UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA IMPLEMENTAÇÃO DISTRIBUÍDA EM NÉVOA DO ALGORITMO DE DETECÇÃO DE NOVIDADE EM FLUXOS DE DADOS MINAS

LUÍS HENRIQUE PUHL DE SOUZA

ORIENTADOR: PROF. DR. HERMES SENGER

São Carlos – SP

Fevereiro/2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA IMPLEMENTAÇÃO DISTRIBUÍDA EM NÉVOA DO ALGORITMO DE DETECÇÃO DE NOVIDADE EM FLUXOS DE DADOS MINAS

LUÍS HENRIQUE PUHL DE SOUZA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação 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 – SP

Fevereiro/2020

AGRADECIMENTOS

O presente trabalho de pesquisa está sendo realizado com o apoio parcial do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

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 deriva 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 ~~it's~~ 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 FIGURAS

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, por vezes omitindo sistema operacional e outros sistemas, ~~, 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 outro tipo de ~~, podem gerar tráfego ~~, maligno à sociedade~~maligno , como o gerado pela *botnet* mirai em 2016 (**??**). 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, ~~recentemente~~dentro destes tópicos, detecção de novidades (*Novelty Detection*, ND). ~~ND além de classificar em modelos conhecidos permite classificar novos~~ Dentre as técnicas de mineração de fluxo de dados, classificadores podem ser utilizados para detectar padrões ~~, já que trata problemas como evolução e deriva de conceito que são característicos de fluxos contínuos de dadose, consequentemente, permite que ações sejam tomadas corretamente mesmo em face a~~ conhecidos e, em conjunto com algoritmos de detecção de novidades ou detecção de anomalias, detectar novos padrões~~nunca vistos~~. 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 assinatura existente~~ , sem assinaturas existentes em bancos de dados de ataques conhecidos.

Análises como *Data Stream Mining* e ~~ND~~ ND são ~~tradicionalmente~~ geralmente implementadas sobre o paradigma de computação na nuvem (*Cloud Computing*) e, recentemente, sobre paradigmas ~~como computação em névoa~~ como (*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 **??**), 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, **??**) avaliou os algoritmos ECSMiner (**??**), AnyNovel (**??**) e MINAS (**??**), 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 (**??**).

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 borda e~~ , 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; ~~sendo~~ 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 **??**), 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 **??**) e **??**) abordam detecção de intrusão em redes utilizando algoritmos de ND em DS, porém com perspectivas diferentes. O primeiro ~~observa~~ investiga *IoT* e processamento em *fog* ~~,~~ e baseia-se em um algoritmo genérico de detecção de novidade. O segundo trabalho ~~observa~~ 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, de outro, 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 ~~deriva~~ 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 **??**), esta pesquisa escolheu

investigar o algoritmo MINAS (**??**) 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 ~~secundários~~ 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 ~~busca-se~~ 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

são 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 (**??**), irá classificar e detectar novidades em DS. Também nesse passo, a implementação será validada comparando os resultados de classificação obtidos com os resultados de classificação do algoritmo original MINAS. Posteriormente, serão realizados experimentos com a implementação e variações em *data sets* e cenários de distribuição em *fog*, coletando as métricas de classificação e escalabilidade.

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

1.4 Organização do trabalho

O restante desse trabalho segue a estrutura: Capítulo 2 aborda conceitos teóricos e técnicos que embasam esse trabalho; Capítulo 3 enumera e discute trabalhos relacionados e estabelece o estado da arte do tema detecção de novidade em fluxos de dados e seu processamento; Capítulo 4 descreve a proposta de implementação, discute as escolhas de plataformas e resultados esperados. Também são discutidos no Capítulo 4 os desafios e resultados preliminares encontrados durante o desenvolvimento do trabalho. Capítulo 5 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, ~~os~~ 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 ~~à~~ a redes de dispositivos IoT , entre outras aplicações. ~~Iniciando com o ambiente mais tradicional, a computação em nuvem () é~~ aplicada a vários problemas e neste trabalho seu papel em sistemas IoT é fornecer ~~recursos e garantias ilimitadas e em que dispositivos enviam todos dados relevantes ao sistema~~ . O segundo e terceiro ambiente ~~é a~~ são computação de borda (*edge computing*) e a computação em névoa (*fog computing*)~~que passa a utilizar~~ , que utiliza os recursos computacionais distribuídos presentes em nós localizados entre os dispositivos de borda e a nuvem~~com diversas intensões~~ , com diversas intenções, desde privacidade até redução de latência.

2.1.1 Computação em Nuvem

A computação em nuvem (~~*cloud computing*~~*cloud computing*), ou simplesmente nuvem (*cloud*), habilita o acesso através da rede a um grupo compartilhado de recursos de computação configuráveis (servidores, redes, aplicativo, armazenamento, serviços, etc.), que podem ser provisionados ou liberados sob demanda rapidamente com o mínimo esforço de gerenciamento ou interação com o ~~provedor de serviços (??).~~ (??). As principais características do ~~*cloud computing*~~ ambiente *cloud computing*, segundo **??**) são:

- **Serviço sob Demanda:** o cliente pode provisionar ou liberar capacidades de computação (ex: tempo de processamento e armazenamento) conforme o necessário~~sem requer~~ , 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 ~~demanda do cliente~~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 ~~fornecendo transparência~~ para o provedor de serviços e para o cliente fornecendo transparência sobre as capacidades que foram consumidas.

Segundo, **??**), 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 ~~empresa terceira~~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 ~~de~~ um mesmo interesse (requerimentos de segurança, desempenho, políticas, etc.). Esse tipo de nuvem pode ser gerenciado pelo próprio grupo, ou ~~organizações terceiras~~por outra organização, podendo estar dentro ou fora das instalações ~~da empresa proprietária~~das empresas proprietárias;

- **Nuvem pública:** ~~a infraestrutura da nuvem~~ É gerenciada e operada por um provedor de nuvem e a infraestrutura é provisionada e oferecida para uso público. ~~É gerenciado e operado por um provedor de nuvem.~~

- **Nuvem híbrida:** a infraestrutura desse tipo de nuvem é uma composição de ~~duas~~ 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 (**??**). Na borda, é possível fazer armazenamento, processamento e descarregamento de dados, assim como distribuir as requisições e entregar os serviços das nuvens aos usuários. **??**) ressalta que essas capacidades ~~dentre outras~~ (dentre outras) dos nós da borda (*edge nodes*) ~~possibilita~~ 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 ~~o *edge computing*~~ a computação de borda pode aumentar a privacidade dos dados, uma vez que ~~o dado pode ser processado~~ eles podem ser processados no próprio dispositivo final.

A computação de borda tenta trazer a computação mais próxima das fontes de dados~~, como mostra a Figura **??**~~. 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 ~~são~~incluem: análise de vídeo; em sistemas críticos para redução de latência~~da nuvem para sistemas críticos~~; descarregar a nuvem de parte da computação; privacidade dos dados produzidos~~mantendo eles~~ , mantendo-os fora de ambientes públicos; redução das cargas de dados na rede ~~;~~ e processamento distribuído de sensoriamento massivo em cidades inteligentes (**??**).

