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: Prof. Dr. Hermes Senger

São Carlos
2021

Resumo

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

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

Abstract

In a scenario of growing number of devices connected to the Internet of Things (IoT) with proportional growth in the volume of data streams generated, robust methods are needed for mining streams continuous data. One of the areas affected by the huge growth in the number of devices and the streams associated with them is the information security, which needs network intrusion detection tools that operate in fog computing environments due to the cost of operating such tools in a cloud only environment. These tools make extensive use of algorithms for novelty detection in data streams to identify treat patterns in network traffic. However, algorithms in wide use do not adequately address the challenges of novelty detection in data streams, such as concept drift, concept evolution and continuous update of the classification model, without expert interference. The MINAS algorithm addresses those novelty detection in data streams challenges and has received recent research attention. However, despite its division in three semi-independent parts, MINAS has not yet been adapted to process large volumes of real streams or to operate in a fog computing environment. The present work proposes a system that implements the MINAS algorithm in a distributed fog environment in the context of intrusion detection to addresses this gap. Preliminary work shows that it is possible to have a distributed version of the MINAS algorithm by using stream processing platforms such as Apache Flink.

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

Lista de ilustrações

Lista de tabelas

Lista de siglas

Sumário

Capítulo 1

Introdução

A Internet das Coisas (*Internet of Things* - IoT) é um sistema global de dispositivos (máquinas, objetos físicos ou virtuais, sensores, atuadores e pessoas) com capacidade de comunicação pela Internet, sem depender de interação com interface humano-computador tradicional. Outra característica de dispositivos IoT são os recursos computacionais dimensionados, para propósitos específicos que limitam a capacidade de computar outras funções além da função original do dispositivo. O número de dispositivos categorizados como IoT na última década teve crescimento sem precedentes e, proporcionalmente, cresceu o volume de dados gerados por esses dispositivos. A análise desses dados pode trazer novos conhecimentos e tem sido um tema frequentemente abordado por trabalhos de pesquisa. Contudo, além dos dados de sensores e atuadores, esses dispositivos se subvertidos, podem gerar tráfego maligno , como o gerado pela *botnet* mirai em 2016 (KAMBOURAKIS; KOLIAS; STAVROU, 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, além da falta das cruciais atualizações de segurança.

Com milhares de dispositivos em redes distantes gerando dados (diretamente ligados às suas funções originais ou metadados produzidos como subproduto) em volumes e velocidades consideráveis, formando fluxos contínuos de dados (*Data Stream* - DS), técnicas de mineração de fluxos de dados (*Data Stream Mining*) são amplamente necessárias. Nesses cenários, essas técnicas são aplicadas, por exemplo, em problemas de monitoramento e classificação de valores originários de sensores para tomada de decisão tanto em nível micro, como na modificação de atuadores remotos, ou macro, na otimização de processos industriais. Analogamente, as mesmas técnicas de classificação podem ser aplicadas para os metadados gerados pela comunicação entre esses nós e a Internet, detectando alterações nos padrões de comunicação num serviço de detecção de intrusão (*Network Intrusion*

Detection System, NIDS).

Técnicas de Mineração de Fluxo de Dados (*Data Stream Mining*) envolvem mineração de dados (*Data Mining*), aprendizado de máquina (*Machine Learning*) e, dentro destes tópicos, detecção de novidades (*Novelty Detection*, ND). 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 ou detecção de anomalias, detectar novos padrões. Essa capacidade é relevante em especial para o exemplo de detecção de intrusão, onde novidades na rede podem distinguir novas funcionalidades (entregues aos dispositivos após sua implantação em campo) de ataques por agentes externos, sem assinaturas existentes em bancos de dados de ataques conhecidos.

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

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, 2015), sendo que o último mostrou resultados promissores. A arquitetura proposta foi avaliada com o conjunto de dados (*data set*) *Kyoto 2006+*, composto de dados coletados de 348 *Honeypots* (máquinas isoladas equipadas com diversos softwares com vulnerabilidades conhecidas expostas à Internet com propósito de atrair ataques) de 2006 até dezembro 2015. 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 (CASSALES et al., 2019).

Contudo, o algoritmo MINAS ainda não foi implementado e avaliado com paralelismo, multi-processamento ou distribuição computacional, que são necessários para tratar fluxos de dados com grandes volumes e velocidades. O tratamento de distribuição em ambiente *fog computing* é essencial para aplicação deste algoritmo ao problema de detecção de intrusão em redes IoT, pois esta aplicação requer tempo de resposta mínimo e mínima

comunicação entre nós distantes, como aqueles na borda e na nuvem. Ainda observando o algoritmo MINAS, destaca-se a possível divisão em três partes semi-independentes, sendo elas treinamento, classificação e detecção de novidade; a classificação é o elemento central cujos resultados são utilizados para a identificação de intrusões.

