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

SUMÁRIO

Capítulo 1

INTRODUÇÃO

A Internet das Coisas (*Internet of Things* - IoT) é um tema frequentemente abordado na última década claramente motivado crescimento sem precedentes do número de dispositivos dessa categoria e respectivos dados gerados que podem trazer novos conhecimentos através da análise desses dados. 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, 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 podem ser aplicadas para os metadados gerados pela comunicação entre esses nós e a Internet num serviço de detecção de intrusão.

Essas técnicas 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.

Esse tipo de análise é implementada 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

Nesse contexto, 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.

Essa arquitetura foi avaliada com conjunto de dados *Kyoto 2006+* que coletou dados de 348 *Honeypots* (maquinas isoladas com diversos softwares com vulnerabilidades conhecidas expostas a 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 trafego entre normal, ataque conhecido e ataque desconhecido. Contudo, esse algoritmo ainda não foi implementado e avaliado com paralelismo *multi-core* ou distribuído.

Outras propostas tratam do caso de grandes volumes e velocidades, como é o caso de Viegas et al. (2019) que apresenta *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 conjunto de dados *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 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. A escolha dessa abordagem não é nova e visa mitigar problemas como *Concept Drift* e *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 MINAS (FARIA; CARVALHO; GAMA, 2016).

Esses trabalhos abordam detecção de intrusão 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

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 experimentos baseados na literatura e conjunto de dados públicos relevantes para facilitar posterior comparação com outras implementações.

Para atingir esse objetivo geral, alguns objetivos secundários são propostos:

- Avaliar plataformas de processamento distribuído de fluxos como:
  - *Apahce Kafka* com *Python*;
  - *Apahce Kafka Streams*;
  - *Apahce Spark Streaming*;
  - *Apahce Flink*;
- Implementar algoritmo MINAS sobre *Apache Flink*;
- Executar a implementação com *datasets* 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 pontas;
  - Detecção de novidade na nuvem;
  - Detecção de novidade nas pontas e em nuvem;

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

1.3 Proposta Metodológica

(Metodologia) (como)

Flink - dizer que vai usar, para fazer o que e porque escolheu

Kafka ?

Raspberry - para todos

A base -

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.

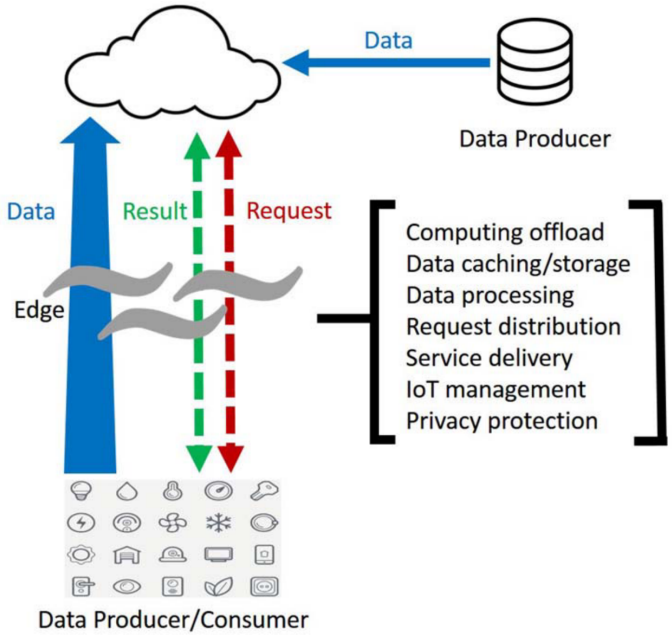


Figura 2.1: Paradigma de *Edge Computing* (SHI et al., 2016).

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).

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.

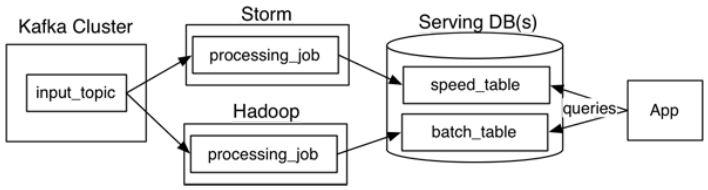


Figura 2.2: Arquitetura *Lambda* com detalhes práticos (KREPS, 2014).

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* (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 (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.

