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. 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 possibilitam a subversão desses dispositivos: 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 classificar novos padrões e, consequentemente, atuar corretamente mesmo em face a 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 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, 2015), mostra resultados promissores. A arquitetura proposta foi avaliada com conjunto de dados (*data set*) *Kyoto 2006+* composto de dados coletados de 348 *Honeypots* (máquinas isoladas com diversos softwares com vulnerabilidades conhecidas expostas à Internet com propósito de atrair ataques) de 2006 até dezembro 2015. O *data set Kyoto 2006+* contém 24 atributos, 3 etiquetas atribuídas por detectores de intrusão comerciais e uma etiqueta distinguindo o tráfego entre normal, ataque conhecido e ataque desconhecido (CASSALES et al., 2019). Contudo, o algoritmo MINAS ainda não foi implementado e avaliado com paralelismo multi-processamento ou 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*.

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 e processamento em *fog*, baseia-se em um algoritmo genérico de detecção de novidade. O segundo trabalho observa *backbones* e 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

Como estabelecido na Seção 1.1, a lacuna no estado da arte observada é constituída da ausência de uma implementação de algoritmo de detecção de novidade que considere o ambiente de computação em névoa aplicada à detecção de intrusão.

Seguindo os trabalhos do Grupo de Sistemas Distribuídos e Redes (GSDR),

Nota: Citar que o MINAS já foi testado para detecção de intrusão [Guilherme2019]. Para a aplicação em questão

Nota: Falar do resultado bem resumuidamente

Nota: Porem a implementação não distribuída.

Propõem-se então 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 e conjunto de dados públicos relevantes. O resultado esperado é uma implementação compatível em qualidade de classificação ao algoritmo MINAS e passível de ser distribuído em um ambiente de computação em névoa.

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

Nota: citar ferramentas e a escolha só depois do python e kafka

Nota: entre flink e spark, outro grupo de pesquisa já está explorando spark

- 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 e os conjuntos de dados associados;
- Avaliar o desempenho da implementação em ambiente de computação em névoa.

Nota: estudar o algoritmo
fazer uma implementação
pegar datasets relevantes
comparar a corretude com o sequencial
avaliar desempenho e escalabilidade

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 DS e técnicas de aprendizado de máquina associadas a elas. Dessa revisão também são 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 é desenvolvida uma aplicação na plataforma escolhida que, com base no algoritmo MINAS (FARIA; CARVALHO; GAMA, 2015), classifica e detecta novidades em DS. Também nesse passo a implementação é 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 distidos 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.

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 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 processamento 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 acrescenta 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: análise de vídeo; redução de latência da nuvem para sistemas críticos; descarregar a nuvem de parte da computação; privacidade dos dados produzidos mantendo eles fora de ambientes públicos; redução das cargas de dados na rede; processamento distribuído de sensoriamento massivo em cidades inteligentes (SHI et al., 2016).

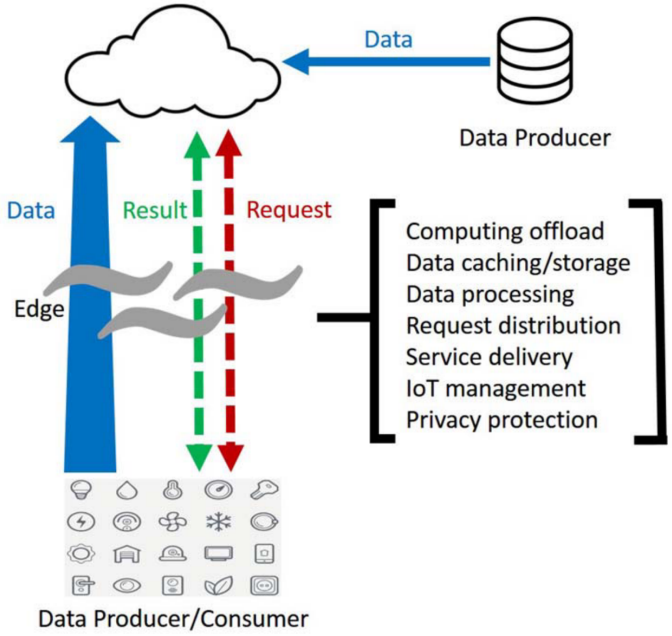


Figura 2.1: Paradigma de *Edge Computing* (SHI et al., 2016).

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 (*fog computing*) é um paradigma que distribui as capacidades de computação, armazenamento e rede entre os nós próximos das fontes dados e dos dispositivos finais, mas não necessariamente localizados na borda, dando a esses nós características de uma nuvem (BONOMI et al., 2012; DASTJERDI; BUYYA, 2016; IEEE Communications Society, 2018). Esse tipo de computação evita a sobrecarga dos dispositivos de borda. Bonomi et al. (2012) e Dastjerdi e Buyya (2016) 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 (*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 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 o *Big Data* onde o conjunto de dados não pode ser processado em um 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 produzidos a uma taxa muito alta, associadas ao tempo real, ilimitados, que excede recursos de armazenamento (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.

A 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 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). 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.