~~Paradigma de *Edge Computing* (**??**).~~

2.1.3 Computação em Névoa

??) e **??**) mencionam que a enorme massa de dados gerados por ambientes IoT pode ser processada ~~pela~~ em nuvem, entretanto ~~a latência~~ produzida pela transferência desses dados para a nuvem e o retorno do resultado pode não ~~são tolerados~~ ser toleradas por sistemas críticos que sejam sensíveis a latência (monitoramento de saúde e resposta a emergências). ~~O autor **??**)~~ ainda acrescenta que enviar tantos dados ~~a *cloud*~~ à nuvem para processamento e armazenamento pode ser ineficiente e não escalável~~devido a~~ , 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 os~~ . 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 (**??**). 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~~ e dos , mas não necessariamente localizados na borda, dando a esses nós características de uma nuvem (**??????**). Esse tipo de computação evita a sobrecarga dos dispositivos de borda. **??**) e **??**) ~~veem o *fog computing*~~ consideram computação em névoa como complementar da ~~*edge computing*~~computação em borda, podendo a computação em névoa aproveitar os recursos da nuvem e da borda. **??**) considera que a principal diferença entre esses dois tipos de computação está no número de camadas. Enquanto ~~o~~ *edge computing* tem ~~camadas menores~~, pois atua só nos dispositivos de borda, ~~o~~ *fog computing* tem mais camadas e um modelo hierárquico, pois não atua só na camada de borda.

Segundo **??**) e **??**), 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:** ~~a computação em névoa~~ foi proposta para atender aplicações que requeiram baixa latência (monitoramento de saúde, jogos, realidade aumentada, etc.);
- **Distribuição geográfica:** ~~o *fog computing* possui~~ computação em névoa pode possuir milhares de sensores e dispositivos distribuídos geograficamente~~e a~~ , com consciência ~~da localização deles~~ 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, ~~então~~ 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 (**??**). ~~Um caso~~ Além de mineração de dados ~~é o~~ tradicional, *Big Data* ~~onde o conjunto de dados~~ trata de conjuntos de dados que não ~~pode ser processado em um~~ podem ser processados em tempo viável, devido a limitações como memória ou armazenamento principal.

Além da dimensão de armazenamento, outra dimensão que afeta a maneira como dados são modelados e manipulados é o tempo. ~~Um Fluxo de Dados~~ (*Data Stream*) é uma sequência de registros associados ao tempo real, ilimitados, que excede recursos de armazenamento (**??**). ~~Modelos de mineração de~~ 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 (**????**).

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, ~~portanto~~ de forma que a distribuição dos requisitos em múltiplos nós computacionais em um sistema distribuído ~~é igualmente~~ pode ser necessária (**??**).

Computação distribuída é a área da ciência da computação que estuda sistemas ~~onde~~ 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 (**??**).

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 **??**), 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 a 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 a à construção de sistemas *Big Data* e *Data Streams*foram propostas , arquiteturas de processamento de fluxo de dados distribuído , como foram propostas, como a arquitetura *Lambda* (**??**) e *Kappa* (**??**), 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 (**??**). 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, portanto 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 (**??**).

A Figura **??** ilustra uma implementação da arquitetura onde a *Apache Kafka*, *Apache Storm*, *Apache Hadoop* implementam o conjunto mestre, camada de velocidade e camada de lotes

respectivamente.

Arquitetura com detalhes práticos (**??**).

2.3.2 Arquitetura *Kappa*

Em contraposição à arquitetura *Lambda*, observações práticas de **??**) 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, **??**) 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 arquitetura *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, uma 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 (**??**).

Em prática, um *cluster MapReduce* tem centenas de tem 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* (**??**).

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* (**??**).

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 ~~de apenas leitura~~ 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 (**??**).

Enquanto programas *MapReduce* fazem sua entrada de dados por leitura de ~~um~~ disco, executam a função *Map* em todos os items, agrupam, executam *Reduce* e armazenam o resultado em disco novamente, ~~o~~ 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 ~~do~~ de *Apache Spark* é ~~o~~ *Spark Streaming*, que é um sistema de processamento de fluxo de dados ~~escalável e tolerante a falhas (????). O~~ (????). *Spark Streaming* implementa ~~o~~ processamento incremental de fluxo de dados usando o modelo de fluxos discretizados em que ~~divide-se~~ dividem-se os dados de entrada em micro-lotes (ex: a cada 100 milissegundos) e ~~combina-se~~ combinam-se regularmente com o estado nos RDDs para produzir novos resultados (**??**). Essa estratégia traz benefícios sobre os sistemas de fluxos de dados distribuídos tradicionais, pois permite a consistência e recuperação de falhas rapidamente~~devido a linhagem de RDD~~, devido à (*RDD lineage*) e à combinação do fluxo de dados com consultas em lotes e interativas (**????**).

2.3.5 Plataforma *Apache Storm*

~~O~~ *Apache Storm* é um sistema de computação tolerante a falhas em tempo real que ~~facilita o processamento~~ de fluxo de dados (**????**). Ao invés de executar trabalhos (*jobs*) como algumas ferramentas citadas anteriormente, ~~o~~ *Apache Storm* ~~executa topologias~~. Os *jobs* eventualmente finalizam, e as topologias executam continuamente até serem finalizadas por comandos. Uma topologia ~~constitui de processos~~ 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*. ~~O *spout*~~*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. ~~Os *bolts*~~*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) ~~e~~ 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 (~~tem um~~ têm início e ~~um~~ fim) e ilimitados (não ~~tem um~~ têm fim definido) (**??**). Essa plataforma segue um paradigma que abrange o processamento de fluxos de dados contínuos e o processamento em lote (**????**). 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 o~~

Quando *Flink* é inicializado, ~~o~~ um processo gerenciador de trabalho(*Job Manager*~~e um ou mais~~) 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 ~~*Job Manager*, como pode ser observado na Figura Figura **??**~~gerenciador de trabalho . Segundo **??**), 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 ~~*Job Manager*~~ 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 ~~*Job Manager*~~ ~~persiste~~ 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 ~~*Job Manager*~~gerenciador de trabalho .

~~Processo do *Apache Flink* (??)~~

A pilha de componentes de software do *Apache Flink* é composta ~~pelas seguintes camadasda Figura ??~~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 (**????**). 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.

~~Componentes de software do *Apache Flink* (??).~~

2.4.2 ~~*Data flow*~~ Abstrações e ~~*data streams*~~estruturas do *Apache Flink*

~~Os~~ Na plataforma *Apache Flink*, as principais abstrações são programas, ~~*data streams*~~*Dataflows* ~~ou~~ (fluxo de dados~~ as~~), e transformações ~~são as principais abstrações do *Apache Flink* (????). Esse~~ (operações ou operadores) (**????**). Um fluxo de dados (*Dataflow*) é definido como um fluxo de registros. Já as transformações são operações (*map, filtering, reduction, join*, etc.) ~~aplicadas de forma incremental nos~~ onde um *data* ~~*streams, gerando*~~ *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 ~~(??)~~(**??**).

