UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA IMPLEMENTAÇÃO PARALELA DO ALGORITMO DE DETECÇÃO DE NOVIDADE EM STREAMS 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 PARALELA DO ALGORITMO DE DETECÇÃO DE NOVIDADE EM STREAMS 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).

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 intercomunicação pela Internet sem depender de interação com interface humano-computador tradicional. O número de dispositivos categorizados como IoT na década passada teve crescimento sem precedentes e, proporcionalmente, cresceu o volume de dados gerados por esses dispositivos.

A análise desses dados pode trazer novos conhecimentos e foi um tema frequentemente abordado por trabalhos de pesquisa na última década. No entanto, além dos dados de sensores e atuadores esses dispositivos, quando subvertidos, podem gerar outro tipo de tráfego, maligno à sociedade, como o gerado pela *botnet* mirai em 2016 (KAMBOURAKIS; KOLIAS; STAVROU, 2017). Nesse cenário são fatores que possibilitaram esses ataques: falta de controle sobre a origem do hardware e software embarcado nos dispositivos além das cruciais atualizações de segurança.

Com milhares de dispositivos em redes distantes gerando dados (diretamente ligados a sua função original e também metadados produzidos como subproduto) com volume e velocidade consideráveis formando fluxos contínuos de dados (*Data Stream* - DS) , técnicas de mineração de fluxos de dados (*Data Stream Mining*) são amplamente necessárias. 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 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 num serviço de detecção de intrusão.

Técnicas de *Data Stream Mining* envolvem mineração de dados (*Data Mining*), aprendizado

de máquina (*Machine Learning*) e recentemente Detecção de Novidades (*Novelty Detection* - ND). ND além de classificar em modelos conhecidos permite descobrir novos padrões e, consequentemente, atuar coerentemente mesmo em face a valores 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 em bancos de dados de ataques conhecidos.

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

1.1 Motivação

No contexto de detecção de novidades para fluxos de dados em *fog*, uma arquitetura recente proposta por Cassales et al. (2019), baseada no algoritmo de detecção de novidades em fluxo de dados MINAS (FARIA; CARVALHO; GAMA, 2016), mostra resultados promissores.

A arquitetura proposta foi avaliada com conjunto de dados (*data set*) *Kyoto 2006+* que coletou dados de 348 *Honeypots* (máquinas isoladas com diversos softwares com vulnerabilidades conhecidas expostas à Internet com propósito de atrair ataques) de 2006 até dezembro 2015. Com 24 atributos, 3 etiquetas atribuídas por detectores de intrusão comerciais de detecção de intrusão e uma etiqueta distinguindo o tráfego entre normal, ataque conhecido e ataque desconhecido (CASSALES et al., 2019). Contudo, esse algoritmo ainda não foi implementado e avaliado com paralelismo multi-processamento ou distribuído.

Outras propostas tratam do caso de grandes volumes e velocidades, como é o caso de Viegas et al. (2019) que apresenta o *BigFlow* no intuito de detectar intrusão em redes *10 Gigabit Ethernet*, que é um volume considerável atualmente impossível de ser processado em um único núcleo de processador (*single-threaded*). Essa implementação é 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*.

Além de apresentar uma implementação, Viegas et al. (2019) também apresenta o *data set MAWIFlow*. Esse conjunto é derivado do ponto de coleta F, localizado em um elo de

comunicação entre o Japão e os EUA (*Backbone*) com capacidade de 1 $Gbps$, diariamente 15 minutos são capturados desde 2006 sob supervisão de MAWI Working Group Traffic Archive (2020) (FONTUGNE et al., 2010). O conjunto *MAWIFlow* limita-se às coletas de 2016 (7.9 TB) e estratificado para 1% desse tamanho facilitando compartilhamento e avaliação por outros softwares. Esse conjunto contempla 158 atributos de nós e fluxos e etiquetado por Fontugne et al. (2010).

O sistema *BigFlow* é composto de dois estágios: extração de características (estatísticas de tráfego da rede) e aprendizado confiável de fluxo. O segundo estágio implementa um algoritmo de detecção de novidade utilizando classificadores já estabelecidos na biblioteca *Massive Online Analysis framework* (MOA) (BIFET et al., 2010) com adição de um módulo de verificação que armazena valores classificados com baixa confiança para serem manualmente avaliados por um especialista (VIEGAS et al., 2019). A escolha dessa abordagem não é nova e visa mitigar problemas como variação temporal de conceitos conhecidos (*Concept Drift*) e conceitos emergentes (*Concept Evolution*) (FARIA et al., 2016) com a redução de acurácia durante na avaliação inicial dos algoritmos tradicionais e devidamente mitigada com a atualização constante do modelo. Esses problemas são amplamente abordados e tratados em outros algoritmos como o MINAS (FARIA; CARVALHO; GAMA, 2016).

Os trabalhos de Cassales et al. (2019) e Viegas et al. (2019) abordam detecção de intrusão em redes utilizando algoritmos de ND em DS porém com perspectivas diferentes. O primeiro observa *IoT*, processamento em *fog*, baseia-se em um algoritmo genérico de detecção de novidade e o segundo observa *backbones*, processamento em *cloud*, implementa o próprio algoritmo de detecção de novidade. Essas diferenças deixam uma lacuna onde de um lado tem-se uma arquitetura mais adequada para *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 outra arquitetura (*cloud*) e com um algoritmo menos preparado para os desafios de detecção de novidades.

