidade ← 0;
  para cada $exemplo_i \in$ *fluxoEntrada* **faça**
    maisPróximo ← `clusterMaisPróximo` (exemplo, Modelo);
    se *maisPróximo.distância* < *maisPróximo.cluster.raio* **então**
      exemplo.rótulo ← maisPróximo.cluster.rótulo;
      maisPróximo.cluster.últimoUso ← i ;
    senão
      exemplo.rótulo ← "desconhecido";
      Desconhecidos ← Desconhecidos ∪ exemplo;
      se | *Desconhecidos* | ≥ gatilhoDetecçãoNovidade **então**
        novidades ← `DetecçãoNovidade` (Modelo ∪ ModeloAntigo, *Desconhecidos);
        Modelo ← Modelo ∪ novidades;
      se $i >$ *(últimaLimpeza +* janelaLimpeza*)* **então**
        Modelo ← `moveModeloAntigo` (Modelo, *ModeloAntigo, últimaLimpeza);
        Desconhecidos ← `removeExemplosAntigos` (Desconhecidos, últimaLimpeza);
        últimaLimpeza ← i;
    fluxoSaída.adicione(exemplo);

Algoritmo 1: Fase *online* interpretada do algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016).

$$
\begin{aligned}
a = \sigma &= SD\{dist(\mathbf{X}_i, centro) : \mathbf{X}_i \in \mathcal{MX}\} \\
b &= min\{dist(centro_{novo}, centro(\mathcal{M}_i)) : \mathcal{M}_i \in Modelo\} \\
silhueta &= \frac{b-a}{max(b,a)}
\end{aligned} \tag{4}
$$

Para atribuição de rótulos 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 rótulo o valor de extensão de conceito conhecido. Caso contrário, se o novo *micro-cluster* está mais distante, um novo conceito foi encontrado e o rótulo marca um padrão novidade. Após a atribuição do rótulo do novo *micro-cluster*, ele é adicionado ao modelo de decisão.

Além do algoritmo MINAS, existem derivados que estendem a implementação original seguindo sua estrutura básica, duas notáveis são FuzzyND e MINAS-LC. O algoritmo FuzzyND foi proposto em Silva et al. (2018), Silva (2018) e incrementa o algoritmo original aplicando teoria de conjuntos *fuzzy* modificando a representação de *cluster*, o algoritmo de agrupamento, a manutenção do modelo para exemplos conhecidos e para novidades. O algoritmo MINAS-LC foi proposto por Costa (2019) e trata a classificação multi-rótulo,

Parâmetros: minExemplos, fatorNovidade
Função `DetecçãoNovidade(`*Modelo, Desconhecidos*`):`
novoModelo ← ∅;
para cada *novoCluster* ∈ `agrupamento` *(Desconhecidos)* **faça**
  se (| *novoCluster.exemplos* | ≥ minExemplos) ∧ (*novoCluster.silhueta* > 0) **então**
    maisPróximo ← `clusterMaisPróximo` (novoCluster, Modelo);
    se *maisPróximo.distância* < *(maisPróximo.cluster.raio* × fatorNovidade*)* **então**
      novoCluster.rótulo ← maisPróximo.cluster.rótulo;
      novoCluster.tipo ← "extensão";
    senão
      novoCluster.rótulo ← 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).

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 no algoritmo MINAS foram recorrentes durante as discussões no desenvolvimento deste trabalho porém, como estão fora do escopo deste trabalho, ficam somente aqui registradas para algum trabalho futuro. As modificações são:

a) Na fase *offline* a verificação de sobreposição de *clusters* pertencentes a classes distintas e tratamento adequado;

b) Diferentes métodos de cálculo de distância entre pontos além da distância euclidiana;

c) Mudança de representação de *clusters*, atualmente esferas no espaço de dimensão d, para paralelogramos no espaço de dimensão d tratando *data sets* onde as características representadas por cada dimensão são completamente independentes;

d) Em uma implementação de avaliação, ajuste automático de parâmetros otimizando para o máximo de classificações corretas e mínimo tempo de execução para um *data set* fornecido.

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 aplicados à detecção de intrusão em redes IoT. Um sumário dos trabalhos abordados pode ser visto na Tabela 1.

Tabela 1 – Sumário dos trabalhos relacionados

Trabalho	**Plataforma**	**Técnica ou Algoritmo**	**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. Como mecanismo de detecção de intrusão na construção de um NIDS, duas abordagens são comuns, detecção por assinatura e detecção por anomalia. Para a detecção

de novos tipos de ataque (*zero day*) a abordagem de detecção por anomalia é vantajosa comparada a detecção por assinatura que tem tempo de resposta 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, extrair as características desses pacotes formando descritores de fluxo 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 eles 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 dos traços de pacotes no *backbone* WIDE, ponto de amostras F (*Packet traces from WIDE backbone, samplepoint-F*), composto por seções diárias de captura de pacotes 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. Por fim, 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 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. Em todos os casos a métrica observada foi a acurácia. 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 a dependência da avaliação de um especialista não são ideais para detecção de ameaças 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*).

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

[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/>.

mensagens (*Apache Kafka*) e produziam para outro tópico. O primeiro programa consiste de um detector de ameaças 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.

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

[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.