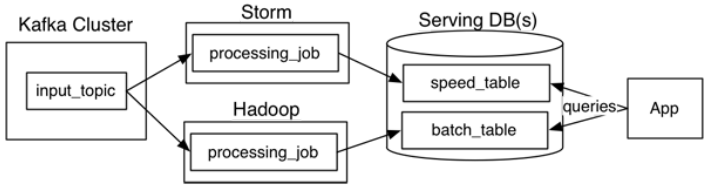


Figura 2.2: Arquitetura *Lambda* com detalhes práticos (KREPS, 2014).

Em contraposição à arquitetura *Lambda*, observações práticas de Kreps (2014) mostram que o sistema de fila de mensagens (no exemplo *Apache Kafka*) já traz as garantias de escalabilidade, consistência, tolerância a falhas, replicação e armazenamento de longo prazo. Com isso, Kreps (2014) propõe que as camadas de lotes e velocidade sejam unificadas em uma camada de processamento de fluxo, cujos resultados sejam entregues continuamente para a camada de serviço através de um SGBD, definindo assim a 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 *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. 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* recebe uma chave e um conjunto de valores gerado pelo agrupamento de pares com essa mesma chave (DEAN; GHEMAWAT, 2004).

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

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

Apache Spark, analogamente ao *Hadoop*, é um *framework* para construção de sistemas de computação distribuída em *cluster* com garantias de tolerância a falhas. No entanto, o modelo de processamento diverge significativamente do tradicional *MapReduce* utilizando em lugar do HDFS um multiconjunto 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, o 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.

Uma das extensões do *Apache Spark* é o *Spark Streaming* que é um sistema de processamento

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

O *Apache Storm* é um sistema de computação tolerante a falhas em tempo real que facilita o processamento de fluxo de dados (Apache Storm, 2020; LOPEZ, 2018). Ao invés de rodar trabalhos (*jobs*) como algumas ferramentas citadas anteriormente, o *Apache Storm* executa topologias. Os *jobs* eventualmente finalizam, e as topologias rodam sempre ou até serem finalizadas por comandos. Uma topologia constitui de processos *workers* sendo executados ao longo de 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* 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* recebem os dados de um *spout* e processam esses dados (filtragem, funções de agregação e união, etc.).

Cada processo *worker* no *Storm* é uma Java Virtual Machine (JVM) física e executa um conjunto de tarefas para uma topologia rodando um ou mais executores. Um executor é uma *thread* gerada por um processo *worker*. Cada executor pode rodar 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 Apache Flink

O *Apache Flink* é uma plataforma de processamento de distribuído para computação *stateful* sobre fluxo de dados limitados (tem um início e um fim) e ilimitados (não tem um fim definido) (Apache Flink, 2020). Essa plataforma segue um paradigma que abrange o processamento de

fluxos de dados contínuos e o processamento em lote (CARBONE et al., 2015; LOPEZ, 2018). O *Flink* pode ser integrado a todos os 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 do Java, Scala ou Python.

2.4.1 Arquitetura

Quando o *Flink* é inicializado, o *Job Manager* e um ou mais *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 2.3. Segundo Carbone et al. (2015), essa fase de transformação examina tipos de dados (esquema) dos dados trocados entre os operadores e cria serializadores e outros códigos para otimização da futura execução. O *Job Manager* 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 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*.

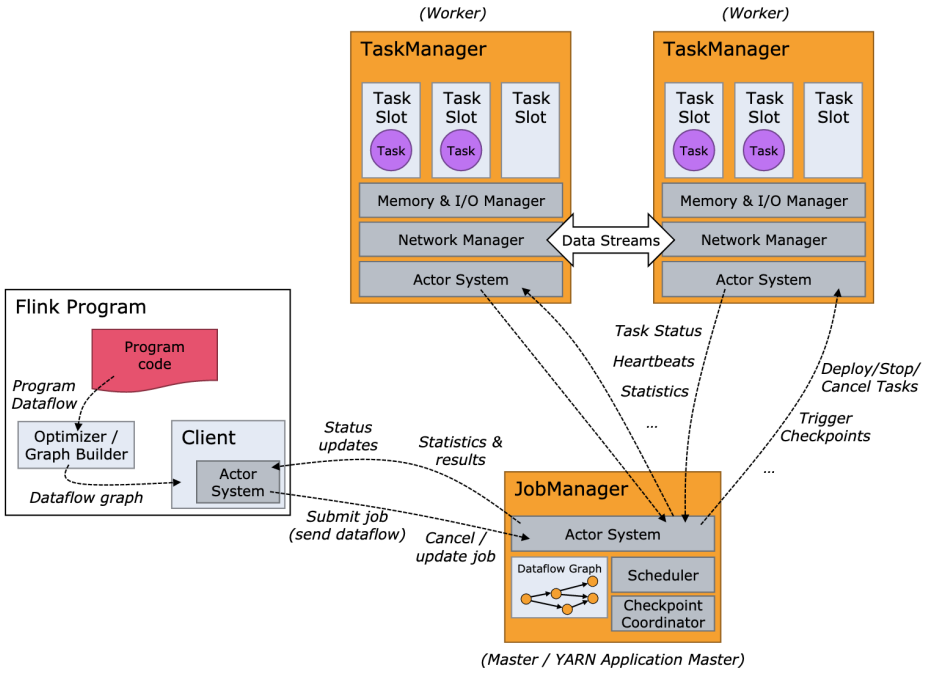


Figura 2.3: Processo do *Apache Flink* (Apache Flink, 2020)

