SISTEMA DE DETECÇÃO DE INTRUSÃO ESCALÁVEL E PARALELO UTILIZANDO *APACHE SPARK* E *KUBERNETES*

Luan Santana da Costa e Diego Lisboa Cardoso
Instituto de Tecnologia, Universidade Federal do Pará
66075-110 - Belém - PA -- Brasil

RESUMO

Neste artigo é problematizado a ineficiência do treinamento de soluções de segurança baseadas em *Machine Learning* capazes de atingir um tempo de resposta reduzido devido ao alto volume de informação que deve ser processado. Dessa forma, é proposto um *N-IDS (Network Intrusion Detection System*) baseado em rede neural *MLP* (*Multilayer Perceptron*) implantado em uma infraestrutura paralela em *Cloud*, composta por *Apache Spark* e *Kubernetes*. Foi verificado que a técnica de rede *MLP* mostra-se como uma solução alternativa eficaz capaz de detectar 98, 0% dos casos analisados, além de que a implementação em conjunto com *Apache Spark* e *Kubernetes*, com recurso de alocação dinâmica, consegue equilibrar o t*radeoff* entre tempo de resposta da aplicação e recursos utilizados, mostrando a viabilidade de implementação em ambientes eficientes e que necessitam de tempo de resposta reduzido.

PALAVRAS-CHAVE

Spark, Kubernetes, Machine Learning, MLP, N-IDS

1. INTRODUÇÃO

Devido ao rápido crescimento de serviços oferecidos online, de qualquer natureza, surge a necessidade da criação de um ambiente seguro, responsável em manter a qualidade dos serviços oferecidos (Ali et al., 2016). Porém, cada vez mais, os ataques a esses serviços estão ficando mais complexos, assemelhando-se a acessos comuns e dificultando a detecção por soluções de segurança. Devido a esse fato, diversos estudos sobre a aplicação de técnicas de *Machine Learning* (ML) (Hussain and Khan, 2020), (Belouch et al., 2018), (Ahmad et al., 2020) foram implementadas como potenciais soluções no auxílio na detecção de intrusão.

Ambientes críticos necessitam que soluções de segurança possam ser atualizadas de forma simples e rápida com o intuito de manter o ambiente protegido de possíveis ameaças. Portando, soluções capazes de trabalhar com alto volume de dados de forma paralela e escalável são essenciais no auxílio à detecção de intrusão, onde o projeto Apache Spark mostra-se como solução alternativa por funcionar de modo paralelo e escalável em *Cloud*, além de permitir a implementação de técnicas de *Machine Learning* (ML).

O dimensionamento inadequado de soluções em modelo *cluster* pode impactar diretamente do tempo de resposta das aplicações, onde em ambientes de missão crítica busca-se por soluções eficientes capazes de reduzir o *tradeoff* existente entre uso de recursos e tempo de execução. Dessa forma, neste trabalho será estudado a viabilidade de implantação de um N-IDS (*Network Intrusion Detection System*) baseado em ML utilizando o Apache Spark e Kubernetes (Ambiente de *Cloud*) com o intuito de reduzir o tempo de processamento de aplicações de segurança em ambientes de missão crítica, além de testar o recurso de alocação dinâmica do Apache Spark com o objetivo de verificar a aplicabilidade para ambientes sensíveis com a intenção de reduzir a perda de desempenho ocasionada pelo dimensionamento inadequado de aplicações em cluster.

2. REFERENCIAL TEÓRICO

2.1 *IDS – Intrusion Detection System*

A área da segurança da informação é um enorme campo do conhecimento dedicado ao desenvolvimento, testes e implantação de diversas técnicas para proteção de ambientes com informações sensíveis. Porém, cada vez mais, invasores estão se especializando nas técnicas de intrusão e quebrando barreiras de segurança como *firewall*, criptografia, redes privadas virtuais (*VPN - Virtual Private Network*), entre outras. Dessa forma, surge a necessidade de métodos alternativos que auxiliem na detecção de invasores, sendo as técnicas de detecção de intrusão barreiras adicionais de proteção ao ambiente, onde estas técnicas podem coletar e utilizar informações de ataques conhecidos, aprender seus padrões, e assim descobrir possíveis ataques à rede.