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 de névoa (*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: ECSMiner (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 proposta de distribuição das tarefas em serviços abre oportunidades para a discussão

de diferentes métodos de distribuição dessas tarefas em diferentes ambientes computacionais. Contudo, o algoritmo MINAS que apresentou resultados promissores para este tipo de cenário ainda não foi implementado e avaliado com paralelismo e distribuição que são necessários para tratar fluxos de dados com grandes volumes de dados 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

Neste trabalho investiga-se uma arquitetura e implementação adequada para executar Detecção de Novidades em Fluxos de Dados (*Novelty Detection in Data Stream*) (ND-DS) em ambiente de névoa para detecção de intrusão em redes de dispositivos Internet das Coisas (*Internet of Things*) (IoT). Esta abordagem é baseada na arquitetura IDSA-IoT (CASSALES et al., 2019) e no algoritmo de ND-DS MINAS (FARIA; CARVALHO; GAMA, 2016). Nomeada sistema M-FOG, esta implementação do algoritmo explora computação distribuída habilitando o uso de computadores de baixa potência e recursos limitados presentes no ambiente de névoa para, além de tarefas habituais, a classificação e detecção de tráfego indesejável.

Sistema de Detecção de Intrusão em Redes (*Network Intrusion Detection System*) (NIDS) monitoram tráfego em redes e analisam as características de cada fluxo de rede com objetivo de identificar intrusos e tráfegos indesejados. No entanto, este problema requer respostas rápidas e acuradas (COSTA et al., 2019): Rapidez é necessária para executar uma ação adequada antes que maior dano seja feito à rede, e para lidar com a velocidade do tráfego sem impor atraso ou perda de pacotes na rede observada; Acurácia é necessária para não identificar padrões incorretamente, gerando falso alarme ou ignorando ataques.

Em cenários IoT comuns, dados são capturados por pequenos dispositivos computacionais e enviados para a nuvem (*cloud computing*) se mais mais recursos computacionais ou de armazenamento são necessários; contudo, para NIDS e o requisito de resposta rápida esta abordagem pode não ser viável. A infraestrutura de Computação em Névoa (*fog computing*) oferece capacidades de realocar parte do processamento e armazenamento dos provedores de nuvem, posicionando dispositivos de capacidade intermediária próximos aos usuários e às fontes de dados.

Dada a natureza distribuída e o típico uso de pequenos computadores em cenários de IoT e névoa, alguns desafios são notáveis:

(*i*) O tráfego nas bordas da rede é inerentemente distribuído, sem uma entidade centralizadora que tenha acesso a todas as transmissões;

(*ii*) A tarefa de classificação do algoritmo deve ocorrer em paralelo em diferentes nós;

(*iii*) A tarefa de detecção de novidade, que provê a atualização do modelo, deve ser assíncrona;

(*iv*) A complexidade do algoritmo (tempo e espaço) de permitir sua execução em computadores modestos, isto é, pouca memória ($< 1GB$) e processadores de pouco desempenho.

Nesta proposta, recursos em névoa são combinados para minimizar a latência entre o recebimento (ingestão) do descritor de fluxo e a identificação daquele fluxo, executado múltiplas instâncias da tarefa de classificação do algoritmo MINAS. Após identificado o rótulo do descritor de fluxo (exemplo), este pode ser utilizado imediatamente para tomada de decisões e, se o exemplo é rotulado como "desconhecido", este é armazenado para posterior execução assíncrona da tarefa de detecção de novidade, portanto sem interromper o processo de identificação.

Além do multiprocessamento da tarefa de classificação e da execução assíncrona da tarefa de detecção de novidade na fase *online*, outra mudança em relação ao MINAS original é a remoção do mecanismo de esquecimento de padrões antigos. Esta remoção tem por finalidade simplificar a atualização e o compartilhamento do modelo entre as múltiplas instâncias e *threads* concorrentes. O impacto desta alteração não é grande pois o algoritmo MINAS não prevê remoção permanente dos *micro-clusters* porém existe um aumento na complexidade computacional já que o número de *micro-clusters* analisados durante a busca do mais próximo para classificação inclui os padrões menos de uso menos frequente.

A fase *offline* e a função de detecção de novidades do algoritmo MINAS permanecem as mesmas.

A arquitetura geral do sistema M-FOG é dividida em dois módulos, Classificação e Detecção de Novidade, que equivalem à fase *online* do algoritmo MINAS. O módulo Classificação executa a tarefa de mesmo nome do algoritmo MINAS e é ponto focal dos esforços de paralelismo desta proposta. Este módulo é replicado em todos os nós do cluster em névoa.

O módulo Detecção de Novidade tem uma instância por cluster, equivalendo a uma instância por rede local portanto, enquanto o compartilhamento de modelo entre redes distintas é possível, ele não foi implementado e testado. Este módulo é responsável pela tarefa homônima do algoritmo MINAS, recebendo exemplos com rótulo "desconhecido"

e os armazenando no conjunto de desconhecidos; quando o tamanho deste conjunto alcança um valor parametrizado, este módulo executa a função de detecção de novidade do algoritmo MINAS.

4.1 Políticas

O projeto desta arquitetura de ND-DS distribuída inclui o particionamento das funcionalidades de MINAS e o estabelecimento dos fluxos de dados apropriados entre os diferentes atores. Mudanças no posicionamento de cada ator e no comportamento tem impactos diferentes no desempenho e na acurácia do modelo e devem ser escolhidas com cuidado. Alguns dos aspectos com impactos potenciais na implementação incluem:

- Com relação à alocação do módulo Detecção de Novidade e sincronização de modelo entre redes distintas para compartilhamento de padrões de novidade local com outras redes que não receberam aquele padrão de ataque:
  - Em névoa (há um módulo em cada cluster alocado à uma rede local): padrões serão somente detectados se um número suficiente de exemplos deste padrão ocorrer na rede observada; considerando o uso do poder computacional da névoa, qualquer sincronização de modelo deve ter a propriedade de aditividade; também é preciso reconhecer padrões duplicados;
  - Em nuvem: capacidade de detecção de padrões dispersos em cada rede local onde se analisado não formaria um cluster válido, neste caso todos exemplos com rótulo "desconhecido" são enviados da rede local para a nuvem implicando aumento do uso do *link* de internet e aumento do atraso entre aparição de um padrão, sua detecção e a propagação para os classificadores em névoa;
  - Em ambos (nuvem e névoa): cada conjunto de desconhecidos em cada rede local é mantido bem como a detecção de novidade para padrões locais, quando um exemplo é considerado ruído e removido do conjunto de desconhecidos o exemplo é enviado à nuvem para detecção de novidade global. Esta opção requer um mecanismo complexo de sincronização de modelo.
- Com relação ao mecanismo de esquecimento: Mesmo quando detecção de novidade global é usada, modelos locais podem ser otimizados para classificação rápida usando estatísticas locais para ordenar (e remover, usando menos memória) clusters menos utilizados;
- Por último, na reclassificação do conjunto de desconhecidos, feita pela tarefa de detecção de novidade do algoritmo MINAS, o subconjunto de desconhecidos que pertencem a um novo cluster válido é, efetivamente, classificado com o rótulo deste

novo cluster. No Algoritmo 2, linha 12, o novo cluster válido inclui o subconjunto de exemplos que o compõem; portanto, se este conjunto de exemplos fosse classificado com o rótulo atribuído ao novo cluster e adicionado ao fluxo de saída, esta saída teria exemplos atrasados e duplicados em relação ao fluxo de entrada e o resultado obtido poderia ser mais acurado. Além disso, esta escolha modificaria o comportamento do operador de fluxo de dados de um *map*, onde para cada exemplo do fluxo de entrada existe um exemplo no fluxo de saída, para *flatMap*, onde cada exemplo entrada pode ter mais de um exemplo no fluxo de saída.

4.2 Implementações Preliminares

No desenvolvimento desta pesquisa, uma vez determinado o modelo de operação distribuída, com processamento nas bordas da rede, algumas experimentações e algumas ferramentas de teste foram desenvolvidas. Aspectos desses desenvolvimentos são descritos a seguir.

4.2.1 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.

4.2.2 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 da fase de treinamento *offline* do algoritmo MINAS, já que o conjunto consumido por esse módulo é limitado.

Após desenvolvimento e testes em um computador pessoal, o sistema foi testado no ambiente de testes como descrito na Seção 5.1, onde observou-se uma redução enorme no desempenho que concluiu-se ser causada pelo uso excessivo de memória pela plataforma *Apache Flink*. Com a configuração dos parâmetros de memória do cluster, resultados compatíveis com o esperado foram obtidos; no entanto, apesar de exemplos de sucesso na literatura (LEE et al., 2017; GRECO; RITROVATO; XHAFA, 2019; BATTULGA; MIORANDI; TEDESCHI, 2020), o ambiente de testes não permaneceu estável para execução de repetições do experimento, necessitando reinicializações para que o controlador não ocupasse mais de $1GB$ na segunda execução, o que degradava imensamente o desempenho.

Em conclusão, apesar de promissora, a plataforma *Apache Flink* ainda não suportava a execução em dispositivos computacionais restritos de maneira confiável, sendo a principal barreira o uso excessivo de memória, comum em plataformas do gênero *BigData*.

4.3 Implementação com MPI

Com os desafios de distribuição e memória encontrados nas implementações preliminares, optou-se por tomar mais controle sobre estes aspectos para maximizar o aproveitamento do *hardware* escolhido, visto que seus recursos são limitados. Para isso escolheu-se implementar a proposta utilizando Interface de Troca de Mensagens (*Message Passing Interface*) (MPI) e linguagem C, onde tem-se controle fino sobre o uso de memória e distribuição. No entanto, perde-se a facilidades e garantias incluída nas plataformas tradicionais que, por exemplo, oferecem o funcionamento confiável e contínuo com mecânicas de recuperação de erros.

O algoritmo MINAS original (FARIA; CARVALHO; GAMA, 2016) tem uma implementação referência[1] (FARIA, 2013), aqui referida como *Ref*, escrita em Java utilizando os algoritmos base como *K-means* e *CluStream* da biblioteca MOA (BIFET et al., 2010). Esta implementação serve de referência para a construção do modelo distribuído previsto, provendo validação dos resultados e servindo também de comparação básica de desempenho.

[1] Disponível em <http://www.facom.ufu.br/~elaine/MINAS>

Os primeiros pontos de divergência do sistema M-FOG e *Ref* são o algoritmo de agrupamento e cálculo de raio. Enquanto *Ref* permite a escolha entre *K-means* e *CluStream* para a fase *offline* e *online*, sistema M-FOG implementa apenas *K-means*. O cálculo de raio em *Ref* é definido com o máximo do conjunto de distância dos exemplos de um *micro-cluster* ao seu centro, seguindo a Equação 3, enquanto o sistema M-FOG segue a definição em Faria, Carvalho e Gama (2016) utilizando o desvio padrão dos valores do mesmo conjunto de distâncias multiplicado pelo parâmetro f_{raio} seguindo a Equação 2.

Os formatos dos fluxos de dados de entrada e de saída também são notáveis. Como entrada, o algoritmo da proposta recebe dois fluxos, o principal é o que contém os exemplos, sendo cada exemplo um vetor de números de dimensão d. O segundo fluxo de entrada consiste de *micro-clusters* representando o modelo inicial criado e capturado da fase de treinamento *offline*. A fase de treinamento *offline* foi implementada e não sofreu alteração com relação ao algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016), além da execução separada com conjunto de treinamento similar à definição do fluxo de exemplos principal com adição de uma dimensão com valor de um caractere, que marca a classe conhecida e saída como um fluxo finito de *micro-clusters*.

O formato do fluxo de saída é definido como a tripla contendo o número de sequência do exemplo no fluxo de entrada (*uid*), a etiqueta de um caractere atribuída ao exemplo, e o tempo em milissegundos entre a ingestão (entrada) e saída do exemplo no sistema.