A pilha de componentes de software do *Apache Flink* é composta pelas seguintes camadas da Figura 2.4. A camada *core* é vista como um mecanismo de processamento e execução de fluxo de dados, enxergando o processamento em lote como um caso especial (LOPEZ, 2018; CARBONE et al., 2015). A camada de APIs é composta pelo *DataStream API* que processa dados infinitos ou fluxos de dados e pelo *DataSet API* que processa dados finitos ou dados em lote. Junto ao *core*, essas APIs montam planos de execução otimizados para cada tipo de conjuntos de dados, gerando programas executáveis pelo *core*. Na camada de bibliotecas (*libraries*), há bibliotecas específicas para cada domínio que geram programas API *Data Stream API* ou *DataSet API*. Essas bibliotecas são: *FlinkML* para aprendizado de máquina, *Gelly* para processamento de grafos, *Table* para domínios relacionais (SQL), e CEP (*Complex Event Processing*) para processamento de eventos.

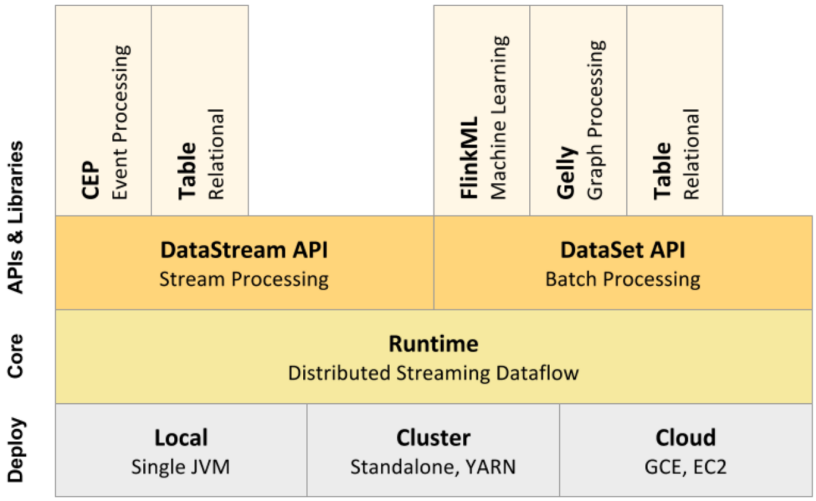


Figura 2.4: Componentes de software do *Apache Flink* (CARBONE et al., 2015).

2.4.2 *Data flow* e *data streams*

Os *data streams* ou fluxo de dados e as transformações são as principais abstrações do *Apache Flink* (LOPEZ, 2018; Apache Flink, 2020). Esse fluxo dados é definido como um fluxo de registros. Já as transformações são operações (*map, filtering, reduction, join*, etc.) aplicadas de forma incremental nos *data streams*, gerando um novo fluxo de dados. Cada uma dessas transformações pode ser paralelizada por um parâmetro de paralelismo (LOPEZ, 2018).

Um programa *Flink* é mapeado para um grafo acíclico direcionado *data flow* utilizado pelo *Job Manager* (CARBONE et al., 2015). Esse grafo é composto por operadores de transformação e fluxo de dados (Apache Flink, 2020). Para facilitar o paralelismo desse grafo de execução, os operadores que agem sobres 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 com uma fonte de dados e terminam com um *sink* que escreve os dados de saída em algum sistema de armazenamento suportado (*Apache Kafka, Amazon Kinesis Streams, Hadoop Filesystem, Apache Cassandra*, etc.) (Apache Flink, 2020). Na Figura 2.5, pode-se observar um exemplo programa *Apache Flink* escrito em Scala e seu grafo de execução. Nesse exemplo, define-se como fonte 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*.