1.2 Objetivos

Com a avaliação inicial formulou-se a questão *"quais resultados podem ser esperados de um sistema multi-processado que implementa um algoritmo estado da arte de detecção de novidades em fluxo de dados?"* que engloba sucintamente os temas abordados nesse trabalho.

Propõem-se então a construção de uma aplicação que implemente o algoritmo MINAS de maneira escalável e distribuível para nós *multi-core* e a avaliação dessa implementação com ex-

perimentos baseados na literatura e conjunto de dados públicos relevantes para facilitar posterior comparação com outras implementações. O resultado esperado é uma implementação compatível em qualidade de classificação a seu original e com escalabilidade semelhante demonstrada pelo *BigFlow*.

Com foco no objetivo geral, alguns objetivos secundários são propostos:

- Identificar métricas de qualidade de classificação e métricas de escalabilidade encontradas na literatura;
- Avaliar plataformas de processamento distribuído de fluxos como:
  - *Apache Kafka* com *Python*;
  - *Apache Kafka Streams*;
  - *Apache Spark Streaming*;
  - *Apache Storm*;
  - *Apache Flink*;
- Implementar algoritmo MINAS sobre *Apache Flink*;
- Executar a implementação com *data sets* públicos relevantes;
- Validar, por meio de comparação com o algoritmo original, se a implementação gera os mesmos resultados quanto a classificação;
- Extrair métricas de escalabilidade por meio de experimentação;
- Avaliar estratégias de distribuição do algoritmo em *fog* como:
  - Detecção de novidade nas brodas;
  - Detecção de novidade na nuvem;
  - Detecção de novidade nas bordas e em nuvem;

  e seu o impacto na qualidade de classificação e volume computado.

1.3 Proposta Metodológica

Para cumprir os objetivos citados na Seção 1.2, foi identificado 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 e distribuído de fluxo de dados. O objetivo da revisão é o estabelecimento do estado da arte desses assuntos e para que alguns desses trabalhos sirvam para comparações e relacionamentos. Além disso, desses trabalhos extrai-se métricas de qualidade de classificação (por exemplo taxa de falso positivo e matriz de confusão) e métricas de escalabilidade (taxa de mensagens por segundo e escalabilidade vertical ou horizontal).

A revisão da literatura técnica foca em plataformas, ferramentas e técnicas para realizar a implementação proposta. Portanto, serão selecionadas plataformas de processamento distribuído de fluxos contínuos de dados e técnicas de aprendizado de máquina associadas a elas. Dessa revisão também são obtidas as 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 é desenvolvida uma aplicação multi-processada que, com base no algoritmo MINAS (FARIA; CARVALHO; GAMA, 2016), classifica e detecta novidades em DS. Os resultados de classificação dessa implementação equivalem-se aos resultados de classificação da implementação original, validando assim essa nova aplicação.

Nota: Não é separado, são ideias e métodos distisntos

Além disso, experimentos com a implementação e variações em *data sets* e cenários de distribuição em *fog* coletando as métricas de classificação e escalabilidade. Os resultados serão comparados entre si e com outros trabalhos.

E 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; Capítulo 5 discute os desafios e resultados preliminares encontrados durante o

desenvolvimento da aplicação; Capítulo 6 adiciona considerações gerais e apresenta o plano de trabalho e cronograma até a defesa.

Capítulo 2

FUNDAMENTOS CIENTÍFICOS E TECNOLÓGICOS

2.1 Computação em Nuvem, Borda e Névoa

2.1.1 Computação em Nuvem

A computação em nuvem (em inglês, *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 e rapidamente com o mínimo esforço de gerenciamento ou interação com o provedor de serviços (MELL; GRANCE, 2012). As principais características do *cloud computing* são:

- **Serviço sob Demanda:** o cliente pode provisionar ou liberar capacidades de computação (ex: tempo de servidor e armazenamento) conforme o necessário sem requer 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;
- **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 sobre as capacidades que foram consumidas.

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

- **Nuvem privada:** a infraestrutura da nuvem é provisionada e dedicada para um único cliente ou organização. Nesse modelo, o cliente gerencia e controla a infraestrutura, ou pode delegar essas tarefas a uma empresa terceira. 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, podendo estar dentro ou fora das instalações da empresa proprietária;
- **Nuvem pública:** a infraestrutura da nuvem 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 ou mais modelos de implantação de *cloud* (privada, pública e comunitária) que formam um 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 (em inglês, *edge computing*) refere-se às tecnologias que permitem que a computação seja executada na borda da rede. Define-se borda ou *edge* como qualquer recurso de computação e de rede ao longo do caminho entre as fontes de dados e os data centers da nuvem (SHI et al., 2016). Na borda, é possível fazer armazenamento, processamento e descarregamento de dados, assim como distribuir as requisições e entregar os serviços das nuvens aos usuários. Shi et al. (2016) ressalta que essas capacidades dentre outras dos nós da borda (*edge nodes*) possibilita 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 acrescentar que o *edge computing* pode aumentar a privacidade dos dados uma vez que o dado pode ser processado no próprio dispositivo final.