Para avaliação desta proposta, ela foi construída com a implementação *Open MPI 4.0.4*[2] do padrão MPI, seguindo o paradigma de programação paralela Único Programa, Múltiplos Dados (*Single Program, Multiple Data*) (SPMD). No paradigma SPMD, uma única versão do programa é inicializada em todos os nós, sendo que para cada instância são passados os parâmetros `mpiSize` e `mpiRank`, que representam o número de nós e o índice de cada nó no cluster. Neste caso, o parâmetro `mpiRank` é o componente de múltiplos dados em SPMD e, como cada nó recebe com um valor diferente, cada nó pode ter um comportamento diferente. Neste quesito, o Algoritmo 3, linha 3 mostra exatamente esse comportamento no ponto de entrada do sistema M-FOG, dividido o comportamento de cada nó de acordo com seu `mpiRank` em dois tipos: raiz e folha.

O Algoritmo 3 mostra os primeiros passos do sistema sistema M-FOG, a linha 2 cria Modelo inicial como um conjunto vazio, e a trava (*lock*) utilizada para controle de acesso concorrente das *threads* `Detector` e `Fonte` no nó raiz e `AtualizaModelo` e `Classificador` nos nós folha.

No processo raiz, de `mpiRank` $= 0$ e com acesso aos fluxos de entrada e de saída, uma *thread* com a função `Detector` é iniciada e a função `Fonte` chamada. Nos processos folha, de `mpiRank` > 0, uma *thread* com a função `AtualizaModelo` é iniciada e a função `Classificador` é chamada pela *thread* associada à função principal.

Destaca-se que a possibilidade de, ao invés de um `Classificador` por processo e múlti-

[2] Documentação em <https://www.open-mpi.org/doc/v4.0/>

```
Parâmetros: mpiRank, mpiSize
Entrada   : fluxoEntrada
Saída     : fluxoSaída
1 Função Mfog(fluxoEntrada, fluxoSaída):
2 |  Modelo ← ∅; trava ← new Trava ();
3 |  se mpiRank == 0 então raiz
4 |  |  new Thread (Detector, [fluxoSaída, Modelo, trava]);
5 |  |  Fonte (fluxoEntrada, Modelo, trava);
6 |  senão folha
7 |  |  new Thread (AtualizaModelo, [Modelo, trava]);
8 |  |  Classificador (Modelo, trava);
```

Algoritmo 3: Sistema M-FOG: ponto de entrada.

plos processos por nó, pode-se utilizar um processo por nó e múltiplas *threads* executando `Classificador` por nó. Esta estratégia economiza a transmissão de *micro-clusters* para cada nó folha na função `AtualizaModelo` porém aumenta a concorrência de acesso ao modelo entre as *threads* executando `Classificador`. Em avaliações simples comparando estas estratégias nenhum resultado conclusivo foi observado e há espaço em trabalhos futuros para avaliações mais profundas destas estratégias e outras de estratégias de otimização.

A função `Classificador`, listada no Algoritmo 4, é o centro do sistema e opera com o modelo atualizado e um exemplo recebido como mensagem do nó raiz. Esta função calcula as distâncias entre o exemplo e todos *micro-clusters* do modelo e encontrando o mais próximo e verificando se o modelo explica aquele exemplo; se explica, o rótulo do *micro-cluster* mais próximo é atribuído como rótulo do exemplo. A função `Classificador`, como as outras, só chega ao seu final se a mensagem recebida (ou instância lida no caso da função `Fonte`) é um marcador de fim de fluxo (`FimDeFluxo`, *end of stream, eos*).

```
1 Função Classificador(Modelo, trava):
2 |  enquanto Verdade faça
3 |  |  exemplo ← recebe (TipoExemplo, raiz);
4 |  |  se exemplo == FimDeFluxo então pare;
5 |  |  exemplo.rótulo ← "desconhecido";
6 |  |  com travaLeitura (trava)
7 |  |  |  (distância, cluster) ← clusterMaisPróximo (exemplo, Modelo);
8 |  |  se distância < cluster.raio então
9 |  |  |  exemplo.rótulo ← cluster.rótulo;
10 |  |  envia (raiz, TipoExemplo, exemplo);
```

Algoritmo 4: Função `Classificador` do nó folha do sistema M-FOG.

A função `AtualizaModelo`, listada no Algoritmo 5, recebe novos *micro-clusters* como mensagens do nó raiz, independente deste *micro-cluster* ser do modelo inicial, novidade ou extensão, adequadamente travando o acesso ao modelo compartilhado com a *thread* execu-

tando a função `Classificador`. Além da trava de acesso à leitura e escrita na implementação, a função `Classificador` espera por um sinal emitido pela função `AtualizaModelo`, indicando que o modelo está completo antes de começar a busca pelo *micro-clusters* mais próximo.

```
1  Função AtualizaModelo(Modelo, trava):
2  |  enquanto Verdade faça
3  |  |  cluster ← recebe (TipoCluster, raiz);
4  |  |  se cluster == FimDeFluxo então pare;
5  |  |  com travaEscrita (trava)
6  |  |  |  Modelo ← Modelo ∪ cluster;
```

Algoritmo 5: Função `AtualizaModelo` do nó folha do sistema M-FOG.