Um programa *Flink* é mapeado para um grafo acíclico direcionado, *data flow*, utilizado pelo *Job Manager* (**??**). Esse grafo é composto por operadores de transformação e fluxo de dados (**??**). Para facilitar o paralelismo desse grafo de execução, os operadores que agem ~~sobres~~ 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* ~~iniciam~~ inicia execução com uma fonte de dados e ~~terminam~~ termina com um *sink* que escreve os dados de saída em algum sistema de armazenamento suportado, como ~~:~~ *Apache Kafka, Amazon Kinesis Streams, Hadoop Filesystem~~,~~ ~~Apache Cassandra(??). Na Figura ??, pode-se observar um exemplo de programa~~ Apache ~~Flink escrito em Java e seu grafo de execução. Nesse exemplo, define-se como fonte de dados o~~ Apache Kafka~~. Em seguida, aplica-se uma transformação~~ map~~, e depois outra transformação de agrupamento por um dos atributos dos dados e por uma janela de 10 segundos. Por fim, o resultado é passado para um~~ sink.~~* e *Apache Cassandra* (**??**).

~~Exemplo de código e *data flow* do *Apache Flink* (??)~~

2.4.3 Tolerância a falhas

O *Apache Flink* implementa ~~a~~ um mecanismo de tolerância a falhas combinando repetição e *checkpoint* dos fluxos (**????**). 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* (**??**).

O algoritmo de *checkpoint* assíncrono e incremental ~~garante~~ um impacto mínimo em latência no processamento (**??**). Além disso, para reduzir o tempo de recuperação, o *Apache Flink* ~~tira um~~ gera ~~*snapshot*~~*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 (**????**).

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 ~~, há estrutura~~ existem estruturas de dados e algoritmos específicos, como o *DataSet API* e operações próprias (agregações, uniões, interações) (**??**).

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.

2.5 Detecção de Novidade

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 $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_n, \ldots$ (**??**).

No âmbito de classificação de dados, parte da área de aprendizado de máquina, os métodos de detecção de novidade (~~também nomeado detecção de anomalia,~~ *Novelty Detection*, ND) lidam com o reconhecimento e a classificação ~~de exemplos que diferem de exemplos anteriores~~ (**????**). Esses métodos tratam da classificação em fluxos de dados que ~~evolvem~~ evoluem com o tempo, levando em consideração as características desse tipo de fluxos.

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

Os métodos de ND são ~~utilizados no reconhecimento de diversos padrões novidade ou anomalias e são aplicadas~~ aplicados a diversos problemas como ~~Detecção de Intrusos (????????) , Detecção de Falhas (??), Diagnósticos Médicos (??), Detecção de Regiões de Interesse em Imagens (??), Detecção de Fraudes (????), Filtros de Spam (??) e Detecção de Variações Comportamentais em um Jogador~~ detecção de intrusos (**????????**), detecção de falhas (**??**) , diagnósticos médicos (**??**), detecção de regiões de interesse em imagens (**??**), detecção de fraudes (**????**), filtros de spam (**??**) e detecção de variações comportamentais em um jogador (**??**).

Alguns métodos de NDutilizam tratam de novidades ~~e anomalias~~ 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. ~~No entanto, a~~ Além da abordagem de classificação binária~~não é adequada para representar múltiplos conceitos~~ , em um mesmo conjunto de dados, para isso é necessário abordar ND como classificação multi-classe. ~~Entretanto, alguns~~ Alguns métodos que abordam ND 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 (**????**).

A maioria dos métodos de ND 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 ~~classifica~~ 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 (**??**).

Dentre os métodos de ND 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 (**??????**).

2.6 ~~O algoritmo MINAS~~

2.5.1 O algoritmo MINAS

Um algoritmo de ND que tem recebido atenção nos últimos anos é o algoritmo MINAS, originalmente proposto por **??**), refinado por **??**) e recentemente aprimorado por **??**)~~com~~ , com o uso de conceitos *Fuzzy*, e expandido por **??**), 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 usa por base algoritmos de agrupamento não supervisionados como *K-means* e *CluStream*. ~~As duas fases, com exceção do método de detecção de padrões novidade da fase *Online*, são ilustradas na .~~

~~Visão geral do algoritmo MINAS com fases *Offline* (a) e *Online* (b) (**??**)~~

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

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

etiqueta do grupo original e todos ~~os conjuntos~~ *micro-clusters* são ~~unidos~~ arranjados em um único conjunto formando o modelo de decisão.

Na fase *Online*, listada no Algoritmo 1, o algoritmo MINAS opera com ~~dois métodos~~três operações: classificação de novos exemplos, detecção de padrões-novidade e atualização do modelo de decisão (**??**). O primeiro método é o de 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 ~~euclideana~~ euclidiana entre o exemplo e todos *micro-clusters* do modelo, selecionando o *micro-cluster* de menor distância. Se a distância entre o exemplo e o centro do *micro-cluster* for menor que o raio do *micro-cluster* , o exemplo é classificado com a etiqueta do *micro-cluster* e o sumário estatístico do *micro-cluster* é atualizado. Caso a distância ~~seja maior o~~ (mínima no modelo) seja maior que o raio, o exemplo é marcado como desconhecido e armazenado em conjunto próprio (**????**).

O segundo método da fase *Online* é a detecção de padrões novidade, que é executada quando o tamanho do conjunto de desconhecidos é maior que um parâmetro predefinido. Esse método executa o agrupamento (*clustering* descrito na fase *Offline*) e valida os *micro-clusters* gerados verificando sua representatividade e coesão.

Algoritmo 1: MINAS (**????**)

Entrada: $Modelo, FCD, T, NumMinExemplos, ts, P$

```
1:  MemTmp ← ∅
2:  MemSleep ← ∅
3:  for all exemplo ∈ FCD do
4:      (Dist, micro) ← micro-mais-proximo(exemplo, Modelo)
5:      if Dist < raio(micro) then
6:          exemplo.classe ← micro.rotulo
7:          atualizar-micro(micro, exemplo)
8:      else
9:          exemplo.classe ← desconhecido
10:         MemTmp ← MemTmp ∪ exemplo
11:         if |MemTmp| ≥ NumMinExemplos then
12:             Modelo ← deteccao-novidade(Modelo, MemTmp, T)
13:         end if
14:     end if
15:     TempoAtual ← exemplo.T
16:     if TempoAtual mod TamJanela == 0 then
17:         Modelo ← mover-micro-grupos-mem-sleep(Modelo, MemSleep, P)
18:         MemTmp ← remover-exemplos-antigos(MemTmp, ts)
19:     end if
20: end for
```

Para atribuição de etiquetas aos *micro-clusters* gerados, o algoritmo MINAS encontra no

modelo atual o *micro-cluster* mais próximo pela distância euclidiana e classifica em dois tipos de conceito. Se a distância é menor que um parâmetro predefinido, o novo *micro-cluster* gerado recebe como etiqueta o valor de extensão de conceito conhecido. Caso contrário, se o novo *micro-cluster* está mais distante, um novo conceito foi encontrado e a etiqueta marca ~~uma~~ um padrão novidade. Após a atribuição da etiqueta do novo *micro-cluster*, ele é adicionado ao modelo de decisão.