A detecção de intrusão é um conjunto de técnicas e métodos usados para detectar atividades suspeitas tanto no nível da rede quanto em host. Estas técnicas são enquadradas em duas categorias básicas (Rehman, 2003): Sistemas de detecção de intrusão baseados em assinatura e sistemas de detecção de anomalia; esta última busca aplicar análise estatística, sistema especialistas, aprendizado de máquina ou sistemas baseado em mineração de dados (Zhenwei, 2011), com o intuito de analisar o comportamento da rede e classificar tentativas de intrusão, mesmo não existindo em base de dados.

2.2 *Machine Learning*

O termo *Machine Learning* refere-se a uma sub-área da *Artificial Intelligence* dedicada a estudar algoritmos que melhoram automaticamente através da experiência. Este processo de melhoria dos algoritmos acontece através de um modelo aprendizado baseado em amostras de dados, onde o algoritmo consegue adaptar-se a criar respostas automáticas à novos dados de entrada, permitindo melhorá-los continuamente com o intuito de concluir suas tarefas de forma eficiente e eficaz (Zhenwei, 2011), sem serem programados especificamente para isso.

O campo de *Machine Learning* abrange diversas técnicas que podem ser aplicadas a detecção de intrusão, podendo-se destacar *Decision Tree*, *Bayesian Learning*, *Support Vector Machines* (Almalawi, 2021), assim como uma das mais utilizadas para modelos de predição e classificação que é *Neural Network* (ex. *Multilayer Perceptron*) por ser umas das redes mais versáteis quanto a sua aplicabilidade (Silva, 2016).

2.3 *Apache Spark*

O projeto Apache Spark é um *framework* mantido pela *Apache Software Foundation*, desenvolvido para trabalhar frente a área de *Big Data*, permitindo manusear diversos tipos de dados (estruturados, não estruturados e semi-estruturados) através de recursos de *Streaming* até *Machine Learning* (Drabas, 2018).

Para um *workload* funcionar no Spark, ele deve ser desenvolvido especificamente para este fim, e deve ser utilizada uma linguagem compatível, como: Python, Java, Scala ou R. Estas aplicações são executadas em processos independentes no cluster, coordenados pelo componente *SparkContext* em seu programa principal (chamado de *Spark-driver*). Por sua vez, o *SparkContext* pode-se conectar a vários *Cluster Manager* que são responsáveis pela alocação de recursos recursos físicos (memória RAM, dispositivos de escrita, *cores* de processamento, entre outros), como é o caso do *Kubernetes* . Dessa forma, o *Apache Spark* pode provisionar a quantidade necessária de E*xecutors* (componentes responsáveis pelo processamento e armazenamento das aplicações). Após toda a infraestrutura provisionada, o *SparkContext* envia as tarefas que devem ser executadas pelos *Executors*, aguardando a finalização, e então retorna os dados de saída da aplicação.

2.4 *Kubernetes*

Kubernetes é um sistema de código aberto para automatizar a implantação, escalonamento e gerenciamento de aplicativos em *containers* (CNCF, 2015). Este sistema funciona em modelo Cluster, composto por nós de controle (*Master*) e de virtualização (*Worker*).

Os nós de controle possuem uma estrutura chamada de *Control Plane*, responsável em armazenar, otimizar e garantir que todas solicitações do cluster sejam executadas. Nesta estrutura, o componente *etcd* é responsável em armazenar todos os comportamentos do *cluster*, o componente *schedule* é responsável em definir o melhor nó de virtualização para criação de *container*, o *Controller Manager* é um conjunto de controladores responsáveis pelo funcionamento adequado do *Cluster*, e *API Server* é a interface de comunicação entre todos os componentes. Já o nó de virtualização, possui o componente *kubelet* que é basicamente um *Middleware* de comunicação entre *API Server* e diversos Container *runtime* (Docker, Containerd e CRI-O), que por sua vez, são responsáveis pela criação e gerenciamento de *containers*.