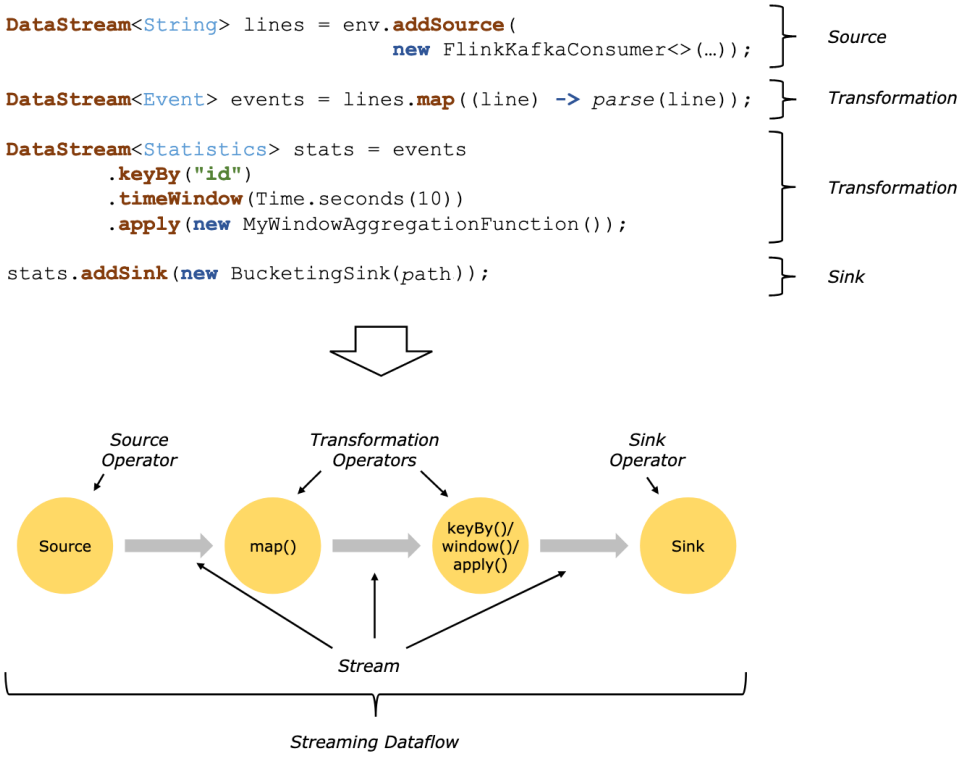


Figura 2.5: Exemplo de código e *dataflow* do *Apache Flink* (Apache Flink, 2020)

2.4.3 Tolerância a falhas

O *Apache Flink* implementa a tolerância a falhas combinando repetição e *checkpoint* dos fluxos (CARBONE et al., 2015; Apache Flink, 2020). Um *checkpoint* está relacionado com pontos específicos dos fluxos de entrada, juntamente com o estado dos operadores. Um fluxo de dados pode ser retornado a partir de um *checkpoint*, mantendo a consistência de ”exatamente uma vez”(não há dados duplicados e nem dados que não sejam processados), e restaurando o

estado dos operadores e eventos naquele momento. Portanto, as falhas são tratadas de forma transparente e não afetam a exatidão da execução de um programa *Flink* (Apache Flink, 2020).

O algoritmo de *checkpoint* assíncrono e incremental garante um impacto mínimo em latência no processamento (CARBONE et al., 2015). Além disso, para reduzir o tempo de recuperação, o *Apache Flink* tira um *snapshot* 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 chamar eventos passados para agregar dados (Apache Flink, 2020; CARBONE et al., 2015).

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

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

2.5 Detecção de Novidade

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 classificação de exemplos que diferem de exemplos anteriores (PERNER, 2007; GAMA; RODRIGUES, 2010). Esses métodos tratam da classificação em conjutos de dados que evolvem com o tempo levando em consideração as características desse tipo de conjuntos.

São características desse tipo de conjunto: a evolução de conceito (*Concept Evolution*) onde novos padrões podem surgir; desaparecimento ou recorrência de conceito, onde 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* (GAMA; RODRIGUES, 2010).

Os métodos de ND para reconhecimento padrões novidade ou anomalias são aplicadas à diversos problemas como Detecção de Intrusos (COULL et al., 2003; SPINOSA; CARVALHO;

GAMA, 2008; VIEGAS et al., 2019; CASSALES et al., 2019), Detecção de Falhas (ZHANG et al., 2006), Diagnósticos Médicos (PERNER, 2009), Detecção de Regiões de Interesse em Imagens (SINGH; MARKOU, 2004) Detecção de Fraudes (WANG et al., 2003; ABDALLAH; MAAROF; ZAINAL, 2016), Filtros de Spam (HAYAT; HASHEMI, 2010) e Detecção de Variações Comportamentais em um Jogador (VALLIM et al., 2013).

Alguns métodos tratam de novidades e anomalaias como um classificação de uma ou duas classes (binariamente) onde um conceito representa a classe normal e anomalias são reprensantadas pela abstência de conceito no modelo ou como um segundo conceito no modelo. A abordagem de classificação binária no entanto não é adequada para representar múltiplos conceitos em um mesmo conjunto de dados, para isso é necessário abordar ND como classificação multi-classe. No entanto, 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 multiplos padrões que surgem simultaneamente num intervalo de avaliação (FARIA et al., 2015; GAMA; RODRIGUES, 2010).

A maioria dos métodos de ND são construidos 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 cada exemplo em um dos conceitos do modelo ou marca o exemplo como desconhecido. Ainda na segunda fase, para atualizar o modelo os exemplos marcados como desconhecidos são utilizados para a extração de novos conceitos ou variações em conceitos conhecidos (GAMA; RODRIGUES, 2010).

Dentre os métodos de ND que baseam-se em aprendizado *Offline-Online*, muitos são baseados em algoritmos de agrupamento não supervisionados, tanto para construção do modelo inicial como na extração de novos conceitos dos exemplos não explicados pelo modelo marcados como desconhecidos (SPINOSA; CARVALHO; GAMA, 2009; MASUD et al., 2011; FARIA et al., 2013)

2.6 O algoritmo MINAS

Um algoritmo de ND que tem recebido atenção nos últimos anos é o algoritmo MINAS orignialmente proposto por Faria et al. (2013), refinado em Faria, Carvalho e Gama (2015) e recentemente aprimorado por Silva (2018) com uso de conceitos *Fuzzy* e expandido por Costa (2019) para tratar problemas multi-rótulo além dos problemas multi-classe já tratados.

Esse algoritmo segue a abordagem de aprendizado *Offline-Online* e usa por base algoritmos de agrupamento não supervisionados como *K-means* e *CluStream*.