Capítulo 3

~~Trabalhos Relacionados~~

~~Este Capítulo trata dos trabalhos relacionados e estabelece o estado da arte dos tópicos Detecção de Novidades em Fluxos de Dados, e Processamento Distribuído de Fluxos de Dados.~~

2.1 ~~Algoritmos de Detecção de Novidades~~

O algoritmo MINAS, como já foi discutido na Seção 2.5.1, classifica exemplos e detecta novidades em DS e considera em sua composição *concept drift* e *concept evolution*, sendo capaz de classificar como extensão de classe conhecida e ~~reconhecer novas classes~~ identificar padrões novidade sem intervenção de especialista (**??**). Neste trabalho, consideram-se algoritmos derivados do algoritmo MINAS aqueles apresentados em trabalhos publicados após 2016, que estendem a implementação original ~~do MINAS~~ seguindo sua estrutura básica.

2.0.1 Algoritmo FuzzyND

O algoritmo FuzzyND, derivado do MINAS ~~é~~ foi proposto por **??**). FuzzyND incrementa o algoritmo original, aplicando a ele teorias de conjuntos *fuzzy* pela modificação da representação dos *clusters*. A modificação afeta ~~os métodos~~ o método de construção de *clusters*, método de classificação de exemplos e método de detecção de novidades de acordo com a nova representação.

A avaliação do algoritmo FuzzyND ~~é~~ foi feita por meio de experimentos usando 3 *data sets* sintéticos (*MOA3*, *RBF*, *SynEDC*) e por comparação com o MINAS. O método de avaliação utilizado baseia-se na matriz de confusão incremental descrita por **??**), extraindo dessa matriz duas métricas: acurácia (*Macro F-Score*) (**??**) e taxa de desconhecidos (*UnkR*) (**??**). Em geral, o

algoritmo FuzzyND detecta melhor novidades e, consequentemente, é mais robusto a valores atípicos (*outlier*), porém perde a capacidade de reconhecer padrões recorrentes.

2.0.2 Algoritmos MINAS-LC e MINAS-BR

O algoritmo MINAS-LC ~~é~~ foi proposto por **??**) e trata a classificação multi-rótulo, porém não trata evoluções de conceito (*Concept Evolution*). As alterações fundamentais propostas são: a representação de *cluster* onde MINAS-LC troca a etiqueta, que era única, por uma multi-rótulo; a transformação de problema aplicada ao conjunto de treinamento para transformá-lo de um conjunto multi-rótulo para um conjunto multi-classe (simplificação) em duas variações *Label Powerset* e *Pruned Sets* com mineração de conjunto de itens frequentes.

Já o trabalho de **??**), estende o algoritmo original para que classifique um exemplo com uma ou mais etiquetas usando a transformação *Binary Relevance*~~propondo o~~ , o que deu origem ao algoritmo MINAS-BR. O algoritmo modifica a representação do modelo, originalmente conjunto de *clusters*, para um grupo de *clusters* por classe (etiqueta). Também modifica o método de agrupamento, substituindo a inicialização do algoritmo *K-means*, originalmente aleatória, pelo algoritmo *Leader Incremental Clustering* (**??**).

O algoritmo MINAS-BR também é experimentalmente avaliado com 4 *data sets* sintéticos: *MOA-3C-5C-2D*, *MOA-5C-7C-2D*, *MOA-5C-7C-3* da ferramenta MOA (**??**) e *4CRE-V2* [1] gerados pelo método *Radial Basis Function* ~~(??). O~~ (????). O algoritmo MINAS-BR ~~é~~ foi comparado com 7 algoritmos da literatura também disponíveis na ferramenta MOA (**??**), diferente da avaliação do FuzzyND que compara diretamente com MINAS. ~~Os~~ Para análise, os 7 algoritmos ~~são~~ foram divididos em dois grupos ~~:~~ (??). O primeiro grupo de 3 algoritmos com acesso às etiquetas corretas para atualização do modelo e com a técnica ADWIN (*ADaptive WINdowing*) para detectar mudanças de conceito (*Concept Drift*) ~~;~~ O segundo grupo com os 4 algoritmos sem acesso às etiquetas corretas, ou seja, sem *feedback* externo, mesma condição do MINAS-BR (??)
.

A avaliação elencada por **??**) leva em consideração que as classes contidas no conjunto de testes podem não ter correlação direta com os padrões identificados pelos algoritmos. Para tratar a divergência, uma estratégia baseada em proposta anterior por **??**) ~~é~~ foi apresentada com alterações para exemplos multi-rótulo. Após associação entre padrões de novidade e classes novidade ~~é~~ foi possível calcular métricas tradicionais. A estratégia é executada na fase de classificação seguindo as regras:

[1] A versão original do *data set* 4CRE-V2 está disponível em ~~https://sites.google.com/site/nonstationaryarchive/home~~ ⟨https://sites.google.com/site/nonstationaryarchive/home⟩

1. após o consumo do exemplo X_n;
2. para todo padrão P_i (etiqueta atribuída) identificado sem associação até o momento;
3. com classes novidade y_j (etiqueta real) presentes em exemplos antes X_n;
4. preenche-se a tabela de contingência $\mathbf{T}_{(i,j)}$ relacionando padrão P_i e classe y_j;
5. calcula-se o grau de dependência $F1$ derivado da tabela de contingência $F1_{(i,j)} = f(\mathbf{T}_{(i,j)})$;
6. valores $F1_{(i,j)} = 0$ são descartados;
7. dentre os valores restantes: o padrão P_i é associado à classe y_j se $F1_{(i,j)}$ é máximo.

As métricas utilizadas por **??**) após a associação de classes e padrões são as tradicionais taxa de desconhecidos (*UnkRM*) e *F1M*. Os resultados apresentados indicam que MINAS-BR capturou todas as novidades dos *data sets* sintéticos de teste e mostrou, como esperado, melhores métricas que os 4 algoritmos equivalentes da literatura ficando abaixo dos 3 com *feedback* externo.

Os trabalhos ~~relacionados nessa , tem~~ abordados nessa Seção 2.5, têm em comum, além do algoritmo base, as métricas de avaliação acurácia (*Macro F-Score* e *Macro F-Measure* F1M) e taxa de desconhecidos, aplicadas com devido tratamento. Também é comum entre eles o uso de *data sets* sintéticos. Outro potencial não explorado do MINAS é em aplicações reais, ou seja, consumindo além de *data sets* reais, fluxos realistas em ambientes simulados ou reais porém considerando uso de recursos computacionais.