A computação de borda tenta trazer a computação mais próxima das fontes de dados, como mostra a Figura 2.1. 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: nós de borda fazendo análise de vídeo; diminuir a latência da nuvem para sistemas críticos; descarregar a nuvem de parte da computação, pois a borda também tem capacidade de processamento; privacidade dos dados produzidos por *smart homes* e sensores IoT, pois dados sensíveis não precisam ir para a nuvem, liberando também cargas de dados na rede; processamento distribuído nos nós da borda de dados provenientes de milhões de sensores em cidades inteligentes.

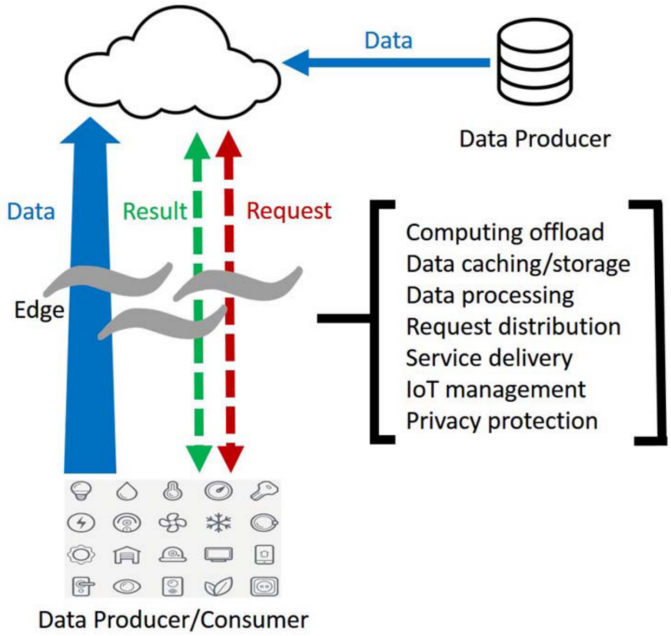


Figura 2.1: Paradigma de *Edge Computing* (SHI et al., 2016).

2.1.3 Computação em Névoa

Dastjerdi e Buyya (2016) e IEEE Communications Society (2018) mencionam que a enorme massa de dados gerados por ambientes IoT pode ser processada pela nuvem, entretanto a latência produzida pela transferência desses dados para a nuvem e o retorno do resultado não são tolerados 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* para processamento

e armazenamento pode ser ineficiente e não escalável devido a saturação de dados na rede. O *edge computing* foi proposto para trazer o processamento e armazenamento para os dispositivos de borda tentando solucionar esses problemas, porém os dispositivos de borda não podem lidar com várias aplicações IoT competindo pelos seus recursos limitados, o que poderia causar a contenção dos recursos e aumento na latência do processamento (DASTJERDI; BUYYA, 2016). Portanto, para solucionar estas questões de latência e capacidade limita dos dispositivos de borda, a computação em névoa foi proposta.

A computação em névoa (em inglês, *fog computing*) é um paradigma que traz as capacidades de computação, armazenamento e rede para próximo das fontes dados e dos dispositivos finais, mas não necessariamente localizados na borda (BONOMI et al., 2012) (DASTJERDI; BUYYA, 2016) (IEEE Communications Society, 2018). Esse tipo de computação evita a contenção dos recursos dos dispositivos mais próximos às fontes de dados fornecendo-os capacidade de nuvem e coordenando-os geograficamente distribuídos. Bonomi et al. (2012) e Dastjerdi e Buyya (2016) veem o *fog computing* como complementar da *edge computing*, podendo a computação em névoa aproveitar os recursos da nuvem e da borda. IEEE Communications Society (2018) considera que a principal diferença entre esses dois tipos de computação está no número de camadas. Enquanto 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 Bonomi et al. (2012) e Dastjerdi e Buyya (2016), as principais características da computação em névoa são:

- **Mobilidade:** é essencial que as aplicações *fog* sejam capazes de se comunicar com dispositivos móveis, por exemplo, utilizando protocolos que considerem a mobilidade dos nós;
- **Heterogeneidade:** os nós nesse tipo de paradigma possuem configurações e formatos diferentes e podem estar implantados em ambientes distintos;
- **Baixa Latência:** 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 milhares de sensores e dispositivos distribuídos geograficamente e a consciência da localização deles (em inglês, *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 inscrevem a informação no fluxo de dados, a informação é processada, e os *insights* extraídos são 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 a partir das informações deles é possível alertar os ocupantes do prédio em algum caso de emergência.

2.2 Mineração de Dados e Fluxo de Dados

A Mineração de Dados é o processo de descoberta de padrões em conjuntos de dados utilizando métodos derivados de aprendizagem de máquina, estatística e banco de dados (GABER; ZASLAVSKY; KRISHNASWAMY, 2005). Um caso de mineração de dados é *Big Data* onde o conjunto de dados não pode ser processado em um tempo viável devido a limitações como armazenado na 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 produzidos a uma taxa muito alta, associadas ao tempo real, ilimitados, que excede recursos de armazenamento e comunicação (GABER; ZASLAVSKY; KRISHNASWAMY, 2005). Modelos de mineração de fluxo de dados atendem a esses desafios utilizando restrições como apenas uma leitura do conjunto de dados e complexidade de processamento menor que linear (GAMA; RODRIGUES, 2007; GABER; ZASLAVSKY; KRISHNASWAMY, 2005).