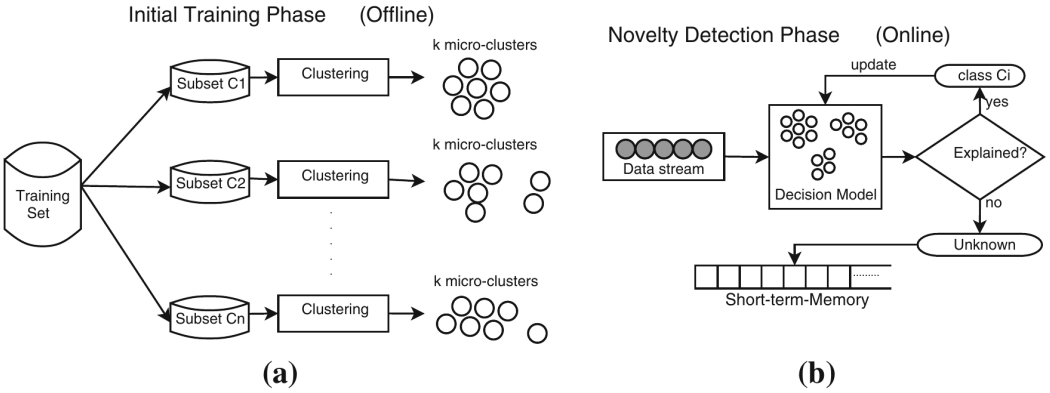


Figura 2.6: Visão geral do algoritmo MINAS com fases *Offline* (a) e *Online* (b). Extraído de Faria, Carvalho e Gama (2015)

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.

3.1 Algoritmos de Detecção de Novidades

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, 2015). Neste trabalho, consideram-se algoritmos derivados aqueles apresentados em trabalhos publicados após 2016 que estendem a implementação original seguindo sua estrutura básica.

3.1.1 Algoritmo FuzzyND

O algoritmo FuzzyND, derivado do MINAS é proposto por Da Silva et al. (2018). FuzzyND incrementa o algoritmo original aplicando a ele teorias de conjuntos *fuzzy* pela modificação da representação dos *clusters*. 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. (2015) extraindo dessa matriz duas métricas: acurácia (*Macro F-Score*) (SOKOLOVA; LAPALME, 2009) e taxa de desconhecidos (*UnkR*) (FARIA; CARVALHO; GAMA, 2015). 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.

3.1.2 Algoritmos MINAS-LC e MINAS-BR

O algoritmo MINAS-LC é proposto por Costa (2019) 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-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). O 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. (2015) 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;

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

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.

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[2] (Da Silva et al., 2018; SILVA, 2018;

[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 (COSTA, 2019), para hiper-cubos tratando *data sets* onde as características representadas pelas dimensões são completamente independentes; c) um modo interativo onde o *cluster* é formado, mostrado ao especialista que o classifica como inválido (ruído ou não representativo) ou válido, podendo conter uma ou mais classes e, se 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 pertencente a 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 a classes distintas e tratamento adequado.

COSTA et al., 2019).

Nota: helio: Mining wireless networks

3.2 Processamento Distribuído de Fluxo de Dados em Tempo Real

Nesta Seçãoabordamos trabalhos que aplicam algoritmos de detecção de novidades para fins diversos em fluxos de dados reais.

3.2.1 Ferramenta BigFlow

Nota: Incompleto

(MAWI Working Group Traffic Archive, 2020; FONTUGNE et al., 2010) 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.

3.2.2 A Ferramenta CATRACA

O trabalho LOPEZ (2018) aborda detecção de ameaças a redes de computadores em tempo real e para atingir esse objetivo, propôs a ferramenta CATRACA[3]. A ferramenta CATRACA é composta de funções virtuais de rede (*Network Function Virtualization*, NFV) executadas em dispositivos de de rede virtuais estabelecendo um ambiente de computação em névoa. Também compõem a ferramenta uma interface gráfica que apresentada a visualização dos fluxos processados bem como os conhecimentos extraídos desse processamento.

Pra avaliação da ferramenta CATRACA dois conjuntos de dados são utilizados. Um conjunto sintético (UFRJ/GTA) criado por uma simulação de rede de computadores com 214,200 fluxos de rede totalizando 95 GB de pacotes capturados, é composto de 24 atributos e 16 classes. O outro conjunto, referido como NetOp, foi coletado de um operador de rede que atende 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.

Para desenvolvimento da ferramenta CATRACA, LOPEZ (2018) avaliou e comparou as plataformas de processamento de fluxo de dados em tempo real disponíveis. A comparação envolveu:

Nota: o que vale é o conceito

1. Apache Storm versão 0.9.4 de 2015-03-18 (atualmente na versão 2.1.0, publicada em 2019-10-25);
2. Apache Flink versão 0.10.2 de 2016-02-03 (atualmente na versão 1.10.0, publicada em 2020-02-11);
3. Apache Spark Streams versão 1.6.1 de 2016-02-27 (atualmente na versão 2.4.5, publicada em 2020-02-02);

Além das plataformas comparadas é importante mencionar a presença em todos os testes do Apache Kafka na versão 0.8.2.1 de 2015-02-26 (atualmente na versão 2.4.0 de 2019-12-13). A estratégia de avaliação continha um

[3] A ferramenta e sua documentação estão disponíveis em http://gta.ufrj.br/catraca

Itens do *data set* UFRJ/GTA eram enviados ao cluster kafka lidos pela plataforma em teste.