Observando a arquitetura dos algoritmos abordados na Seção 2.5 , nota-se as semelhanças: a fase offline centrada no processo de agrupamento e criação de modelo; a fase online dividida em classificação (com atualização das estatísticas do modelo) e detecção de padrões, onde novamente o processo de agrupamento é central. Portanto, apesar de outros trabalhos expandirem o algoritmo com diferentes técnicas, seu núcleo continua relevante[2] (**??????**).

[2]Propostas de modificação do algoritmo MINAS estão longe de serem exauridas. Não cabe ao presente trabalho expandir e validar conceitos de aprendizagem de máquina, porém alguns exemplos mencionados ainda não abordados são: a) diferentes métodos de cálculo de distância entre pontos além da distância euclidiana; b) a mudança de representação de *clusters*, atualmente hiper-esferas (**??**), para hiper-cubos tratando *data sets* onde as características representadas pelas dimensões são completamente independentes; c) um modo interativo onde o *cluster* é formado, mostrado ao especialista que o classifica como inválido (ruído ou não representativo) ou válido, podendo conter uma ou mais classes e, se ~~conter~~ contiver mais que uma classe corte em grupos menores até conter somente uma classe; d) ainda considerando interação com especialista, a possibilidade ~~dele~~ de injetar um exemplo não pertencente a uma classe, ou seja, marcar o exemplo como não pertencente a uma classe para ~~manter ele~~ mantê-lo na memória de desconhecidos e, eventualmente forçar criação de um *cluster* que represente uma classe geometricamente próxima mas semanticamente distinta; e) na fase *offline* a verificação de sobreposição de *clusters* pertencentes a classes distintas e tratamento adequado.

2.1 ~~Processamento Distribuído de Fluxo de Dados em Tempo Real~~

Capítulo 3

TRABALHOS RELACIONADOS

~~Nesta aborda-se~~

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

Nesta Capítulo, abordam-se trabalhos que aplicam ~~algoritmos de detecção de novidades em ambiente de processamento distribuído de~~ em de fluxo de dados em tempo real. Um sumário dos trabalhos abordados pode ser visto na Tabela 3.1.

3.0.1 ~~Ferramenta BigFlow~~

Tabela 3.1: Sumário dos trabalhos relacionados

Trabalho	Plataforma	Técnica	Conjunto de dados	Métricas
Ferramenta BigFlow (??)	*Python, flowtbag, Apache Kafkae Apache Flink*	*Hoeffding Tree, OzaBoosting, Leveraging Bag* e comitê	*MAWILab*	Acurácia (geral e por classe), Taxa de bytes
Ferramenta CATRACA (??)	*Virtual Network Function, Apache Kafkae Apache Spark*	PCA, SFS, e SVM-RFE	NSL-KDD, GTA/UFRJ e NetOp	Acurácia, precisão, sensibilidade e F1-score
Arquitetura IDSA-IoT(??)	Java, *Apache Kafkae Python*	ECSMiner, AnyNovel e MINAS	*Kyoto 2006+*	Fnew, Mnew e erro

3.1 Ferramenta BigFlow

Proposta por **??**), a ferramenta BigFlow é um sistema de detecção de intrusão em rede (*Network Intrusion Detection System*, NIDS) baseado em detecção de anomalias. Duas aborda-

gens, detecção por assinatura e detecção por anomalia, ~~são de uso frequente como o mecanismo de detecção de intrusão na construção de NIDS~~ . Para a detecção de novos tipos de ataque (*zero day*), a abordagem de detecção por anomalia é vantajosa, em contraste com a abordagem de detecção por assinatura, devido ao tempo de resposta (que ~~evolve~~ envolve a identificação e criação de uma assinatura), grande demais para prevenir esse tipo de intrusão.

A ferramenta BigFlow é composta ~~por dois~~ pelos módulos ~~:~~ de extração de atributos e de aprendizado confiável. O módulo de extração de atributos ~~, ilustrado na ,~~ é responsável por coletar ~~pactoes~~ da rede monitorada, ~~transformar esses pacotes em fluxos com as~~ com estatísticas de comunicação e enviar ~~os~~ informações desses fluxos como exemplos para o módulo de aprendizado confiável. O módulo de aprendizado confiável, ~~ilustrado na ,~~ 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.

~~Módulo de extração de atributos da ferramenta BigFlow (??).~~

~~Módulo de aprendizado confiável da ferramenta BigFlow (??).~~

??) destaca que *data sets* adequados para NIDS são poucos~~devido o~~ , devido ao conjunto de qualidades que os mesmos devem atender, como realismo, validade, etiquetamento, grande variabilidade e reprodutividade (disponibilidade pública).

Para avaliar o desempenho de NIDS, o *data set* MAWIFlow é proposto por **??**). ~~Originário~~ Este *data set*é derivado do *data set Packet traces from WIDE backbone, samplepoint-F* , composto por seções de captura de pacotes diárias de 15 minutos de um link de 1Gbps entre Japão e EUA, com início em 2006 continuamente até hoje, anonimizados e etiquetados por MAWILab (**????**). Desse *data set* original, o *data set* MAWIFlow utiliza apenas os eventos de 2016, dos quais 158 atributos são extraídos resultando em 7.9 TB de captura de pacotes. Além disso, os dados são estratificados para redução de seu tamanho a um centésimo~~mantendo ,~~ as proporções de etiquetas (Ataque e Normal)~~facilitando ,~~ o compartilhamento e avaliação de NIDS, além de atender às qualidades anteriormente mencionadas.

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

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

~~Quanto a~~ Quanto à distribuição do processamento, ~~ilustrada na ,~~ 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*).

~~Visão geral da arquitetura e distribuição da ferramenta BigFlow (??).~~

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

~

~

3.1.1 ~~Ferramenta CATRACA~~

3.2 Ferramenta CATRACA

O trabalho de **??**) 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~~, ilustradas na Figura **??**~~.

~~Arquitetura em camadas da ferramenta CATRACA (??).~~

Na camada de captura, pacotes são capturados da rede e ~~transformadas em~~ são geradas informações sumário de fluxos por uma aplicação *Python* utilizando a biblioteca *flowtbag*[2]~~e~~

[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/⟩.

~~os fluxos~~ . Esses sumários são enviados para um tópico de um sistema de fila de mensagens (*Apache Kafka*) ~~hospedada~~ 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, **??**) 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~~ . Ambos consumiam dados de um tópico de um sistema de fila de mensagens (*Apache Kafka*) e produziam para outro tópico. O primeiro programa consiste de um detector de ameças composto por uma rede neural classificadora escrito em *Java*~~e é~~ , que foi testado com o conjunto de dados sintético UFRJ/GTA (**??**). 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~~com 214200~~ , 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 ~~atende~~ 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.