Nota: falta abordar processamento paralelo e distribuído

2.3 Arquiteturas e Plataformas de Processamento de Fluxos

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

O volume e velocidade de dados em um *Data Stream* leva a 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 esses problemas foram propostas arquiteturas de processamento de fluxo de dados distribuído como arquitetura *lambda* e *kappa* além de diversas plataformas tanto de *Big Data* com características de tempo real como especializadas em fluxo de dados.

Arquitetura *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 aquivos 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), 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 a camada de velocidade complementa os dados das visões com dados diretamente do conjunto mestre em tempo real diretamente para a camada de serviço (MARZ; WARREN, 2015).

A Figura 2.2 ilustra uma implementação da arquitetura *Lambda* onde a *Apache Kafka*, *Apache Storm*, *Apache Hadoop* implementam o conjunto mestre, camada de velocidade e camada de lotes respectivamente.

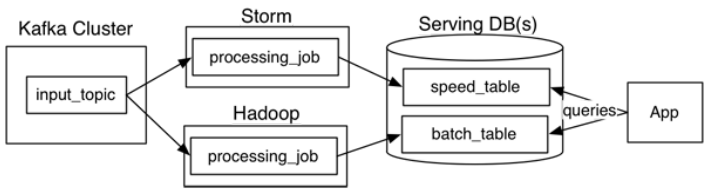


Figura 2.2: Arquitetura *Lambda* com detalhes práticos (KREPS, 2014).

No entanto, observações práticas de Kreps (2014) mostram que como o sistema de fila de mensagens (no exemplo *Apache Kafka*) fonte do conjunto de dados mestre por lidar com diversas

fontes, garantir tolerância a falhas, replicação e armazenamento de longo prazo, poderia ser usado como fonte de verdade em conjunto com a arquitetura totalmente incremental (tradicional) substituindo as camadas de lotes e velocidade e o sistema de aquivos distribuído simplificando a aplicação de três implementações para duas eliminando a repetição de tarefas executadas pelas camadas de lotes e velocidade que produziam o mesmo resultado.

Essa arquitetura que limita-se ao processamento de *Stream* apenas *on-line* destila outras observações quanto a complexidade temporal do sistema completo de que se uma aplicação não processa os dados em tempo menor que linear, dados serão perdidos ou a complexidade espacial são será satisfeita, em resumo, a aplicação sempre deve ser mais rápida que o *Stream* (MARZ; WARREN, 2015).

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

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

Em prática um *cluster* tem centenas 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 nó mestre. Essa implementação tem esquema geral de processamento em lotes que não atende o requisito de baixa latência porém suas vantagens sobrepunham as desvantagens. *MapReduce* é uma das principais influencias na criação da arquitetura *lambda* (MARZ; WARREN, 2015).

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

Apache Spark, analogamente ao *Hadoop*, é um *framework* para construção de sistemas de computação distribuída em *cluster* com garantias de tolerância a falhas. No entanto, o modelo de processamento diverge significativamente do tradicional *MapReduce* utilizando em lugar do HDFS um multi-set de apenas leitura distribuído (*Resilient Distributed Dataset* - RDD) com um

escalonador de trabalhos representados por grafos acíclicos direcionados (*directed acyclic graph* - DAG), otimizador de consultas e motor de execução (Apache Spark, 2020).

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

Nota: aqui termnia ——————

Além de

Foi desenvolvido no AMPLab da Universidade da Califórnia[2] e posteriormente doado para a Apache Software Foundation[3] que o mantém desde então.

Spark provê uma interface para programação de clusters com paralelismo e tolerância a falhas.

Apache Spark (ZAHARIA et al., 2010) é um (execução em computadores não confiáveis) utilizando como premissas: paralelização e localidade de dados, como

api em Python (dataframe de pandas)

2.4 Apache Flink

Breve descrição do Flink (como esse vai ser usado, precisa explicar um pouco melhor - 2 paginas pelo menos):
- arquitetura
- modelo de programacao
- 1 pequeno exemplo de codigo explicando

(LOPEZ, 2018)

2.5 Detecção de Novidade

Novelty Detection

breve descrição do que sao algoritmos para DN

(COSTA, 2019)

(PERNER, 2007)(GAMA, 2010)

ECSMiner (MASUD et al., 2011)

ver se tem algum survey e citar

2.6 O algoritmo MINAS

Breve descrição do MINAS (FARIA; CARVALHO; GAMA, 2016).

ver paper da profa. Elaine

Capítulo 3

TRABALHOS RELACIONADOS

Esse 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.

3.1 Algoritmo MINAS e Algoritmos Derivados

O algoritmo MINAS, como já foi discutido, 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 sem intervenção de especialista (FARIA; CARVALHO; GAMA, 2016). Neste trabalho, consideram-se algoritmos derivados aqueles apresentados em trabalhos publicados após 2016 que estendem a implementação original seguindo sua estrutura básica.

Algoritmo FuzzyND

O algoritmo FuzzyND, derivado do MINAS é proposto por Da Silva et al. (2018). FuzzyND incrementa o algoritmo inicial aplicando à ele teorias de conjuntos *fuzzy* pela modificação da representação dos *clusters*. A modificação afeta os métodos de construção de *clusters*, classificação de exemplos e detecção de novidades de acordo com a nova representação.