Ainda no contexto de ND 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, atualmente impossível de ser processado em um único núcleo de processador (*single-threaded*). Essa implementação foi feita sobre uma plataforma distribuída processadora de fluxos (*Apache Flink*) executada em um cluster com até 10 nós de trabalho, cada um com 4 núcleos de processamento, totalizando 40 núcleos, para atingir taxas de até $10,72$ *Gbps*.

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

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 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. Seguindo a comparação entre algoritmos desse gênero realizada por Cassales et al. (2019), esta pesquisa escolheu investigar o algoritmo MINAS (FARIA; CARVALHO; GAMA, 2015) para receber o tratamento necessário para adequá-lo ao ambiente de névoa e para fluxos de grandes volumes e velocidades.

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 de maneira escalável e distribuível para ambientes de computação em névoa e 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;
- ❏ Avaliar a qualidade de detecção de intrusão em ambiente distribuído conforme a arquitetura IDSA-IoT;
- ❏ Avaliar o desempenho da implementação em ambiente de computação em névoa.

1.3 Proposta Metodológica

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

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

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

Uma vez definidos o estado da arte, as ferramentas técnicas e os *data sets*, o passo seguinte é a experimentação. Nesse passo, será desenvolvida uma aplicação na plataforma escolhida que, com base no algoritmo MINAS (FARIA; CARVALHO; GAMA, 2015), irá classificar e detectar novidades em DS. Também nesse passo, a implementação será validada comparando os resultados de classificação obtidos com os resultados de classificação do algoritmo original MINAS. Posteriormente, serão realizados experimentos com a implementação e variações em *data sets* e cenários de distribuição em *fog*, coletando as métricas de classificação e escalabilidade.

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

1.4 Organização do trabalho

O restante desse trabalho segue a estrutura: Capítulo 2 aborda conceitos teóricos e técnicos que embasam esse trabalho; Capítulo **??** enumera e discute trabalhos relacionados e estabelece o estado da arte do tema detecção de novidade em fluxos de dados e seu processamento; Capítulo **??** 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 3 adiciona considerações gerais e apresenta o plano de trabalho e cronograma até a defesa do mestrado.

Capítulo 2

Fundamentos Científicos e Tecnológicos

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

2.1 Ambientes de Computação Distribuída

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

2.1.1 Computação em Nuvem

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:** o cliente pode provisionar ou liberar capacidades de computação (ex: tempo de processamento e armazenamento) conforme o necessário, sem requerer interação com o provedor de serviço;
- **Amplo acesso à rede:** o acesso aos recursos de computação e capacidades ocorre pela rede através de mecanismos padrões que permitem o acesso por plataformas heterogêneas (celulares, computadores, tablets, etc.)
- **Agrupamento de recursos:** para servir múltiplos clientes, os recursos de computação são agrupados usando o modelo *multi-tenancy* com recursos físicos e virtuais diferentes dinamicamente atribuídos e reatribuídos de acordo com a demandas do clientes;
- **Elasticidade:** as capacidades de computação são rapidamente provisionadas ou liberadas, em alguns casos automaticamente, para escalar conforme a demanda;
- **Serviço mensurado:** os recursos de computação são monitorados, controlados e reportados para o provedor de serviços e para o cliente fornecendo transparência sobre as capacidades que foram consumidas.

Segundo, Mell e Grance (2012), a implantação da Computação em Nuvem pode ocorrer através dos seguintes modelos:

- **Nuvem privada:** a infraestrutura da nuvem é provisionada e dedicada para um único cliente ou organização. Nesse modelo, o cliente gerencia e controla a infraestrutura, ou pode delegar essas tarefas a uma outra empresa. A infraestrutura pode estar dentro ou fora das instalações da organização proprietária;
- **Nuvem comunitária:** a infraestrutura de nuvem é fornecida para um grupo exclusivo de clientes que compartilham um mesmo interesse (requerimentos de segurança, desempenho, políticas, etc.). Esse tipo de nuvem pode ser gerenciado pelo próprio grupo, ou por outra organização, podendo estar dentro ou fora das instalações das empresas proprietárias;
- **Nuvem pública:** É gerenciada e operada por um provedor de nuvem e a infraestrutura é provisionada e oferecida para uso público.
- **Nuvem híbrida:** a infraestrutura desse tipo de nuvem é uma composição de dois ou mais modelos de implantação de *cloud* (privada, pública e comunitária) que

formam uma entidade única e são unidos por tecnologias padronizadas que habilitam a portabilidade de dados e aplicações.