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

Também para a avaliação da ferramenta CATRACA, ~~utilizou-se~~ 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 ~~utiliza~~ 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 é~~. Essa ferramenta foi avaliada com métricas de qualidade, métricas de escalabilidade e dois conjuntos de dados relevantes. No entanto, o algoritmo de detecção de anomalias desenvolvido para a ferramenta consiste de um modelo de classificação pelo método árvore de decisão e a atualização do modelo durante a fase *Online* depende de todos os exemplos do último intervalo de atualização. Esse tipo de algoritmo de detecção de anomalias ~~não é capaz de lidar adequadamente com as características de fluxos contínuos de dados~~ de dados, como os descritos na Seção 2.5 (*Concept Drift*, *Concept Evolution*, limitado a ler o conjunto somente uma vez), que são atendidos por algoritmos de detecção de novidade.

3.3 Arquitetura IDSA-IoT

A arquitetura IDSA-IoT, proposta por **??**), tem por objetivo monitorar uma rede local com dispositivos IoT e detectar tentativas de intrusão e alguma subversão do comportamento das transmissões destes dispositivos. O principal destaque da arquitetura é a distribuição de tarefas do sistema de detecção de intrusão entre nós na ~~rede de borda ()~~ 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 Figura 3.1 ilustra a estrutura física da arquitetura IDSA-IoT, destacando os dispositivos IoT, dispositivos de borda e nuvem pública.

A arquitetura proposta é avaliada com três algoritmos de detecção de novidade: ECSMiner (**??**), AnyNovel (**??**) e MINAS (**??**). 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

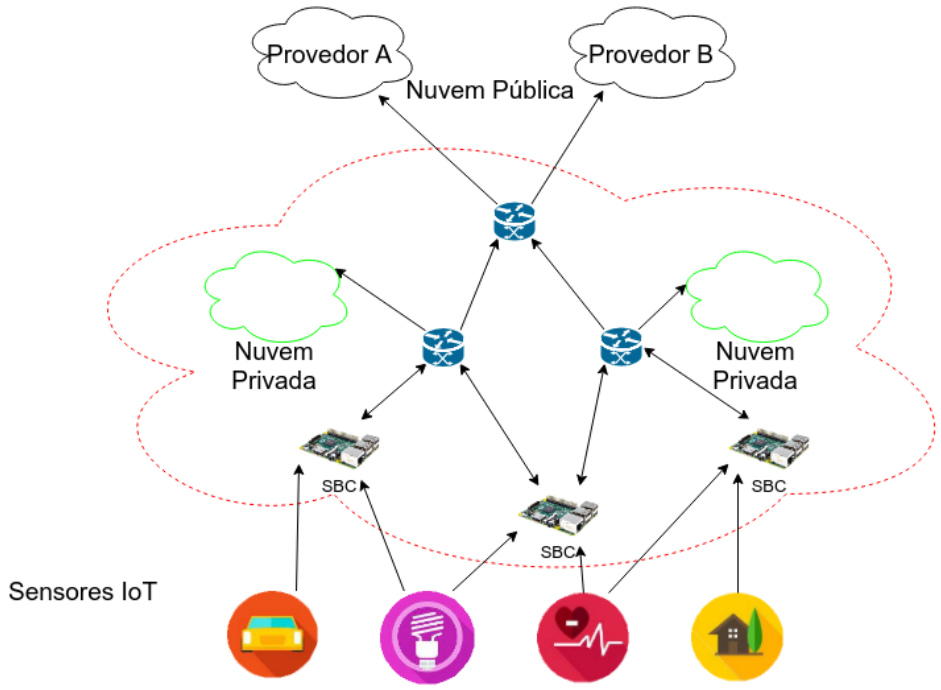

Figura 3.1: Estrutura Física da Arquitetura IDSA-IoT. Produzida e traduzida por ??).

até dezembro 2015. Esse *data set* tem as características desejáveis de um conjunto para detectção 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 ~~avalição~~ avaliação envolveu dois computadores. Para tanto, um computador pessoal com recursos ~~convecionais~~ 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~~métrica Fnew~~. O algoritmo AnyNovel não apresentou consistência nos resultados e o consumo de recursos computacionais (memória) foi elevado.

Com as avaliações realizadas, a arquitetura IDSA-IoT ~~opta por distribuir~~ as tarefas de mineração dos fluxos para detecção de intrusão em serviços e aloca os serviços em diferentes camadas físicas, conforme ilustrado na Figura 3.2.

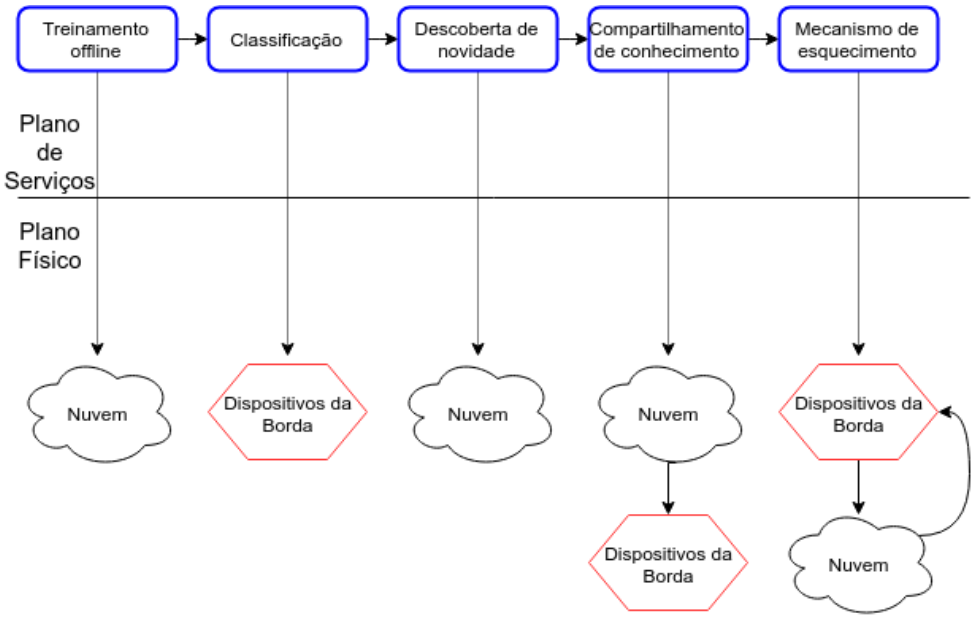

Figura 3.2: Distribuição de Serviços da Arquitetura IDSA-IoT. Produzida e traduzida por ??).

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

3.4 Conclusão

Em conclusão, os trabalhos discutidos nesse Capítulo ~~tem~~ 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.