A avaliação do algoritmo FuzzyND é feita por meio de experimentos usando 3 *data sets* sintéticos (*MOA3*, *RBF*, *SynEDC*) e comparação com o MINAS. O método de avaliação utilizado baseia-se na matriz de confusão incremental descrita por Faria et al. (2016) extraindo dessa matriz duas métricas: acurácia (*Macro F-Score*) (SOKOLOVA; LAPALME, 2009) e taxa de desconhecidos (*UnkR*) (FARIA; CARVALHO; GAMA, 2016). Em geral o algoritmo FuzzyND detecta melhor novidades e, consequentemente, é mais robusto à valores atípicos (*outlier*) porém

perde a capaciade de reconhecer padrões recorrentes.

Algoritmos MINAS-LC e MINAS-BR

O algoritmo MINAS-LC é proposto por Costa (2019) e trata classificação multi-rótulo porém não trata evoluções de conceito (novas classes). AS alterações fundamentais 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 transforma-lo de um conjunto multi-rotulo 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 Costa et al. (2019), estende o algoritmo original para que classifique um exemplo com uma ou mais etiquetas usando a transformação *Binary Relevance* propondo o 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* (VIJAYA; MURTY; SUBRAMANIAN, 2004).

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 (BIFET et al., 2010) e *4CRE-V2* [1] gerados pelo método *Radial Basis Function* (SOUZA et al., 2015). MINAS-BR é comparado com 7 algoritmos da literatura também disponíveis na ferramenta MOA (BIFET et al., 2010), diferente da avaliação do FuzzyND que compara diretamente com MINAS. Os 7 algoritmos são divididos em dois grupos: 3 com acesso às etiquetas corretas para atualização do modelo e com a técnica ADWIN (*ADaptive WINdowing*) para detectar mudanças de conceito (*Concept Drift*); 4 algoritmos sem acesso às etiquetas corretas, ou seja, sem *feedback* externo, mesma condição do MINAS-BR.

A avaliação elencada por Costa et al. (2019) 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 Faria et al. (2016) é apresentada com alterações para exemplos multi-rótulo. A estratégia é executada na fase de classificação seguindo as regras: 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

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

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. Após associação entre padrões de novidade e classes novidade é possível calcular métricas tradicionais.

As métricas utilizadas por Costa et al. (2019) 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 Seção tem em comum muito além do algoritmo base, tem também 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 de 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, todas são extremamente semelhantes: 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 (Da Silva et al., 2018; SILVA, 2018; COSTA et al., 2019)[2].

3.2 AnyNovel

Nota: Incompleto

Nota: também é da mesma classe do minas porém Cassales et al. (2019) destaca um desempenho inferior para o data set *testado.*

[2]Propostas de modificação do algoritmo MINAS estão longe de serem exauridas. Não cabe ao presente trabalho expandir e validar conceitos de aprendizagem de maquina 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 para tratar *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 (ruido ou não representativo) ou válido, podendo conter uma ou mais classes e, se conter mais que uma classe corte em grupos menores até conter somente uma classe; d) ainda considerando interação com especialista, a possibilidade dele injetar um exemplo não pertencer à uma classe, ou seja, marcar o exemplo como não pertencente à uma classe para manter ele 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 à classes distintas e tratamento adequado.

Abdallah et al. (2016)

3.3 Catraca Lopez2018

Nota: Incompleto

(LOPEZ, 2018)

3.4 BigFlow

Nota: Incompleto

BigFlow destaca em sua secção 2 (backgroud) o processamento de streams [18, 19], a preferencia de NIDS por anomalia em contraste aos NIDS por assinatura [30, 31, 32], a variabilidade e evolução dos padrões de tráfego em redes de propósito geral [9, 11, 20], a necessidade de atualização regular do modelo classificador [8, 9, 10, 20] e o tratamento de eventos onde a confiança resultante da classificação é baixa [9, 12, 13].

Também destaca em sua secção 3 (MAWIFlow) que data sets adequados para NIDS são poucos devido o conjunto de qualidades que os mesmos devem atender como realismo, validade, etiquetamento, grande variabilidade e reprodutividade (disponibilidade pública) [8, 9, 10, 17, 38].

Para avaliar o desempenho de NIDS o data set MAWIFlow é proposto. Originário do '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 inicio em 2006 continuamente até hoje, anonimizados [22], etiquetados por MAWILab [8]. Desse data set original apenas os eventos de 2016 são utilizados e desses 158 atributos são extraídas resultando em 7.9 TB de captura de pacotes. Além disso, os dados são estratificados [24] 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 as qualidades anteriormente mencionadas.

Com o data set MAWIFlow original e reduzido foram avaliados quatro classificadores [42, 43, 44, 45] da literatura em dois modos de operação quanto seus dados de treinamento (ambos contendo uma semana de captura) o primeiro usando somente a primeira semana do ano e as demais como teste e o segundo modo usando a semana anterior como treinamento e a seguinte como teste. Demostrando, com 62 atributos, que a qualidade da classificação retrai com o tempo quando não há atualização frequente do modelo classificador.

Nota: Falar do modelo de distribuição.