A função `Fonte`, listada no Algoritmo 6, lê uma instância do fluxo de entrada que pode ser do tipo exemplo ou do tipo cluster. Se a instância for um exemplo, ele é enviado para um dos nós folha, o nó é escolhido via balanceamento de carga *round-robin*. Se a instância for do tipo cluster, o modelo compartilhado com a *thread* executando a função `Detector` é travado para leitura e escrita e atualizado e o novo cluster é enviado para todas os nós folhas.

```
1  Função Fonte(fluxoEntrada, Modelo, trava):
2  |  dest ← 1;
3  |  para cada exemplo_i ∈ fluxoEntrada faça
4  |  |  se tipoDe (exemplo) é TipoCluster então
5  |  |  |  broadcast (TipoCluster, exemplo, raiz);
6  |  |  |  com travaEscrita (trava)
7  |  |  |  |  Modelo ← Modelo ∪ exemplo;
8  |  |  |  continue;
9  |  |  envia (dest, TipoExemplo, exemplo);
10 |  |  dest ← dest +1;
11 |  |  se dest > mpiSize então dest ← 1;
```

Algoritmo 6: Função `Fonte` do nó raiz do sistema M-FOG.

Enquanto a função `Fonte` gerencia a entrada de dados, a função `Detector`, listada no Algoritmo 7, gerencia a saída de dados e, como tem acesso aos exemplos já classificados, também gerencia o conjunto de desconhecidos. Se o tamanho do conjunto de desconhecidos atinge um valor mínimo, a função de detecção de novidade é chamada, os *micro-clusters* representando padrões novidades ou extensões são adicionados ao modelo e enviados para todos os nós folha. Além da remoção dos exemplos utilizados para formar *micro-clusters* de novidades e extensões, exemplos que participaram duas vezes do processo de detecção de novidade são considerados *outliers* ou ruído e são removidos.

As interações entre os diferentes módulos e ciclo de vida do sistema é ilustrado no diagrama de sequencia da Figura 2.

```
1  Função Detector(fluxoSaída, Modelo, trava):
2  |  Desconhecidos ← ∅; últimaLimpeza ← 0;
3  |  enquanto Verdade faça
4  |  |  exemplo ← recebe (TipoExemplo, qualquer);
5  |  |  se exemplo == FimDeFluxo então pare;
6  |  |  fluxoSaída.adicione(exemplo);
7  |  |  se exemplo.label == unknown então
8  |  |  |  Desconhecidos ← Desconhecidos ∪ exemplo;
9  |  |  |  se | Desconhecidos | ≥ gatilhoDetecçãoNovidade então
10 |  |  |  |  novidades ← DetecçãoNovidade (Modelo, *Desconhecidos);
11 |  |  |  |  com travaEscrita (trava)
12 |  |  |  |  |  Modelo ← Modelo ∪ novidades;
13 |  |  |  |  para cada cluster ∈ novidades faça
14 |  |  |  |  |  broadcast (TipoCluster, cluster, raiz);
15 |  |  |  se exemplo.uid > ( últimaLimpeza + janelaLimpeza) então
16 |  |  |  |  Desconhecidos ← removeExemplosAntigos (Desconhecidos, últimaLimpeza);
17 |  |  |  |  últimaLimpeza ← exemplo.uid;
```

Algoritmo 7: Função `Detector` do nó raiz do sistema M-FOG.

Em suma, a arquitetura do sistema M-FOG é composta de múltiplos nós numa névoa, que detectam intrusão numa rede IoT. Para isso, os nós da rede processam de forma paralela e distribuída o cálculo das distâncias e detecção de novidades, sendo assim um sistema multiprocessado e distribuído. Utiliza-se um único programa com vários processos por nó, e cada processo recebe um conjunto de dados diferentes seguindo o paradigma

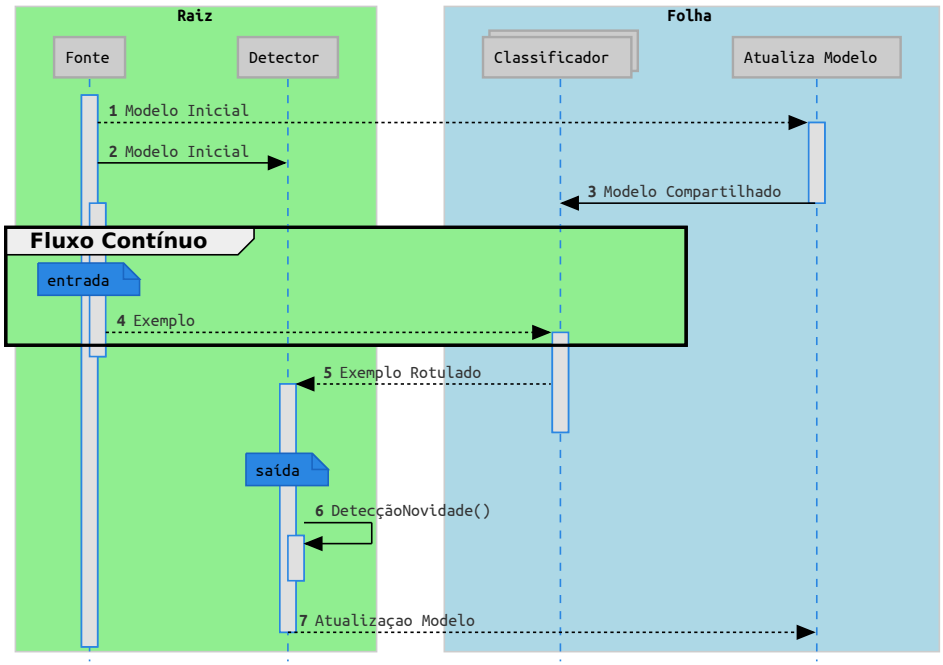


Figura 2 – Diagrama UML de Sequência do sistema M-FOG: visão geral das linhas de vida.

SPMD.

4.4 Metodologia de Avaliação

A avaliação da proposta aqui apresentada é feita por meio de métricas da literatura, divididas em duas partes: métricas de qualidade de classificação e métricas de escalabilidade.