Apesar de já existirem propostas que estabelecem o estado da arte separadamente em cada um dos temas, ~~falta ainda uma abordagem que estabeleça uma união~~ entre o estado da arte em ~~algoritmos de detecção de~~ de novidade e o estado da arte em ~~processamento distribuído~~ de fluxos de dados, em especial para ~~o ambiente de computação em névoa focado em fluxos de dados relacionados a dispositivos .~~ focado em relacionados a

Capítulo 4

PROPOSTA E METODOLOGIA

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

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

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

~~Além dos aspectos~~ inerentes a *data stream mining*, são considerados na construção de um ~~sistema~~ que computa *data streams* a taxa de eventos ~~(itens atômicos de um)~~ gerados por cada produtor e o número de produtores nesse sistema, totalizando o volume de eventos ~~do sistema~~ . Volumes elevados ~~são dificilmente~~ dificilmente são computados em apenas um nó (e muito menos em um único núcleo processador) e por isso, esses sistemas ~~geralmente~~ são distribuídos.

Sistemas que utilizam ND para *data streams* gerados por dispositivos IoT devem utilizar algoritmos que considerem os desafios inerentes ~~de~~ a fluxos de dados (*Concept Evolution* e *Concept Drift*) para adequada detecção de novidades e, para tanto, requerem processamento

em arquiteturas que atendam os requisitos de volume de mensagens e latência de detecção. O algoritmo MINAS é adequado para esse caso, pois trata os desafios de *data stream mining*, porém não tem ainda implementação que atenda os requisitos de volume e latência, especialmente para aplicações IoT onde um ambiente de *fog computing* é atrativo.

Para preencher a lacuna de algoritmo de ND em ambiente *fog computing*, propõem-se então o sistema M-FOG, uma implementação do algoritmo MINAS sobre a plataforma *Apache Flink*, que considera distribuição em um ambiente de *fog computing*. O sistema M-FOG descrito neste documento foi refinado com os resultados dos experimentos descritos na Seção 4.3 e poderá ser revisado ao longo da pesquisa conforme os resultados de outros experimentos evidenciarem obstáculos ou oportunidades de melhoria.

4.1 Descrição da Implementação

Nesta Seção, apresenta-se o sistema M-FOG, objeto proposta deste trabalho. O sistema M-FOG é composto de três módulos principais e dois auxiliares. Os módulos principais implementam o algoritmo MINAS, sendo eles: módulo treinamento (*Training Module*), módulo classificador (*Classification Module*) e módulo detector de novidades (*Novelty Detection Module*). Dois módulos auxiliares são utilizados para avaliação do sistema M-FOG: módulo auxiliar *source* (fonte) e módulo auxiliar *sink* (sorvedouro, consumidor final). Os módulos e as interações entre eles são ilustradas na Figura 4.1.

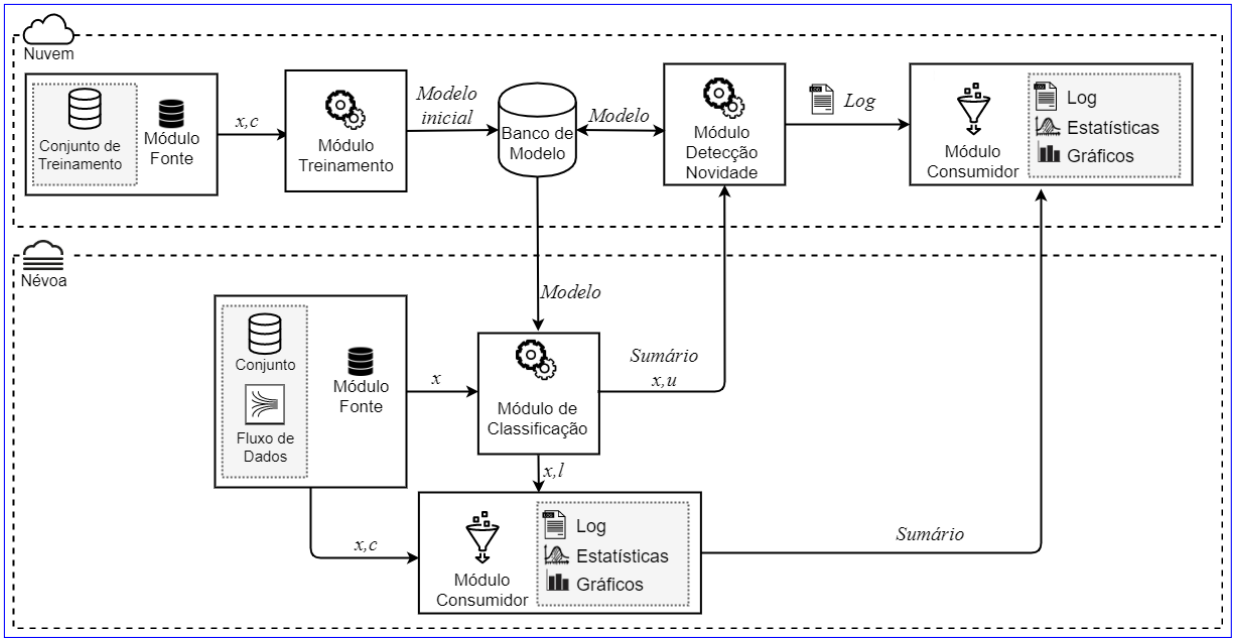

Figura 4.1: Arquitetura e fluxos de dados do sistema M-FOG.

A implementação do sistema M-FOG segue a arquitetura IDSA-IoT formalizada por **??**), discutida na Seção 3.3. A arquitetura IDSA-IoT estabelece que um serviço de captura e tratamento de dados é instalado na borda de uma rede local com dispositivos IoT. Na presente implementação, esse serviço de captura e tratamento é representado pelo módulo auxiliar *source*.

O módulo auxiliar *source* é dependente da fonte de dados, executando a transformação dos formatos dos *data sets* para um fluxo de exemplos (representado por x na Figura 4.1) compatível com o restante da implementação. Além de fornecer exemplos tratados para o módulo classificador, o módulo auxiliar *source* também fornece exemplos com a classe original (representado por x, c na Figura 4.1) ~~para o e para o~~ .

O módulo auxiliar *sink* é responsável por agregar todos resultados do sistema M-FOG e, juntamente com os valores do *data set* fornecidos pelo módulo auxiliar *source*, por computar as métricas de qualidade de classificação~~e~~ . Além disso, esse módulo também coleta e agrega métricas base para as ~~métricas~~ avaliação de escalabilidade e métricas de uso de recursos computacionais.

Os dados resultantes do serviço de captura e tratamento (representado no sistema M-FOG pelo módulo auxiliar *source*) são ingeridos pela aplicação no módulo classificador ~~,~~. A ingestão é feita por meio de ~~conexão TCP (*Transmission Control Protocol*)~~ um operador fonte, fornecida pela plataforma *Apache Flink*. Na plataforma, com o modelo de classificação disponível, os exemplos são classificados seguindo o algoritmo MINAS original discutido na Seção 2.5.1. A etiqueta atribuída pela classificação, ou meta-etiqueta de desconhecido, juntamente com o exemplo original (representado por x, l na Figura 4.1) são enviados para o módulo auxiliar *sink*. Além disso, se o exemplo não for classificado, o exemplo e a meta-etiqueta de desconhecido (representado por x, u na Figura 4.1) ~~é enviado~~ são enviados para o módulo detector de novidades. Outra comunicação é o envio ~~periódico do~~ das modificações ao sumário estatístico do modelo de classificação (representado por *Summary* na Figura 4.1) do módulo classificador para o módulo detector de novidades.