Os resultados apresentados por essa avaliação mostraram que Apache Storm

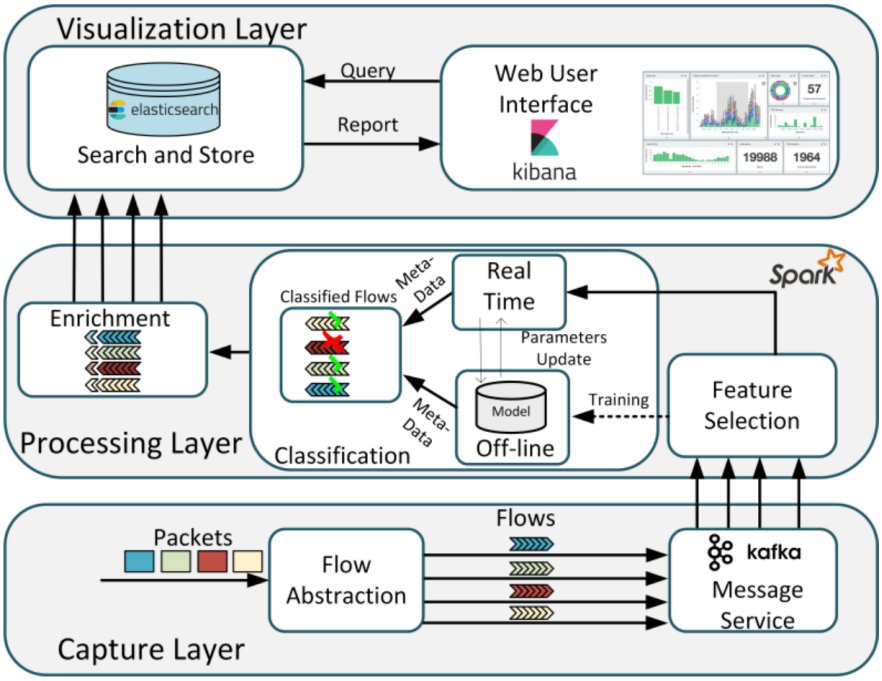


Figura 3.1: Arquitetura em camadas da ferramenta CATRACA (LOPEZ, 2018).

Catraca Lopez2018

Nota: Incompleto

Nota: Discutir que o data set não é de borda de uma rede, portanto não tem relevância para fog.

3.3 Arquitetura IDSA-IOT

Arquitetura IDSA-IOT, proposta por Cassales et al. (2019), tem por objetivo monitorar uma rede local com dispositivos

Nota: Falar do modelo de distribuição.

Nota: 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.

Em conclusão, os trabalhos discutidos nesse Capítulo tem 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

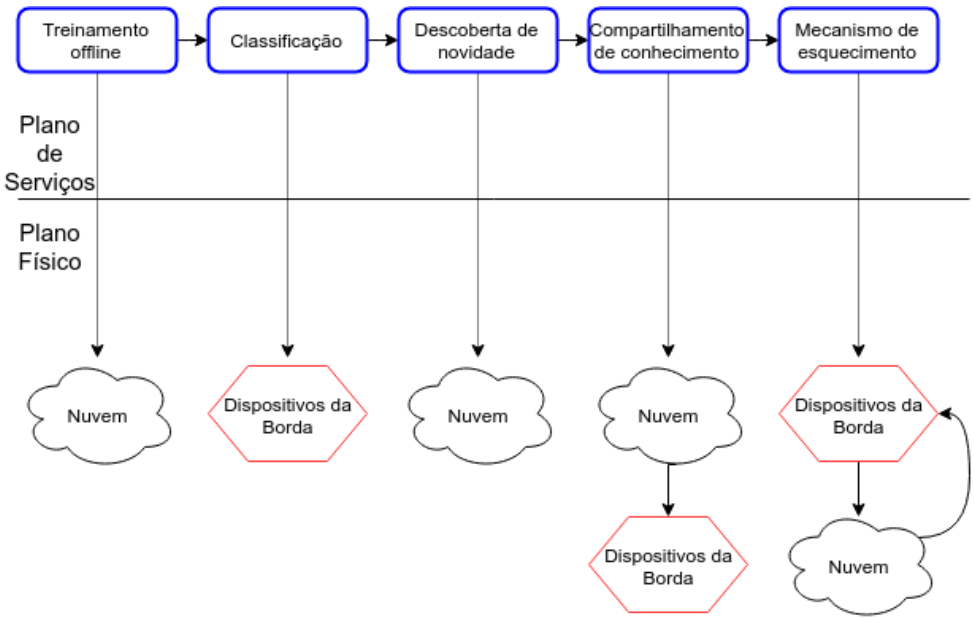


Figura 3.2: Distribuição de Serviços da Arquitetura IDSA-IOT. Produzida e traduzida por Cassales et al. (2019).

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 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 IoT.

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 milhares de dispositivos distribuídos geograficamente conectados à Internet. Com capacidades diversas como sensores e atuadores, esses dispositivos produzem e consomem Fluxos Continuos de Dados (*data streams*) com diversos objetivos. Alguns objetivos 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 computação em névoa (*fog computing*) é uma solução.