2.1.2 Computação de Borda

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

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

2.1.3 Computação em Névoa

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

(DASTJERDI; BUYYA, 2016). Portanto, para solucionar estas questões de latência e capacidade limitada dos dispositivos de borda, a computação em névoa foi proposta.

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

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

- **Mobilidade:** é essencial que as aplicações *fog* sejam capazes de se comunicar com dispositivos móveis, por exemplo, utilizando protocolos que considerem a mobilidade dos nós;
- **Heterogeneidade:** os nós nesse tipo de paradigma possuem configurações e formatos diferentes e podem estar implantados em ambientes distintos;
- **Baixa Latência:** 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 ;
- **Uso de fluxo de dados e aplicações em tempo real:** a computação em névoa pode envolver aplicações que processam em lote, mas na maior parte das vezes envolve aplicações com requisito de processamento em tempo real, e para isso fazem o uso de fluxo de dados. Por exemplo, os sensores de um rede IoT escrevem a informação no fluxo de dados, a informação é processada, ações são inferidas e traduzidos em ações nos componentes atuadores.

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

2.2 Mineração de Dados e Fluxo de Dados

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

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

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

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

Computação distribuída é a área da ciência da computação que estuda sistemas em que os componentes são localizados em diferentes computadores (nós), que comunicam-se apenas por troca de mensagens e, para que o objetivo do sistema seja atingido, a cooperação entre os nós é necessária. Outras propriedades de um sistema distribuído são a concorrência entre os nós e possibilidade de falhas em partes independentes (TANENBAUM; STEEN, 2018).

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

2.3 Arquiteturas e Plataformas de Processamento de Fluxos

Tradicionalmente, aplicações foram construídas com um sistema gerenciador de banco de dados (SGBD) relacional ou não-relacional associado. Essa arquitetura, nomeada de "arquitetura totalmente incremental" por Marz e Warren (2015), foi evoluída e simplificada iterativamente durante décadas de uso, porém ela não é adequada para sistemas em tempo real, como os sistema de fluxo de dados. O volume e a velocidade de dados em um *Data Stream* leva à necessidade de distribuir o processamento, acrescentando poder computacional a cada nó adicionado. Porém, desafios como comunicação eficiente e sincronização de estado entre os nós, assim como tolerância a falhas, aumentam a complexidade de construção de um sistema distribuído em relação a um sistema tradicional.

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

2.3.1 Arquitetura *Lambda*

A arquitetura de processamento distribuído de fluxos de grande volume de dados *Lambda* divide o processamento em três camadas: lotes, serviço e velocidade (MARZ; WARREN, 2015). A camada de lotes atua sobre o conjunto mestre em modo de leitura sequencial, armazenando-o em sistema de arquivos distribuído e pré-processando várias visões sobre esse conjunto mestre. Essas visões (armazenadas num SGBD tradicional) são consumidas pela camada de serviço, que portanto tem acesso regular (leitura aleatória) dos dados. No entanto, as garantias oferecidas pela camada de lotes (escalabilidade, consistência, tolerância a falhas) não atendem os requisitos de latência em um sistema em tempo real, para isso a camada de velocidade complementa os dados das visões com dados diretamente do conjunto mestre em tempo real diretamente para a camada de serviço (MARZ; WARREN, 2015).

2.3.2 Arquitetura *Kappa*

Em contraposição à arquitetura *Lambda*, observações práticas de Kreps (2014) mostram que o sistema de fila de mensagens (no exemplo *Apache Kafka*) já traz as garantias de escalabilidade, consistência, tolerância a falhas, replicação e armazenamento de longo prazo. Com isso, Kreps (2014) propõe que as camadas de lotes e velocidade sejam unificadas em uma camada de processamento de fluxo, cujos resultados sejam entregues continuamente para a camada de serviço através de um SGBD, definindo assim a arqui-

tetura *Kappa*. Essa proposta simplifica a aplicação de três implementações para duas, eliminando a repetição de tarefas executadas pelas camadas de lotes e velocidade que produziam o mesmo resultado.

Em sincronia com os desenvolvimentos em arquiteturas de processamento de fluxo de dados, durante as últimas duas décadas foram construídas diversas plataformas de processamento para *Big Data* e *Data Streams*.