3. TRABALHOS CORRELATOS

Possuir um modelo de detecção de intrusão não é suficiente quando trata-se de ambientes altamente dinâmicos, existindo a necessidade de que ajustes no modelo possam ser realizados com rapidez, de forma controlada e segura, surgindo a necessidade da aplicação de sistemas paralelos e bem gerenciados com o intuito acelerar o mecanismo de detecção. Dessa forma, são apresentados 3 (três) modelos de N-IDS utilizando paralelização em nuvem.

Em (Kurnaz, 2019), o Prof. Dr. Sefer Kurnaz, criou um *IDS* utilizando Apache Spark com uma quantidade fixa de executors e duas técnicas de aprendizado de máquina: *Multilayer Perceptron* e *Random Forest*. A primeira, classifica como ”ataque” ou ”não ataque”, e a segunda classifica o tipo do ataque caso, tal como: *Exploit*, *Worm*, entre outros. As técnicas aplicadas mostraram-se eficazes com acurácia de 99,37% e 99,56%, respectivamente, para o dataset aplicado. Mostrando que a utilização desta técnica de *ML* em conjunto com Apache Spark é capaz de auxiliar fortemente na detecção de intrusão.

Em (Wu, W. and Xu, S., 2020), Wu e Xu, utilizaram uma abordagem semelhante para detecção de intrusão utilizando computação em *cloud*. Para isso, propuseram a implementação de um algoritmo probabilístico responsável em encontrar associações de regras, e como mecanismo de paralelismo, foi utilizado um *Cluster* Apache Hadoop. Foram utilizados 5 (cinco) nós fixos de processamento com recursos bem delimitados. No artigo, foi apresentado que o modelo proposto, para base de dados acima de 1,2 G mostra-se eficiente, possuindo um tempo de resposta menor que um modelo a priori, sem clusterização.

Em (Hai and Khiem, 2020), Hai e Khiem, apresentam um comparativo entre 3 (três) técnicas de paralelismo (Hadoop, HBase e Spark), com nós fixos de processamento, para N-IDS (*Network Intrusion Detection System*). A arquitetura proposta utiliza como base uma ferramenta bem difundida no mercado, o Snort, como centro para coleta dos *logs* e então, direcionamento para dentro dos 3 (três) modelos em comparação. Foi utilizado um processo de *Streaming* com cargas diferentes na entrada do sistema, onde no primeiro modelo foram adicionadas 1 (um) milhão de registros, e seguiu de forma linear crescente até o décimo modelo com 10 (dez) milhões de registros na entrada no sistema. No artigo, é apresentado que o modelo utilizando o Apache Hadoop em conjunto com o Snort, conseguiu um melhor desempenho em todos os casos, seguido dos modelos utilizando HBase e Spark.

Em todos os modelos apresentados, os autores são responsáveis pelo dimensionamento manual das soluções de *clusterização*, definindo dessa forma, a quantidade de *vCPUs*, memória *RAM* e nós de clusterização. Porém, uma definição inadequada irá impactar diretamente no desempenhos dos modelos, e consequentemente no tempo de processamento, o que pode ser indesejado para determinados ambientes. Dessa forma, busca-se uma análise do recurso de alocação dinâmica do Apache Spark com o Kubernetes, com o intuito de otimizar o *tradeoff* entre tempo de resposta e consumo de recursos, com o intensão de detectar um modelo aplicável e que garanta um ambiente rápido, controlado e seguro.

4. MATERIAIS E MÉTODOS

A criação do modelo de N-IDS utilizando ML e paralelismo foi separado em duas etapas que são: A definição da estrutura da rede neural, que inclui a escolha do dataset, e a definição da infraestrutura paralela que utiliza Apache Spark e o Kubernetes.