As métricas de qualidade de classificação escolhidas devem ser adequadas para avaliar detecção de novidades em *data streams*, felizmente o tratamento necessário é estabelecido por Faria et al. (2013) e expandido por 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 capturado no final do fluxo, e sim para cada exemplo $\mathbf{X}$ classificado. Portanto, as métricas têm como índice o contador de exemplos x, informando a posição do exemplo em relação ao fluxo, ou seja, as métricas são de todos os exemplos até a instância x.

Definição 4. *O problema de Detecção de Novidades em fluxo de dados pode ser modelado como uma função f onde, dado o um exemplo $\mathbf{X}$ de um fluxo de dados S, pertencente a uma classe real $c \in \mathbf{C}$, é associado à um rótulo $l \in \mathbf{L}$ podendo ser uma classe conhecida $c' \in \mathbf{C}'$, "desconhecido" ou um rótulo novidade $y \in \mathbf{Y}$:*

$$f : \mathbf{X} \mapsto l \quad \mathbf{X} \in S, \quad l \in \mathbf{L} \tag{5}$$

$$\mathbf{C} = \{c_1, c_2, \cdots, c_m\} \tag{6}$$

$$\mathbf{Y} = \{y_1, y_2, \cdots, y_k\} \tag{7}$$

$$\mathbf{L} = \{l_1, l_2, \cdots, l_n\} = \mathbf{C}' \cup \{\textit{"desconhecido"}\} \cup \mathbf{Y} \tag{8}$$

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 $\mathbf{E}_x$ na Equação 9, adaptada para detecção de novidade.

Definição 5. *Uma matriz de confusão $\mathbf{E}$ para a instância x, de dimensão $m \times n$ para m classes e n rótulos, é preenchida com o número de instâncias da classe c_i classificados com o rótulo l_j.*

$$\mathbf{E}_x = (e_{ij}) \in \mathbb{N}^{m \times n} = \begin{pmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,n} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m,1} & e_{m,2} & \cdots & e_{m,n} \end{pmatrix} \tag{9}$$

Note que a soma de todos os elementos da matriz de confusão é igual a x, ou seja, a contagem de todos exemplos.

Um dos possíveis rótulos $l_j \in \mathbf{L}$ é o rótulo "desconhecido", que indica um erro de classificação diferente dos demais e é tratado separadamente. A taxa de desconhecidos *UnkR* na Equação 11 é uma das métricas escolhidas pois trata deste tipo específico de erro (FARIA et al., 2013).

Definição 6. *A taxa de desconhecidos $UnkR_x$ para um instante x é definida como a média da taxa de desconhecidos de cada classe (FARIA et al., 2013). A taxa de desconhecidos de cada classe c_i é definida como o elemento $e_{ij} \in \mathbf{E}_x$ onde j é a coluna do rótulo "desconhecido" na matriz de rótulos $\mathbf{L}$ dividido pelo número de exemplos da classe.*

$$UnkR_x = \frac{1}{m}\sum_{i=1}^{m} UnkR_{x,i} \tag{10}$$

$$UnkR_{x,i} = \frac{e_{ij} : l_j = \text{"desconhecido"}}{\sum_{j=1}^{n} e_{ij}} \tag{11}$$

Para todos os rótulos l_j diferentes de "desconhecido", uma classe c_i é associada se o rótulo l_j é procedente do modelo inicial, ou seja $l_j = c_i : c_i \in \mathbf{C}$ ou, em último caso, o rótulo l_j é associado à classe com o maior número de exemplos com o rótulo l_j, ou seja $e_{ij} = max\{e_{aj} : a \in [0, m]\}$ para $c_i \in \mathbf{C}$ e $l_j \in \mathbf{L}$ (FARIA et al., 2013).

Definição 7. *A função de associações $A(l)$ é definida como:*

$$A(l_j) : l_j \in \mathbf{L} \mapsto c_i \in \mathbf{C} = \begin{cases} \textit{indefinido} & \textit{se } l_j = \text{"desconhecido"} \\ c_i & \textit{se } \exists c_i = l_j \quad : c_i \in \mathbf{C}' \\ c_i & \textit{se } e_{ij} = max\{e_{aj} : a \in [0, m]\} \end{cases} \tag{12}$$

No contexto de classificação multi-classe, a acurácia acc_x para um instante x é definida como a média da acurácia de cada classe, de maneira semelhante a taxa de desconhecidos. A métrica acurácia, de forma semelhante à definição da matriz de confusão, é definida como extensão da acurácia da classificação binária. Para classificação binária a acurácia calculada com os valores verdadeiro-positivo (tp), verdadeiro-negativo (tn), falso-positivo (fp) e falso-negativo (fn). Para cada classe c_i, o valor verdadeiro-positivo (tp_i) é definido como número de exemplos onde o rótulo do exemplo l é associado à classe c_i, o valor falso-negativo (fn_i) são os exemplos da classe onde o rótulo do exemplo não é associado à classe c_i e o rótulo não é "desconhecido". Os valores verdadeiro-negativo (tn) e falso-positivo (fp) são zero.

$$tp_i = \sum_{j=1}^{n} e_{ij} \quad \text{se } l_j \neq \text{“desconhecido” e } A(l_j) = c_i \tag{13}$$

$$fn_i = \sum_{j=1}^{n} e_{ij} \quad \text{se } l_j \neq \text{“desconhecido” e } A(l_j) \neq c_i \tag{14}$$

$$acc_i = \frac{tp + tn}{tp + fn + fp + tn} = \frac{tp_i}{fn_i + tp_i} \tag{15}$$

$$acc_x = \frac{1}{m} \sum_{i=1}^{m} acc_i \tag{16}$$

Concluíndo as métricas de qualidade de classificação, a métrica de erro combinado (err) é a média do erro de cada classe, sendo o erro de cada classe o valor falso-negativo (fn_i) dividido pelo número de exemplos da classe.

$$err = \frac{1}{m} \sum_{i=1}^{m} \frac{fn_i}{fn_i + tp_i} \tag{17}$$

Para a validação da corretude da implementação do sistema M-FOG com relação ao algoritmo MINAS original, ambos são executados com o mesmo *data set* e as métricas de qualidade de classificação são comparadas. As métricas de escalabilidade extraídas são o número de processadores, tempo de processamento e latência de eventos e, estas métrias permitem o cálculo de *speedup* prático.