2.3.3 Plataformas *MapReduce* e *Apache Hadoop*

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).

2.3.4 Plataforma *Apache Spark*

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).

Enquanto programas *MapReduce* fazem sua entrada de dados por leitura de disco, executam a função *Map* em todos os items, agrupam, executam *Reduce* e armazenam o resultado em disco novamente, RDD opera com um conjunto de trabalho distribuído em formato de memória compartilhada com restrições. Esse conjunto de trabalho distribuído

facilita a operação de programas iterativos que são típicos de análise, mineração de dados e aprendizado de máquina.

Uma das extensões de *Apache Spark* é *Spark Streaming*, que é um sistema de processamento de fluxo de dados escalável e tolerante a falhas (ZAHARIA et al., 2016a; ZAHARIA et al., 2016b). *Spark Streaming* implementa processamento incremental de fluxo de dados usando o modelo de fluxos discretizados em que dividem-se os dados de entrada em micro-lotes (ex: a cada 100 milissegundos) e combinam-se regularmente com o estado nos RDDs para produzir novos resultados (ZAHARIA et al., 2016a). Essa estratégia traz benefícios sobre os sistemas de fluxos de dados distribuídos tradicionais, pois permite a consistência e recuperação de falhas rapidamente, devido à (*RDD lineage*) e à combinação do fluxo de dados com consultas em lotes e interativas (ZAHARIA et al., 2016b; LOPEZ, 2018).

2.3.5 Plataforma *Apache Storm*

Apache Storm é um sistema de computação tolerante a falhas em tempo real que de fluxo de dados (FOUNDATION, 2020; LOPEZ, 2018). Ao invés de executar trabalhos (*jobs*) como algumas ferramentas citadas anteriormente, *Apache Storm* . 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).

Além de topologias e nós mestre, outros componentes do funcionamento dessa ferramenta são os *spouts* e os *bolts*. *Spout* representa uma fonte de dado da ferramenta, sendo um ponto de entrada que lê os dados de fontes externas, converte-os para um fluxo de dados e emite-os para dentro da topologia. *Bolts* recebem os dados de um *spout* e processam esses dados (filtragem, funções de agregação e união, etc.).

Cada processo *worker* no *Storm* é uma instância de Java Virtual Machine (JVM) que executa um conjunto de tarefas para uma topologia, processando um ou mais executores. Um executor é uma *thread* gerada por um processo *worker*. Cada executor pode processar uma ou mais tarefas para um mesmo componente (*spout* ou *bolt*). O número de processos *workers*, executores e tarefas (para os *spouts* e *bolts*) que são passados como parâmetro (*parallelism hint*) definem o "paralelismo" do *Storm*. A principal característica desse paralelismo é que ele pode ser alterado em tempo de execução da topologia.

2.4 Plataforma *Apache Flink*

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, 2020a). 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.

2.4.1 Arquitetura

Quando *Flink* é inicializado, um processo gerenciador de trabalho (*Job Manager*) e múltiplos gerenciadores de tarefa (*Task Manager*) são criados. Quando um código de programa é submetido, o cliente transforma-o em um grafo acíclico direcionado - *data flow* - e submete-o ao gerenciador de trabalho. Segundo Carbone et al. (2015), essa fase de transformação examina o esquema dos dados trocados entre os operadores e cria serializadores e outros códigos para otimização da futura execução. O gerenciador de trabalho coordena toda execução distribuída do grafo *data flow*. Ele rastreia o estado e o progresso de cada fluxo, agenda novos operadores e coordena os *checkpoints* e recuperação. Para alta disponibilidade, o gerenciador de trabalho em disco um conjunto mínimo de metadados em cada *checkpoint* para um armazenamento tolerante a falhas, de modo que esse gerenciador possa recuperar a execução do grafo a partir desse ponto. O processamento de dados ocorre no *Task Manager* que executa um ou mais operadores que produzem fluxos de dados, e reportam seus estados ao gerenciador de trabalho.

A pilha de componentes de software do *Apache Flink* é composta em camadas. A camada *core* é vista como um mecanismo de processamento e execução de fluxo de dados, enxergando o processamento em lote como um caso especial (LOPEZ, 2018; CARBONE et al., 2015). A camada de APIs é composta pelo *DataStream API*, que processa dados infinitos ou fluxos de dados, e pelo *DataSet API*, que processa dados finitos ou dados em lote. Junto ao *core*, essas APIs montam planos de execução otimizados para cada tipo de conjuntos de dados, gerando programas executáveis pelo *core*. Na camada de bibliotecas (*libraries*), há bibliotecas específicas para cada domínio que geram programas API *Data Stream API* ou *DataSet API*. Essas bibliotecas são: *FlinkML* para aprendizado de máquina, *Gelly* para processamento de grafos, *Table* para domínios relacionais (SQL), e CEP (*Complex Event Processing*) para processamento de eventos.