O tema de *data stream mining* envolve a classificação de novos elementos com base em um modelo, porém como *data streams* variam temporalmente e são ilimitados, todas classes contidas em um *data stream* não são previamente conhecidas. A identificação e classificação de novas classes em *data streams* é denominado Detecção de Novidades (*Novelty Detection*, ND) em *data streams*. No tema de ND, o surgimento e desaparecimento de classes é nomeado Evolução de Conceito (*Concept Evolution*) e a mudança no decorrer do *data stream* é denominado Mudança ou Deriva de Conceito (*Concept Drift*).

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

Sistemas que utilizam ND para *data streams* gerados por dispositivos IoT devem utilizar algoritmos que considerem os desafios inerentes de fluxos de dados (*Concept Evolution* e *Concept*

Drift) para adequada detecção de novidades e arquiteturas que atendam os requisitos de volume de mensagens e latência de detecção. O algoritmo MINAS é adequado 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 M-FOG, uma implementação do algoritmo MINAS sobre a plataforma *Apache Flink* que considera distribuição em um ambiente de *fog computing*.

4.1 Descrição da Implementação

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

A implementação proposta segue a arquitetura IDSA-IOT formalizada por Cassales et al. (2019) discutido 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 *source*.

O módulo auxiliar *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 treinamento.

O módulo auxiliar *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 do serviço de 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, atuali-

zando 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*.

4.2 Metodologia de Avaliação e Resultados Esperados

A avaliação da proposta apresentada será 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 aprendizado de máquina não são adequadas para avaliar detecção de novidades em *data streams* sem tratamento inicial, felizmente o tratamento necessário é estabelecido por Faria et al. (2013) e expandido por Da Silva et al. (2018), Silva (2018), Costa et al. (2019), Costa (2019). O tratamento estabelece que as métricas são extraídas de uma matriz de erro de classificação multi-classe 4.3 adaptada para detecção de novidade. A matriz 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 à 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 selecionadas são taxa de desconhecidos ($UnkR$) (FARIA et al., 2013), acurácia média (acc) e Macro F-score ($Fscore$, F1M) (SOKOLOVA; LAPALME, 2009; SILVA, 2018).

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

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

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

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

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

$$Fscore_1 = 2\cdot\frac{Precision \cdot Recall}{Precision + Recall} \tag{4.9}$$

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

O tratamento do fluxo de saída é realizado no módulo *sink* onde tem-se disponível o fluxo original com as etiquetas corretas. Esse módulo deve levar em consideração que como 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. Portanto os resultados são computados em função do fluxo de saída, então n nas equações são o índice do evento de saída. **unk** é o conjunto de eventos marcados como desconhecidos.

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 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.

4.3 Resultados preliminares

Nota: introdução dizendo que já foi caminhada/explorada uma parte

4.3.1 Implementação com *Python* e *Apache Kafka*

A primeira implementação e avaliação do 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 deu-se devido a ampla disponibilidade de bibliotecas de aprendizagem de máquina no ecossistema *Python* e simplicidade geral da linguagem. 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 pelo serviço de gerenciamento de configuração distribuída *Apache ZooKeeper*. A aplicação python consome eventos através da biblioteca *Consumer API* que expõem a distribuição através da associação de um consumidor às partições mantidas pelo *Apache Kafka*.

No entanto, a hipótese de que a distribuição de mensagens gerenciada pelo *Apache Kafka* se estenderia às processos consumidores, efetivamente distribuindo o volume de mensagens entre eles igualmente, não se realizou nos experimentos realizados. Os experimentos em questão foram compostos de 8 processos consumidores, 1 processo produtor, 1 instância *Apache Kafka* com 8 partições em seu tópico principal e uma instância *Apache ZooKeeper* associada à instancia *Apache Kafka*.

A hipótese era que como o número de partições igualava o número de produtores e cada produtor associaria-se a uma partição. No entanto essa 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 iniciativa de implementar o algoritmo MINAS em *Python* com *Apache Kafka* e atingir os objetivos de distribuição falhou, o que levou a reconsideração das plataformas escolhidas.

4.3.2 Implementação com *Apache Flink*

A segunda alternativa explorada teve por inspiração o trabalho de Viegas et al. (2019)

Nota: Esse trabalho propos-se a fazer

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 (?)

4.3.3 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

Nota: - gráficos, tempos, tabelas...

- análises e comentários

- Mostrar alguma implementação já feita e que esteja funcionando minimamente

Nota: 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 5

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.

Dando continuidade a este trabalho, seguiremos o desenvolvimento da implementação objeto (M-FOG) 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.