4.1 Rede Neural

Em (Khraisat et al., 2019) são apresentados 9 (nove) datasets muito utilizados como *benchmark* para a criação de N-IDS, porém o tamanho desses conjuntos de dados não abrange um volume considerável e implementável a um ambiente de produção, onde as redes neurais devem abranger o maior número de casos possíveis. Dessa forma, Sarhan et al. (2020), é apresentado o *dataset* NF-UQ-NIDS-v2 disponibilizado pela *The University Of Queensland*, Austrália, que é o conjunto de 4 (quatro) outros, todos no padrão *Netflow*. Este conjunto de dados possui 75.987.976 amostras, onde 50.822.681 são ataques a rede e o restante, acessos comuns.

Khraisat et al. (2019) apresenta que as métricas mais utilizadas para análise de performance de N-IDS estão relacionadas diretamente a matriz de confusão, dessa forma, serão utilizadas para avaliação do modelo proposto *accuracy*, *precision* e *recall*, assim como a matriz de confusão.

4.2 Infraestrutura Paralela

Objetivando acelerar a execução do N-IDS, assim como ter um ambiente resiliente e controlado, será proposto a utilização do Apache Spark em conjunto com Kubernetes, onde será analisado o recurso de Alocação Dinâmica o qual define a quantidade de *executors* que devem ser inicializados pelo Spark, de forma a verificar se esta implementação é viável a aplicações onde menor tempo de execução é crucial. Dessa forma, como mecanismo avaliativo, assim como em (Wu, W. and Xu, S., 2020), será utilizado o tempo de execução da aplicação, assim como métricas de consumo de processamento, memória *RAM* e largura de banda da rede, com o intuito de avaliar se a alocação proposta pelo Spark é capaz de otimizar os recursos e reduzir o tempo de execução do N-IDS.

5. METODOLOGIA

Seguindo o padrão apresentado nos materiais e métodos, este tópico será dividido em três grupos. Um direcionado a rede neural MLP, outro a infraestrutura paralela, onde será apresentado as configurações utilizadas por cada ambiente, e por fim como a coleta dos dados será executada.

5.1 Rede Neural *Multilayer Perceptron*

Para aplicação da rede neural, houve a necessidade de realizar um conjunto de procedimentos que são: a coleta dos dados, tratamento da base, transformações de dados, definição de parâmetros e por fim, a utilização da rede neural.

5.1.1 Pré Processamento

Inicialmente, o *dataset* foi reduzido à 35.795.255 de entradas, onde 11.855.572 são ataques a rede e o restante são acessos comuns. Dos dados utilizados, não houve remoção de nenhuma entrada, porém as colunas presentes na Tabela 1 foram removidas, pois são dados que representam o atacante, o *dataset* o qual a entrada faz parte, assim como a categoria do ataque, e que não são utilizados nesta pesquisa. Houve a necessidade de adaptar, no processo de transformação, os dados para o modo de leitura da biblioteca utilizada (Spark MLlib), onde todos os dados analisados serão representados por um vetor nomeado ”*feature*” e a saída desejada estará presente na coluna ”*label*”. No processo seguinte, normalização, todos os dados no vetor ”*feature*” serão normalizados seguindo uma distribuição normal de desvio padrão 1 e centrada em 0. Após estas etapas, a rede neural MLP é configurada.

Tabela 1. Atributos removidos do *dataset*

Colunas removidas
attack
IPV4 SRC ADDR
IPV4 DST ADDR
Dataset

5.1.2 Configurações

Diversos estudos utilizam redes neurais *MLP* como classificadores para N-IDS, sendo os parâmetros da rede neural, como épocas, taxa de aprendizado, neurônios por camada, entre outros, definidos a partir de testes empíricos que melhor adéquem o comportamento da rede. Dessa forma, foi utilizada esta abordagem no artigo e todos os parâmetros de configuração são apresentados na Tabela 2, assim como os parâmetros interpretados pelo Apache Spark para paralelismo.