3.5 Arquitetura IDSA-IOT

3.6 Conjuntos de Dados e Referência de Desempenho para Detecção de Anomalia

The Numenta Anomaly Benchmark

Nota: concluir com um gap

Nota: Discutir que o data set não é de borda de uma rede, portanto não tem relevância para fog. Também discutir a classificação dos fluxos por endpoint exarcerbando assim a distinção na fog com o efeito de particionamento dos dados. Ou seja, um nó só vê e classifica os próprios dados.

Capítulo 4

PROPOSTA

Nota: Reestruturar: remember, cenário (iot, fog, stream), problema (4.1, ND em fog, terminar com minas e cassales), solução (4.2, apresetnação, resumo M-FOG, metodologia)

Esse Capítulo apresenta a descrição de uma implementação paralela do algoritmo de detecção de novidades em fluxos de dados contínuos (*Novelty Detection in Data Streams*) MINAS inserido no cenário de processamento distribuído no ambiente de computação em névoa (*fog computing*) com ênfase nos fluxos gerados por dispositivos de Internet das Coisas (*IoT*) doravante referido por M-FOG ou "a presente implementação". Os requisitos dessa implementação são derivados das lacuna discutidas no Capítulo 3.

Nota: resumo e retomada do objetivo

Uma das lacunas é a falta de uma implementação do algoritmo MINAS mais próxima de aplicações reais, ou seja: que trate de fluxos realistas em contraste aos conjuntos de dados (*data sets*) sintéticos; que considere restrições computacionais tanto teóricas que são características de processamento de fluxos contínuos e ilimitados como restrições práticas de ambientes diversos como computação pessoal (*Desktop Computing*), computação em nuvem (*cloud computing*) e computação em névoa (*fog computing*).

Outras lacunas são encontrada na implementação de sistemas de que dependem de algoritmos de detecção de novidades. Nessas implementações nota-se a falta de algoritmos que tenham as considerações teóricas inerentes da variabilidade de dados em fluxos contínuos de dados como deriva e evolução de conceito (*Concept Drift* e *Concept Evolution*). Por outro lado, outros sistemas consideram a variabilidade porém o mecanismo de atualização do modelo assume que as etiquetas reais estarão disponíveis após classificação (*feedback*) imediatamente ou com atraso

predizível. A suposição de *feedback* em fluxos reais também não é realista pois a presença de um especialista que classifique corretamente os exemplos solicitados é, na melhor das hipóteses, eventual e não contínua contrastando com a natureza do fluxo de dados.

4.1 Cenário

A presente Seção descreve o cenário base para a proposta de implementação, expondo características, desafios e hipóteses que a implementação deve atender.

A Internet das Coisas *IoT* é composta por milhares de dispositivos distribuídos geograficamente conectados à Internet. Com capacidades diversas como sensores e atuadores, esses dispositivos dependem de conexões para executarem suas funções, enviando e recebendo informações trocadas com outros dispositivos ou aplicações centrais hospedadas em ambiente *cloud computing*. A a dependência de comunicação é uma restrição que tem diversas implicações como confiabilidade de um sistema e, dependendo da conexão limita ou impede funcionalidades comuns em outros ambientes. Além das restrições, devido aos encargos atribuídos à manutenção da comunicação, o envio e recebimento de dados em dispositivos deve ser minimizado.

Mesmo com as restrições e desafios, a implementação de dispositivos ainda é vantajosa, pois o valor sobrepõe o custo. Esse valor origina de várias fontes como automação, redução de risco e coleta de dados granulares que possibilitam análises antes impraticáveis. A coleta de dados para análise contínua é um dos principais fatores na implantação de um sistema IoT levando a geração de diversos fluxos com propriedades e processos igualmente diversos.

Nota: Falta: fog, processamento distribuído de streams, detecção de novidade

4.2 Descrição da Implementação

Nesta Seção apresenta M-FOG, proposta deste trabalho. M-FOG é composto pelos módulos *offline*, classificador e detector. Para correta avaliação M-FOG é acompanhado de dois módulos auxiliares: módulo fonte (*source*) e módulo sorvedouro (*sink*).

notamétodos para alcançar os objetivos
o que fazer e como fazer, resultados esperados

A implementação proposta segue a arquitetura proposta por Cassales et al. (2019) onde na borda de uma rede local com dispositivos *IoT* um serviço de captura e tratamento de dados é instalado. Na presente implementação esse serviço de captura e tratamento é representado pelo

módulo *source*. O módulo *source* é dependente da fonte de dados, executando a transformação dos formatos dos *data sets* para um fluxo de dados compatível com o restante da implementação. Além de fornecer dados tratados para M-FOG, o módulo *source* também fornece dados para o módulo *sink* e *offline*.

O módulo *sink* é responsável por agregar todos resultados do M-FOG e, juntamente com os valores do *data set* fornecidos pelo módulo *source*, computar as métricas de qualidade de classificação e métricas base para as métricas de escalabilidade e métricas de recursos computacionais.

Os dados resultantes da captura e tratamento são ingeridos pela aplicação no módulo classificador por meio de conexão TCP (*Transmission Control Protocol*) 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 2.6. A etiqueta atribuída pela classificação, ou meta-etiqueta de desconhecido, juntamente com o exemplo original são enviados para o módulo *sink*. Além disso, se o exemplo não for classificado, o mesmo é enviado para o módulo detector.