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;

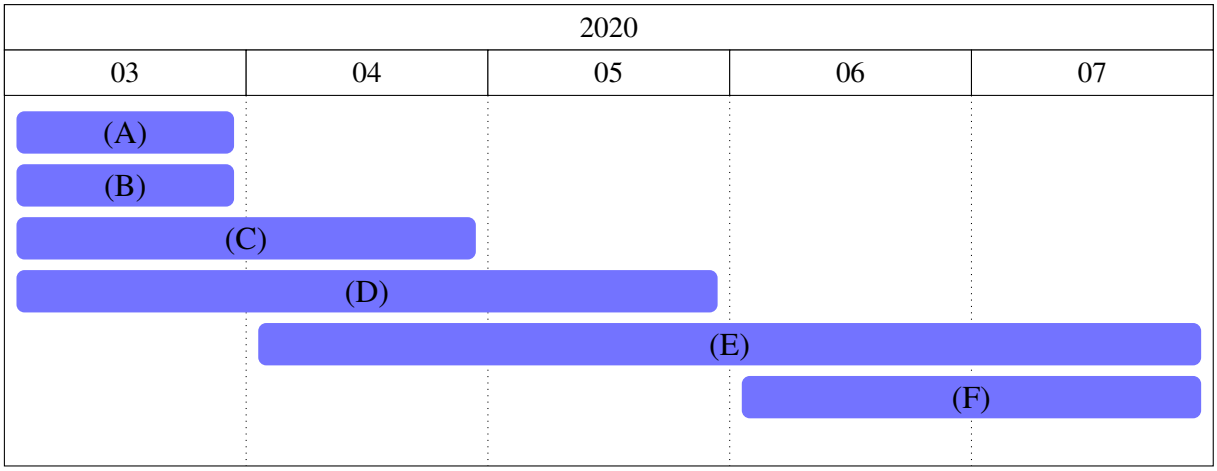
C) Validação da aplicação em contraste com a implementação MINAS original:

- 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 Disertação.



REFERÊNCIAS

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

Apache Flink. *Apache Flink*. 2020. Disponível em: ⟨https://flink.apache.org/⟩.

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/⟩.

Apache Storm. *Apache Storm*. 2020.

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⟩.

CARBONE, P.; KATSIFODIMOS, A.; EWEN, S.; MARKL, V.; HARIDI, S.; TZOUMAS, K. *Apache Flink^{TM}: Stream and Batch Processing in a Single Engine*. [S.l.], 2015. v. 36, n. 4. Disponível em: ⟨http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-198940⟩.

CASSALES, G. W.; 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.

COULL, S.; BRANCH, J.; SZYMANSKI, B.; BREIMER, E. Intrusion detection: A bioinformatics approach. In: IEEE. *19th Annual Computer Security Applications Conference, 2003. Proceedings.* [S.l.], 2003. p. 24–33.

Da Silva, T. P.; 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.; GAMA, J.; CARVALHO, A. C. Evaluation methodology for multiclass novelty detection algorithms. *Proceedings - 2013 Brazilian Conference on Intelligent Systems, BRACIS 2013*, p. 19–25, 2013.

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 2015. 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 2015. 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⟩.

GAMA, J.; RODRIGUES, P. P. *Knowledge Discovery from Data Streams*. [S.l.]: Chapman and Hall/CRC, 2010. ISBN 9781439826119.

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

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

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.

MASUD, M.; GAO, J.; KHAN, L.; HAN, J.; THURAISINGHAM, B. M. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Trans. on Knowledge and Data Engineering*, IEEE, v. 23, n. 6, p. 859–874, June 2011. ISSN 1041-4347.

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

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

PERNER, P. Concepts for novelty detection and handling based on a case-based reasoning process scheme. In: *Advances in Data Mining. Theoretical Aspects and Applications*. [S.l.]: Springer, 2007. p. 21–33. ISBN 978-3-540-73435-2.

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

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⟩.

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

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

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⟩.

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

SPINOSA, E. J.; CARVALHO, A. P. de Leon F. de; GAMA, J. a. Novelty detection with application to data streams. *Intell. Data Anal.*, IOS Press, NLD, v. 13, n. 3, p. 405–422, ago. 2009. ISSN 1088-467X.

VALLIM, R. M.; FILHO, J. A. A.; MELLO, R. F. de; CARVALHO, A. C. de. Online behavior change detection in computer games. *Expert Systems with Applications*, v. 40, n. 16, p. 6258 – 6265, 2013. ISSN 0957-4174. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0957417413003576⟩.

VIEGAS, E.; 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⟩.

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

ZAHARIA, M.; XIN, R.; WENDELL, P.; DAS, T.; ARMBRUST, M.; DAVE, A.; MENG, X.; ROSEN, J.; VENKATARAMAN, S.; FRANKLIN, M.; GHODSI, A.; GONZALEZ, J.; SHENKER, S.; STOICA, I. Apache spark: A unified engine for big data processing. *Communications of the ACM*, v. 59, p. 56–65, 11 2016.

ZAHARIA, M.; XIN, R. S.; WENDELL, P.; DAS, T.; ARMBRUST, M.; DAVE, A.; MENG, X.; ROSEN, J.; VENKATARAMAN, S.; FRANKLIN, M. J.; GHODSI, A.; GONZALEZ, J.; SHENKER, S.; STOICA, I. Apache spark: A unified engine for big data processing. *Communications of the ACM*, v. 59, n. 11, p. 56–65, 2016. ISSN 15577317.

ZHANG, J.; YAN, Q.; ZHANG, Y.; HUANG, Z. Novel fault class detection based on novelty detection methods. In: ____. *Intelligent Computing in Signal Processing and Pattern Recognition: International Conference on Intelligent Computing, ICIC 2006 Kunming, China, August 16–19, 2006*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 982–987. ISBN 978-3-540-37258-5. Disponível em: ⟨https://doi.org/10.1007/978-3-540-37258-5_124⟩.