O módulo detector de novidades é responsável por executar o processo de detecção de novidade, atualizando o modelo de classificação, e ~~entrega do~~ entregar o novo modelo às instâncias do módulo classificador , através do serviço de armazenamento de modelo (*Model Store* na Figura 4.1). O módulo detector de novidades também envia meta-informações sobre o processo de detecção de novidade (representado por *Log* na Figura 4.1) para o módulo auxiliar *sink*.

O sistema M-FOG utiliza em seus módulos a distribuição oferecida pela plataforma *Apache Flink* como paralelização, ou seja, utiliza uma instância de trabalho (*job*) por dispositivo de

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

4.2 Metodologia de Avaliação e Resultados Esperados

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

O tratamento estabelecido das métricas de qualidade para *data stream mining* define que as métricas sejam extraídas de uma matriz de erro de classificação multi-classe $\mathbf{E}_n$ (Equação 4.3), adaptada para detecção de novidade. A matriz de erro é preenchida com o número de eventos da classe c_i classificados com etiqueta l_j até o instante n. A Equação 4.1 representa o conjunto de classes presentes nos eventos do fluxo até o instante n e a Equação 4.2 representa o conjunto de etiquetas atribuídas pelo classificador a eventos até o mesmo instante.

$$\mathbf{C}_n = \{c_1, c_2, \cdots, c_M\} \tag{4.1}$$

$$\mathbf{L}_n = \{l_1, l_2, \cdots, l_J\} \tag{4.2}$$

$$\mathbf{E}_n = \begin{pmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,J} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ e_{M,1} & e_{M,2} & \cdots & e_{M,J} \end{pmatrix} \tag{4.3}$$

As métricas de qualidade de classificação selecionadas para avaliar a implementação do sistema M-FOG ~~são~~ serão taxa de desconhecidos (*UnkR* na Equação 4.4) (**??**), acurácia média (*acc* na Equação 4.5) e Macro F-score (*Fscore* na Equação 4.9, também referido na literatura por F1M) (**????**). As métricas são extraídas para todos os exemplos classificados (instantes n) da respectiva matriz de erro $\mathbf{E}_n$.

$$UnkR_n = \frac{1}{M}\sum_{i=1}^{M}\frac{\#Unk_i}{\#ExC_i} \tag{4.4}$$

$$acc_n = \frac{1}{M}\sum_{i=1}^{M}\frac{tp_i + tn_i}{tp_i + fn_i + fp_i + tn_i} = \frac{1}{M}\sum_{i=1}^{M}\frac{\#Acc_i}{\#ExC_i} \tag{4.5}$$

$$Precision_n = \frac{1}{M}\sum_{i=1}^{M}\frac{tp_i}{tp_i + fp_i} \tag{4.6}$$

$$Recall_n = \frac{1}{M}\sum_{i=1}^{M}\frac{tp_i}{tp_i + fn_i} \tag{4.7}$$

$$Fscore\beta_n = (\beta^2 + 1)\cdot\frac{Precision\cdot Recall}{\beta^2\cdot Precision + Recall} \tag{4.8}$$

$$Fscore1_n = 2\cdot\frac{Precision\cdot Recall}{Precision + Recall} \tag{4.9}$$

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

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

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

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

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

c) Recursos computacionais utilizados (memória, tempo de processamento, operações de

leitura e escrita).

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

4.2.1 Ambiente de Teste

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

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

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

Faz parte também do ambiente de teste os conjuntos de dados (*data sets*) *KDD99* ~~e *Kyoto 2006+*~~

que foram selecionados por motivos distintos. O *data set Kyoto 2006+* é o foco deste trabalho, pois contém dados ainda representativos (até 2015) e as característica desejáveis de um conjunto de dados (realismo, validade, etiquetas previamente definidas, alta variabilidade, reprodutibilidade e disponibilidade pública) são atendidas. O *data set KDD99* é amplamente utilizado em trabalhos de detecção de anomalia e neste trabalho é utilizado somente para que o leitor possa comparar com outros trabalhos, pois não possui mais a característica de realismo, uma vez que foi construído em 1998.

[1]Disponível em ⟨http://portalcloud.ufscar.br/servicos⟩

Tabela 4.1: Sumário dos conjuntos de dados

Nome	Origem	Descrição	Acesso Público
KDD99	Captura de Fluxos de rede com ataques simulados	41 atributos, 23 classes, 4898431 instâncias, 709 MB	⟨https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html⟩
Kyoto 2006+ (??)	Captura de Fluxos de rede com HoneyPot	23 atributos, 3 classes, 22 GB (2015)	⟨https://www.takakura.com/Kyoto_data/new_data201704/⟩
ISCXTor2016 (??)	Origem	28 atributos,	⟨https://www.unb.ca/cic/datasets/tor.html⟩
ISCXVPN2016 (??)	Origem	28 atributos,	⟨https://www.unb.ca/cic/datasets/tor.html⟩
CICIDS2017	Captura de Fluxos de rede com ataques simulados com perfil de trafego de 25 usuários normais e de 6 perfis de ataques durante 5 dias (1º dia sem ataque)	80 atributos (CICFlowMeter), 15 classes, 2830751 instâncias 1.2GB	⟨https://www.unb.ca/cic/datasets/ids-2017.html⟩
Radial Basis Function (RBF) da biblioteca *Massive Online Analysis* (MOA) *4CRE-V2*	Origem	Descrição	⟨https://sites.google.com/site/nonstationaryarchive/home⟩

4.3 Resultados preliminares

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

4.3.1 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 ~~suportado~~ 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.

4.3.2 Implementação com *Apache Flink*

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

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

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

Esta implementação segue a arquitetura descrita na Seção 4.1 e as avaliações e resultados esperados descritos neste Capítulo 4 referem-se à implementação do sistema M-FOG na plataforma *Apache Flink*.

Capítulo 5

Considerações Finais

~~O~~ Este Capítulo ~~a seguir~~ resume o trabalho realizado até agora e estabelece os próximos passos até sua completude.

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 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~~ 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.

5.1 Cronograma

Nesta Seção apresentam-se as etapas previstas e sua distribuição temporal até o final deste trabalho de pesquisa.

A) Exame de Qualificação;

B) Desenvolvimento da aplicação;

C) Validação da aplicação em contraste com a implementação MINAS original:

- preparação e, se necessário, adaptação da implementação original e *data sets*;
- comparação e, se necessário, ajustes à implementação.

D) Experimentos com ~~*data sets*~~ e estratégias de distribuição em *fog*;

E) Submissão de artigos com resultados de (D)

F) Defesa da Dissertação.