Tabela 2. Configuração da rede neural

Configuração	Valor
Camada de entrada	**41 Neurônios**
Camadas escondidas	**[20, 10, 15] Neurônios**
Camada de saída	**2 Neurônios**
Máximo de épocas	**10^3**
Tolerância	**10^6**
Taxa de aprendizagem	**0.03**
Validação Cruzada	**3**
Treinamento, Teste	**[0.7, 0.3]**
BlockSize	**256 kB**

5.2 Infraestrutura Paralela

A implementação do Apache Spark no Kubernetes foi configurada em uma infraestrutura composta por 6 (seis) equipamentos físicos Dell Power Edge R940 que simula um ambiente de *Cloud*, e que permite virtualizar até 1296 modelos de virtualização. Para que a comunicação entre as ferramentas seja possível, um componente chamado Spark-submit é responsável pelo carregamento da rede neural e criação do Spark-driver, como apresentado na Figura 1A. Este componente, além de enviar o código principal da aplicação, é responsável em solicitar à ferramenta de cluster, todos os componentes necessários para implantação do Cluster Spark, tais como Spark-driver e Spark-executors, apresentado na Figura 1B. Dessa forma, toda a infraestrutura criada é dinâmica e controlada pois sua base esta centralizada em um único código responsável pelo provisionamento.

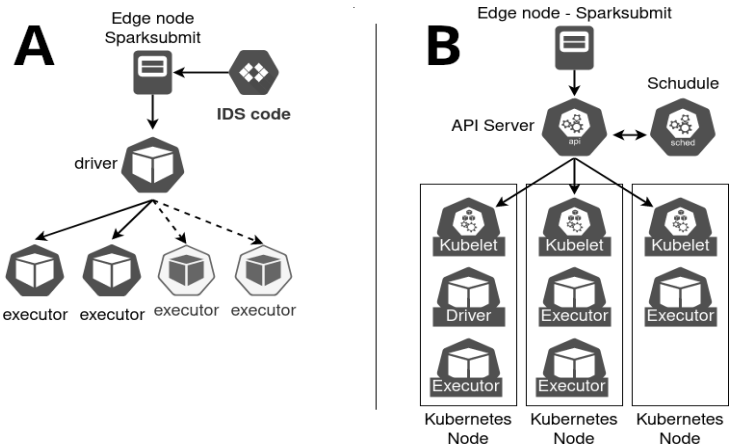

Figura 1. A. Funcionamento do cluster spark, com carregamento do workload até a distribuição entre spark-executors, a partir do spark-driver. B. Comunicação do spark-submit com o cluster kubernetes para provisionamento do cluster spark

O Spark-submit solicita ao Cluster Kubernetes que os nós sejam configurados com os parâmetros apresentados na Tabela 3.

Tabela 3. Configuração da infraestrutura do *apache spark*

Configuração	Valor
Executor Cores	**2**
Executor Memory	**6 GB**
Driver Cores	**2**
Driver Memory	**2 GB**
Memory Overhead Factor	**0.1**

Toda a infraestrutura utilizada possui equipamentos físicos com processador Intel(R) Xeon(R) Gold 6230 CPU 2.10GHz, memória RAM DDR4 Speed 2666 MHz e placa de rede Speed 1 Gbits.

5.3 Coleta dos Dados

A utilização do Apache Spark em Cluster mostra-se eficiente quando comparado ao mesmo modelo sem clusterização (Kurnaz, 2019)(Hai and Khiem, 2020), porém a configuração dos recursos é bem delimitada, o que impede um auto ajuste da aplicação quando a carga de processamento esta elevada. Dessa forma, será analisado o recurso de Alocação Dinâmica do Apache Spark em conjunto com o Kubernetes, para isso, foram montados três cenários de simulação, com 5 (cinco) *executors*, 40 (quarenta) *executors* e *executors* dinamicamente alocados. Em cada cenário serão coletadas as seguintes métricas: Consumo de processamento, consumo da memória RAM, Largura de banda consumida e tempo de execução do código. Sabendo que o N-IDS baseado em anomalias será executado em cada cenário especificado, serão coletadas métricas ligadas a matriz de confusão que são: *accuracy*, *precision* e *recall*, e serão comparadas com o intuito de avaliar se há alteração na eficácia da rede neural.