2.4.2 Abstrações e estruturas do *Apache Flink*

Na plataforma *Apache Flink*, as principais abstrações são programas, *Dataflows* (fluxo de dados), e transformações (operações ou operadores) (CARBONE et al., 2015; Apache Flink, 2020b). Um fluxo de dados (*Dataflow*) é definido como um fluxo de registros. Já as

transformações são operações (*map, filtering, reduction, join*, etc.) onde um *data stream* é consumido, processado, e um novo fluxo de dados gerado como saída. Cada uma dessas transformações pode ser paralelizada por um parâmetro de paralelismo (Apache Flink, 2020a).

Um programa *Flink* é mapeado para um grafo acíclico direcionado, *data flow*, utilizado pelo *Job Manager* (CARBONE et al., 2015). Esse grafo é composto por operadores de transformação e fluxo de dados (Apache Flink, 2020a). Para facilitar o paralelismo desse grafo de execução, os operadores que agem sobre os fluxos de dados podem ser divididos em sub-tarefas que são executadas pelos *slots* dos *Task Manager*, e os fluxos de dados podem ser particionados entre os operadores consumidores e produtores.

Cada *data flow* dos programas do *Apache Flink* inicia execução com uma fonte de dados e termina com um *sink* que escreve os dados de saída em algum sistema de armazenamento suportado, como *Apache Kafka, Amazon Kinesis Streams, Hadoop Filesystem* e *Apache Cassandra* (Apache Flink, 2020a).

2.4.3 Tolerância a falhas

O *Apache Flink* implementa um mecanismo de tolerância a falhas combinando repetição e *checkpoint* dos fluxos (CARBONE et al., 2015; Apache Flink, 2020a). Um *checkpoint* está relacionado com pontos específicos dos fluxos de entrada, juntamente com o estado dos operadores. Um fluxo de dados pode ser retornado a partir de um *checkpoint*, mantendo a consistência de "exatamente uma vez" (não há dados duplicados e nem dados que não sejam processados), e restaurando o estado dos operadores e eventos naquele momento. Portanto, as falhas são tratadas de forma transparente e não afetam a exatidão da execução de um programa *Flink* (Apache Flink, 2020a).

O algoritmo de *checkpoint* assíncrono e incremental um impacto mínimo em latência no processamento (CARBONE et al., 2015). Além disso, para reduzir o tempo de recuperação, o *Apache Flink* gera *snapshots* do estado dos operadores, incluindo a posição atual dos fluxos de entrada, em intervalos regulares.

O *Apache Flink* realiza computações com estado (*stateful*) que guardam eventos ou resultados intermediários para acessá-los posteriormente, contribuindo para planos de execução, mecanismo de recuperação de falhas e para lembrar de eventos passados para agregar dados (Apache Flink, 2020a; CARBONE et al., 2015).

O *Apache Flink* considera o processamento em lotes como um caso especial de fluxo de dados, que nesse caso é limitado em número de elementos. Para esse tipo de dados existem estruturas de dados e algoritmos específicos, como o *DataSet API* e operações próprias (agregações, uniões, interações) (CARBONE et al., 2015).

Para o processamento em lote, não há o mecanismo de *checkpoint* como há para o fluxo de dados. No lugar, a recuperação é feita repetindo completamente o fluxo ou repetindo as últimas partições perdidas do fluxo intermediário materializado.

Capítulo 3

Considerações Finais

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.

Referências

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 Flink 1.10 Documentation: Dataflow Programming Model**. 2020. Disponível em: <https://ci.apache.org/projects/flink/flink-docs-release-1.10/concepts/programming-model.html>.

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

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

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

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/>.

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.

FARIA, E. R. d.; 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 2015. ISSN 1573-756X. Disponível em: <https://doi.org/10.1007/s10618-015-0433-y>.

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

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

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

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>.

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/>.

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

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>.

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.

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>.

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>.

TANENBAUM, A. S.; STEEN, M. V. **Distributed systems: principles and paradigms**. [S.l.]: Maarten van Steen, 2018. ISBN 978-90-815406-2-9.

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>.

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.