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.

Um programa de avaliação foi construído seguindo as técnicas de referência como matriz de confusão multi-classe com associação de classe de rótulo (FARIA; CARVALHO; GAMA, 2016) para extrair medidas de qualidade de classificação. Este script recebe duas entradas, o conjunto de dados de teste e o fluxo de saída capturado, e gera a matriz de confusão, associação de classe de rótulo, resumo de qualidade final com: *Hits* (acurácia), *Misses* (Err), *Unknowns* (UnkR); and

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 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. 2.

Capítulo 5

Experimentos e Resultados

5.1 Ambiente de Teste

Com o objetivo de avaliar esta proposta e averiguar os efeitos da distribuição da detecção de novidades em um cenário IoT, construiu-se um ambiente experimental de névoa (*fog computing*).

Este ambiente é composto por três computadores de única placa (*Single Board Computer*) modelo Raspberry Pi 3 model B, equipados com o SoC (*system-on-chip* sistema em um chip) de arquitetura ARM *BCM2837* com 4 núcleos de processamento à frequência de $1.2GHz$, $32kB$ e $512kB$ de memória *cache* nível 1 e 2 respectivamente, $1GB$ de memória, armazenamento em cartão SD e conectados por rede cabeada *Ethernet*.

A ideia central é criar um cluster simples simulando *gateway* de uma rede IoT com recursos limitados. Este cluster armazenou todo o código-fonte, binários (compilados no mesmo cluster) e *data set*. Nesta configuração, o *data set* é armazenado no cartão SD do nó raiz e é lido para cada execução do experimento. Todos os experimentos foram executados neste cluster para isolamento de outras variações imprevistas e para garantir que as comparações seriam justas com *software* e *hardware* constante.

O *data set Kyoto 2006+*[1], tráfego das *Honeypots* da Universidade de Kyoto, é o foco deste trabalho. Este *data set* 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., 2011).

O segmento utilizado é o de Dezembro de 2015, contendo 7 865 245 instâncias. Deste segmento, são filtradas apenas instâncias associadas a tráfego normal ou ataques conhe-

[1] Disponível em <http://www.takakura.com/Kyoto_data/>

cidos, identificados por NIDS que tenham mais de 10 000 instâncias para significância, como feito previamente por Cassales et al. (2019). As instâncias mantidas são normalizadas para que o valor de cada característica original (como endereço IP, duração do fluxo, serviço) seja transposta para o intervalo Real $[0, 1]$.

O conjunto resultante da operação recém descrita é então dividido em dois conjuntos, treinamento e teste. Para avaliar a detecção de ataques, o conjunto treinamento é composto apenas de de trafego normal, contendo 72000 instâncias. O conjunto de teste possui as 653 457 instâncias, sendo elas 206 278 instâncias com classe "N" (normal) e 447 179 instâncias com classe "A" (ataque).

Destaca-se que esta escolha pode resultar em *Overfitting* para a classe normal e *underfitting* para a classe ataque, pois o sistema primeiro precisa detectar um padrão novidade para então adicionar ao modelo.

Quanto às configurações do algoritmo MINAS, os valores dos parâmetros foram escolhidos por serem valores comuns para o algoritmo presentes na literatura (FARIA; GAMA; CARVALHO, 2013; FARIA; CARVALHO; GAMA, 2016) e na implementação de referência *Ref* (FARIA, 2013), ou escolhidos por aproximarem os resultados aos obtidos pela implementação de referência *Ref* com os mesmos parâmetros, até onde é possível, e *data set*.

Os parâmetros utilizados, ilustrados na Figura 3, são os da literatura: `k` número de *micro-clusters* gerados pelo algoritmo de agrupamento, `minExamplesPerCluster` número de mínimo de exemplos para um *micro-cluster* válido, `thresholdForgettingPast` limite para remoção de exemplos do conjunto de desconhecidos e, o restante escolhidos por aproximação, `precision` é o valor limite para melhora na distância global reduzindo as iterações no algoritmo de agrupamento (otimização) e, `radiusF` fator que multiplica o desvio padrão das distâncias entre o centro e cada exemplo formador do novo *micro-cluster* definindo o raio do *micro-cluster* e `noveltyF` fator que multiplica o desvio padrão das distâncias do *micro-cluster* mais próximo distinguindo um novo padrão entre extensão e novidade.

```
1    MinasParams minasParams = {
2      .k=100, .dim=22, .precision=1.0e-08,
3      .radiusF=0.25, .minExamplesPerCluster=20, .noveltyF=1.4,
4      .thresholdForgettingPast = 10000,
5    };
```

Figura 3 – Parâmetros do algoritmo MINAS.

Tabela 2 – 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 3 – 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

5.2 Métricas e Visualizações

Tabela 4 – 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 5 – 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

Tabela 6 – Sumário das métricas.