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.,) 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

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

Detecção de intrusão em redes - riscos de segurança - técnicas de intrusão e tipos de ataques - mecanismo de detecção (análise de fluxo de rede -¿ detecção de anomalia) Detecção de novidades - técnicas de Detecção de novidades - MINAS (incluir métricas) - BigFlow (incluir métricas) Processamento de Streams (big data) - cloud? - redes como stream - Atraso - Kafka/Spark/Flink Redes IoT - Restrição hardware (Energia, CPU, Mem, Rede) - Consideração FOG vs Cloud

Capítulo 3

TRABALHOS RELACIONADOS

cap 3: Trabalhos relacionados - Artigos sobre o Minas - outros que paralelizaram algoritmos de mineração de dados/streams alguns online (5-10 refs) - implementação paralelas/distribuídas em dispositivos pequenos

FuzzyND Minas

(LOPEZ, 2018)

Aqueles que contenham: - detecção de anomalia em streams - detecção de intrusão em rede com processamento de streams - BigFlow

3.1 BigFlow

Table 1 Network-level feature set used in the experiments throughout this work [18]. Types: Host-based (Host to All), Flow-based (Source to Destination, Destination to Source, Both)

Features:
- Number of Packets,
- Number of Bytes,
- Average Packet Size,
- Percentage of Packets (PSH Flag),
- Percentage of Packets (SYN and FIN Flags),
- Percentage of Packets (FIN Flag),
- Percentage of Packets (SYN Flag),
- Percentage of Packets (ACK Flag),
- Percentage of Packets (RST Flag),

- Percentage of Packets (ICMP Redirect Flag),
- Percentage of Packets (ICMP Redirect Flag),
- Percentage of Packets (ICMP Time Exceeded Flag),
- Percentage of Packets (ICMP Unreachable Flag),
- Percentage of Packets (ICMP Other Types Flag),
- Throughput in Bytes,
- Protocol

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 datasets 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 dataset 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 dataset 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 dataset 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.

concluir com um gap

Capítulo 4

PROPOSTA

o que fazer e como fazer, resultados esperados

Capítulo 5

IMPLEMENTAÇÃO E TESTES

5.1 Descrição da Implementação

- 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 Cenário 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

- detecção de intrusão - Arquitetura guilherme (dispositivos pequenos vs cloud) (CASSALES et al., 2019)

- BigFlow com dataset atual e maior dataset kdd99

5.3 Experimentos e Resultados

- 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

passos feitos/a fazer 1. Entender Minas 2. Analisar/descrever dataset KDD 3. Notas sobre implementação Python/Kafka/Minas (não escala como esperado) 4. BigFlow (dataset mais novo, usa flink) 5. Plataforma Flink (processamento distribuído) Proposta 6. Implementar minas em Scala/Flink 7. Testar com datasets KDD e BigFlow 8. Validar/Comparar métricas com seus trabalhos correspondentes

- 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?

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

Capítulo 6

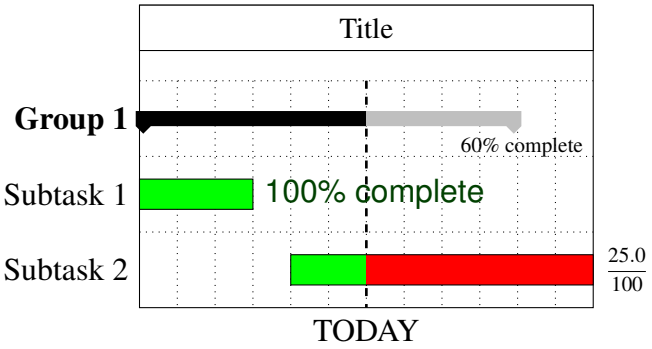
CRONOGRAMA

cap 5: Cronograma até a defesa

6.1 Cronograma

Lista de etapas

Quadro (linhas: etapas x colunas: meses)



Etapas previstas:

- E1 - Implementação distribuída do MINAS em uma rede de SBC com Rapberry: meses 01 a 03
- E2 - Preparação da base de dados para treinamento e teste: Mês 02
- E3 - Teste ... : meses 04 a 07
-

REFERÊNCIAS

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/⟩.

Dastjerdi, A. V.; Buyya, R. Fog computing: Helping the internet of things realize its potential. *Computer*, 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⟩.

HADOOP, A. *Apache Hadoop*. 2020. Disponível em: ⟨https://hadoop.apache.org/⟩.

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⟩.

SPARK, A. *Apache Spark™ - Unified Analytics Engine for Big Data*. 2020. Disponível em: ⟨https://spark.apache.org/⟩.

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⟩.

ZAHARIA, M.; CHOWDHURY, M.; FRANKLIN, M. J.; SHENKER, S. *Spark: Cluster Computing with Working Sets*. [S.l.].