O módulo detector é responsável por executar o processo de detecção de novidade, atualizando o modelo de classificação, e entrega do novo modelo ao módulo classificador. Este módulo também envia meta-informações sobre o processo de detecção de novidade para o módulo *sink*.

4.3 Resultados Esperados

Da implementação M-FOG é prevista execução de experimentos com *data sets* diversos, em especial os *data sets* reais que contém evolução de conceitos. Os resultados desses experimentos são válidos se contém as seguintes métricas para o M-FOG: 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) e para o minas somente as métricas de qualidade de classificação.

Capítulo 5

IMPLEMENTAÇÃO E TESTES

Nota: usar coeficiente d-intra vs d-extra grupo para determinar K de cada label

5.1 Descrição da Implementação

Nota: - offline, online, ND, Clustering
- observação/Considerações de paralelização
- Notas sobre implementação Python/Kafka/Minas (não escala como esperado)
- Dificuldade no processamento distribuido em Flink.
- complexidade bigO (?)

5.2 Ambiente de Teste

Para testar e demonstrar essa implementação um cenário de aplicação é construído onde seria vantajoso distribuir o processamento segundo o modelo *fog*. Alguns cenários de exemplo são casos onde deve-se tomar ação caso uma classe ou anomalia seja detectada

Nota: - detecção de intrusão
- Arquitetura guilherme (dispositivos pequenos vs cloud)
- Descrever a arquitetura IDS-IoT do paper do Guilherme citeCassales2019a
- dataset kyoto não está disponível
- dataset kdd99
- BigFlow com dataset atual e maior

5.3 Experimentos e Resultados

Nota: - gráficos, tempos, tabelas...

- análises e comentários

- Mostrar alguma implementação já feita e que esteja funcionando minimamente

Mostrar resultados mesmo que sejam bem simples e básicos, apenas para demonstrar que vc domina o ambiente e as ferramentas e que está apto a avançar no trabalho

Nota: discussão de 2020-02-01:

passos feitos/a fazer

1. Entender Minas

2. Analisar/descrever data set KDD

3. Notas sobre implementação Python/Kafka/Minas (não escala como esperado)

4. BigFlow (data set mais novo, usa flink)

5. Plataforma Flink (processamento distribuído)

Proposta

6. Implementar minas em Scala/Flink

7. Testar com data sets KDD e BigFlow

8. Validar/Comparar métricas com seus trabalhos correspondentes

Nota: lendo Quali Casssales:

- Descrição do hardware utilizado pode conter:

- Arch, OS, Kernel,

- CPU (core, thread, freq),

- RAM (total/free size, freq),

- Disk (total/free size, seq RW, rand RW),

- Net IO between nodes (direct crossover, switched, wireless, to cloud) (bandwidth, latency).

essas métricas permitem relacionar trade-offs para as questões de fog: Processar em node, edge ou cloud?

Nota: Provavelmente vou retirar o kafka da jogada em node/edge, deixando apenas em cloud.

Capítulo 6

Considerações Finais

O 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 enfasê 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 fluxos de dispositivos da Internet das Coisas (*IoT*) em ambiente de computação em névoa (*fog computing*).

O objeto central desse trabalho (M-FOG) implementa o 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 adequado para cada tarefa. A avaliação do M-FOG é feita por meio de métricas de qualidade de classificação e métricas de escalabilidade.

Nota: deixar claro que é futuro, voz singular, 3ª pessoa
metodologia do restante

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

Dessa forma contribuímos com uma adição de uma ferramenta para os interessados em construir sistemas que dependem de detecção de novidades para seu bom funcionamento, especialmente os baseados em dispositivos *IoT* e/ou com fluxos contínuos que tradicionalmente

sofrem com os ônus de latência e largura de banda na comunicação dos resultados da borda para a nuvem onde são normalmente executadas as técnicas de mineração usuais.

6.1 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;

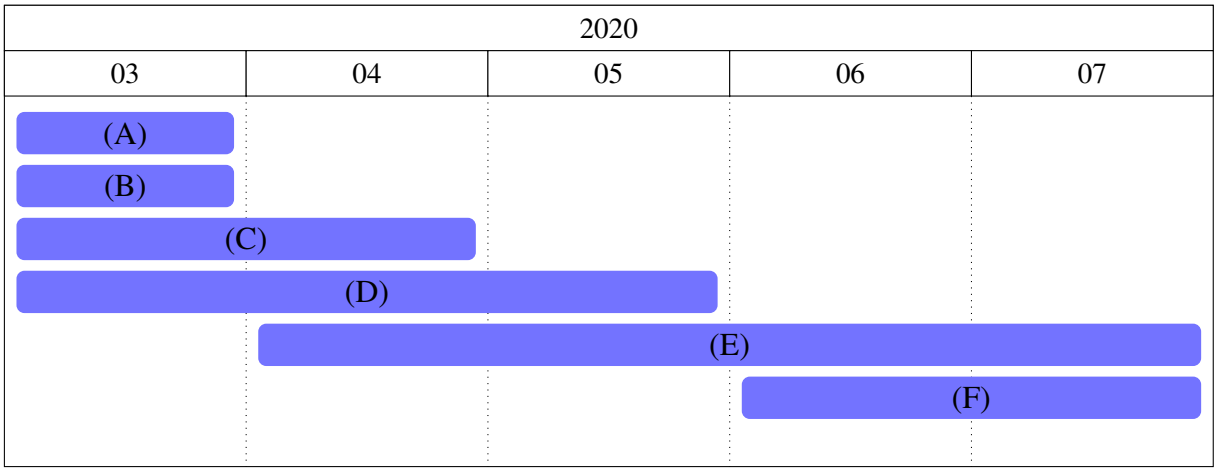
C) Validação da aplicação em contraste com a implementação MINAS original:

- 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 Disertação.