	Ref (a)	Offline	Serial (b)	Single Node (c)	Multi Node (d)
Hits	199708 0.305618		195017 0.298438	204151 0.312416	204191 0.312478
Misses	441769 0.676049		429873 0.657843	433936 0.664061	433767 0.663802
Unknowns	11980 0.018333		28567 0.043717	15370 0.023521	15499 0.023718
Time	2761.83	194.12	80.79000	522.1000	207.1400
System	7.15	0.075	11.51000	47.7700	157.6100
Elapsed	2772.07	194.27	93.03000	145.0400	95.3800

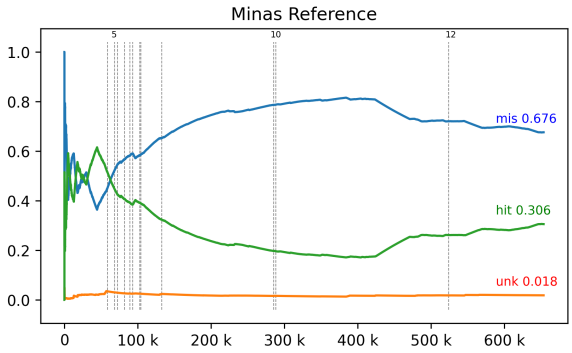
expended in user-mode (indicates time used doing CPU intensive computing, e.g., math); *System* is the amount of CPU seconds expended in kernel-mode (for our case, it indicates time doing input or output); *Elapsed* is the real-world (wall clock) elapsed time and indicates how long the program took to complete. The lower the times, the better. Our four main experiments are shown in Tab. 6.

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 7 shows complete stream visualization charts.

5.3 Conclusão

Portanto ... fix-me



Reference Implementation

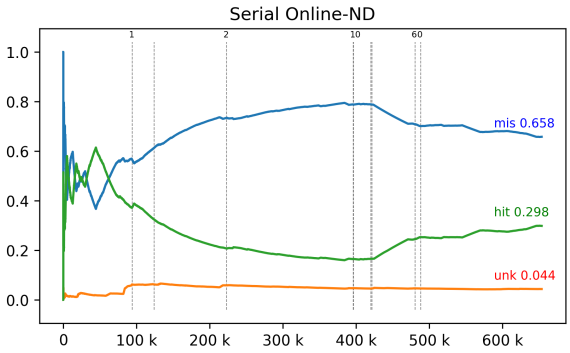


Figura 4 – Serial Implementation

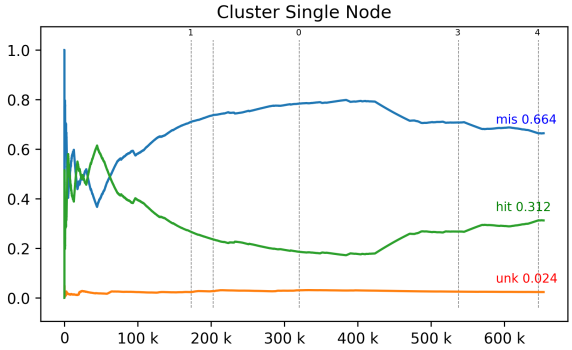


Figura 5 – Parallel single-node

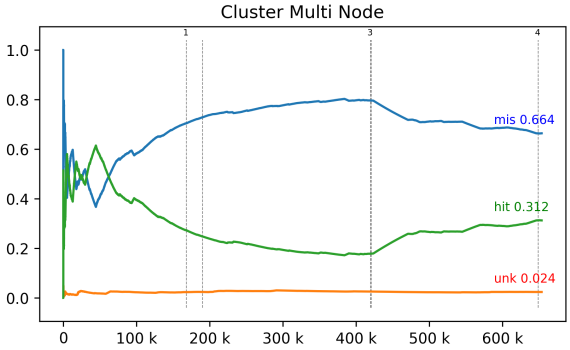


Figura 6 – Parallel multi-node

Figura 7 – Stream hits and novelties visualization

Capítulo 6

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 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 MINAS shows a negligible loss of accuracy. In regards to time and scale, our distributed executions was faster than the previous sequential implementation of 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 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 Software Foundation. **Apache Storm**. 2020. Disponível em: <https://storm.apache.org/>.

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

BATTULGA, D.; MIORANDI, D.; TEDESCHI, C. Fogguru: a fog computing platform based on apache flink. In: **23rd Conference on Innovation in Clouds, Internet and Networks (ICIN 2020)**. Paris, France: [s.n.], 2020. Disponível em: <https://hal.inria.fr/hal-02463206>.

BIFET, A. et al. MOA: massive online analysis. **J. Mach. Learn. Res.**, v. 11, p. 1601–1604, 2010. Disponível em: <http://portal.acm.org/citation.cfm?id=1859903>.

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.

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. **Source codes of MINAS: MultI-class learNing Algorithm for data Streams**. Universidade Federal de Uberlândia, Faculdade de Computação, Uberlândia, Minas Gerais, Brasil, 2013. Disponível em: <http://www.facom.ufu.br/~elaine/DadosMINAS/MINAS-SourceCode.rar>.

FARIA, E. R.; GAMA, J. a.; CARVALHO, A. C. P. L. F. Novelty detection algorithm for data streams multi-class problems. In: **Proceedings of the 28th Annual ACM Symposium on Applied Computing**. New York, NY, USA: Association for Computing Machinery, 2013. (SAC '13), p. 795–800. ISBN 9781450316569. Disponível em: <https://doi.org/10.1145/2480362.2480515>.

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.

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.

GRECO, L.; RITROVATO, P.; XHAFA, F. An edge-stream computing infrastructure for real-time analysis of wearable sensors data. **Future Generation Computer Systems**, v. 93, p. 515–528, 2019. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X18314031>.

GROPP, W.; LUSK, E.; SKJELLUM, A. **Using MPI: portable parallel programming with the message-passing interface**. London, UK: MIT Press, 2014. v. 3. 330 p. ISBN 978-0-262-52739-2.

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/>.

LEE, H. et al. A data streaming performance evaluation using resource constrained edge device. In: IEEE. **2017 International Conference on Information and Communication Technology Convergence (ICTC)**. [S.l.], 2017. p. 628–633. ISBN 9781509040315.

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>.

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.

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>.

SILVA, T. P. da 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.

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.

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. 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.

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.