REFERÊNCIAS

ABDALLAH, Z. S.; GABER, M. M.; SRINIVASAN, B.; KRISHNASWAMY, S. Anynovel: detection of novel concepts in evolving data streams. *Evolving Systems*, Springer, v. 7, n. 2, p. 73–93, 2016.

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

Apache Spark. *Apache Spark^{TM} - Unified Analytics Engine for Big Data*. 2020. Disponível em: ⟨https://spark.apache.org/⟩.

BIFET, A.; HOLMES, G.; KIRKBY, R.; PFAHRINGER, B. MOA: massive online analysis. *J. Mach. Learn. Res.*, v. 11, p. 1601–1604, 2010. Disponível em: ⟨http://portal.acm.org/citation.cfm?id=1859903⟩.

BONOMI, F.; MILITO, R.; ZHU, J.; ADDEPALLI, S. Fog computing and its role in the internet of things. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. [s.n.], 2012. p. 13–16. ISBN 9781450315197. Disponível em: ⟨http://www.lispmob.org⟩.

CASSALES, G. W.; SENGER, H.; DE FARIA, E. R.; BIFET, A. IDSA-IoT: An Intrusion Detection System Architecture for IoT Networks. In: *2019 IEEE Symposium on Computers and Communications (ISCC)*. [s.n.], 2019. p. 1–7. ISBN 978-1-7281-2999-0. ISSN 1530-1346. Disponível em: ⟨https://ieeexplore.ieee.org/document/8969609/⟩.

COSTA, J. D. *DETECÇÃO DE NOVIDADE EM FLUXOS CONTÍNUOS DE DADOS MULTIRRÓTULO*. 127 p. Tese (Master) — UNIVERSIDADE FEDERAL DE SÃO CARLOS, 2019. Disponível em: ⟨https://repositorio.ufscar.br/handle/ufscar/12197⟩.

COSTA, J. D.; FARIA, E. R.; SILVA, J. A.; GAMA, J.; CERRI, R. Novelty detection for multi-label stream classification. *Proceedings - 2019 Brazilian Conference on Intelligent Systems, BRACIS 2019*, n. 8, p. 144–149, 2019.

Da Silva, T. P.; SCHICK, L.; De Abreu Lopes, P.; De Arruda Camargo, H. A fuzzy multiclass novelty detector for data streams. *IEEE International Conference on Fuzzy Systems*, IEEE, v. 2018-July, p. 1–8, 2018. ISSN 10987584.

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

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

FARIA, E. R.; GONÇALVES, I. J. C. R.; CARVALHO, A. C. P. L. F. de; GAMA, J. Novelty detection in data streams. *Artificial Intelligence Review*, Springer, v. 45, n. 2, p. 235–269, Feb 2016. ISSN 1573-7462. Disponível em: ⟨https://doi.org/10.1007/s10462-015-9444-8⟩.

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

FONTUGNE, R.; BORGNAT, P.; ABRY, P.; FUKUDA, K. MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking. In: *ACM CoNEXT '10*. Philadelphia, PA: [s.n.], 2010. p. 1–12.

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

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

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

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

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

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

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

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

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

SHI, W.; CAO, J.; ZHANG, Q.; LI, Y.; XU, L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, Institute of Electrical and Electronics Engineers Inc., v. 3, n. 5, p. 637–646, oct 2016. ISSN 23274662. Disponível em: ⟨https://ieeexplore.ieee.org/abstract/document/7488250⟩.

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

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

SOUZA, V. M.; SILVA, D. F.; GAMA, J.; BATISTA, G. E. Data stream classification guided by clustering on nonstationary environments and extreme verification latency. In: SIAM. *Proceedings of the 2015 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics Publications, 2015. p. 873–881. ISBN 9781510811522. Disponível em: ⟨https://doi.org/10.1137/1.9781611974010.98⟩.

VIEGAS, E.; SANTIN, A.; BESSANI, A.; NEVES, N. Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. *Future Generation Computer Systems*, v. 93, p. 473 – 485, 2019. ISSN 0167-739X. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0167739X18307635⟩.

VIJAYA, P.; MURTY, M. N.; SUBRAMANIAN, D. Leaders–subleaders: An efficient hierarchical clustering algorithm for large data sets. *Pattern Recognition Letters*, Elsevier, v. 25, n. 4, p. 505 – 513, mar 2004. ISSN 0167-8655. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0167865503002824⟩.

ZAHARIA, M.; CHOWDHURY, M.; FRANKLIN, M. J.; SHENKER, S. *Spark: Cluster Computing with Working Sets*. [S.l.], 2010. v. 10, n. 10-10, 95 p.