6. RESULTADOS

A execução dos três cenários propostos, com 5 (cinco) *executors*, 40 (quarenta) *executors* e executors dinamicamente alocados, foram realizados uma vez e de forma independente com o intuito de garantir que toda a infraestrutura esteja disponível para utilização se necessário. As métricas de infraestrutura coletadas estão diretamente relacionadas ao consumo de recursos como processamento, memória RAM e da rede, assim como métricas para avaliação de MLP estão relacionadas a matriz de confusão.

Valores absolutos permitem analisar a quantidade de recursos que foram extraídos do ambiente, porém uma análise relativa torna-se decisiva na verificação do consumo real em relação aos recursos atribuídos, dessa forma, é possível verificar a capacidade de utilização do processamento da ferramenta nos cenários distintos. Com base nessa análise, na Figura 2A é apresentado o consumo relativo de processamento onde, 5 (cinco) *executors* consumiram 94, 5% dos recursos atribuídos, 40 (quarenta) *executors* consumiram 80% e os executors dinamicamente alocados consumiram 84% dos recursos atribuídos.

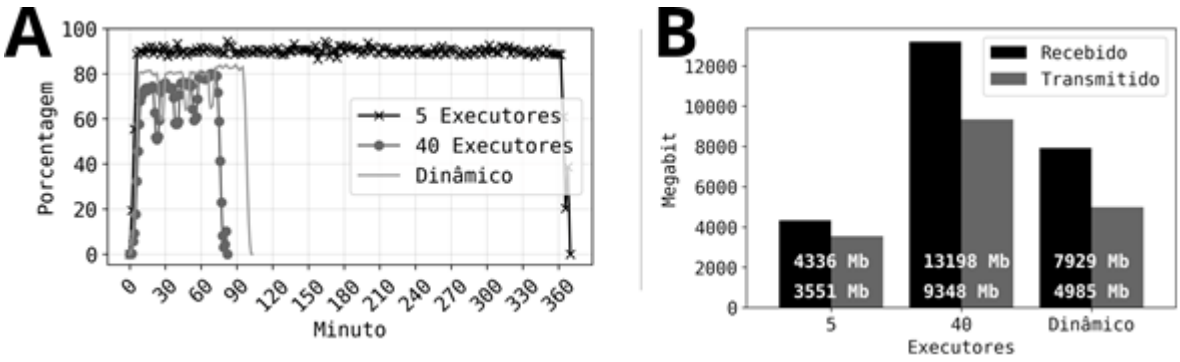

Figura 2. A. Consumo relativo de *cores* de processamento. B. Largura de banda transmitida e consumida

O modelo dinâmico mostra um equilíbrio do *tradeoff* entre tempo de execução e utilização dos recursos, permitindo ser implantado em ambientes de *Cloud* onde o valor pago é proporcional a recursos alocados. Dessa forma, atribuir muitos *executors* não é eficiente, visto que, o consumo de processamento é reduzido a

medida que *executors* não necessários são adicionados a infraestrutura (Figura 2A). Todavia, aumentar o número de *executors* (Figura 3B), apesar dos desperdício de recursos alocados, diminui o tempo de execução do projeto, podendo ser utilizado em ambientes com níveis de urgência. Devido a aplicação funcionar em modo *Cluster*, além da utilização de um *dataset* com 6 GB de armazenamento, fez com que a largura de banda consumida e transmitida nos três cenários fosse proporcional a quantidade de nós (Figura 2B), sendo um ponto delicado que deve ser analisado no projeto da rede que será utilizada para processamento paralelo.

O Apache Spark é uma tecnologia que funciona com processamento em memória RAM o que torna o desempenho frente a outras tecnologias que utilizam a escrita em disco. Este diferencial é apresentado na Figura 3A onde toda a memória RAM atribuída a todos executors foi utilizada, estourando até os recurso Memory Overhead Factor (Tabela 3) o qual permite ao Apache Spark alocar se necessário, 10% a mais da memória RAM atribuída.

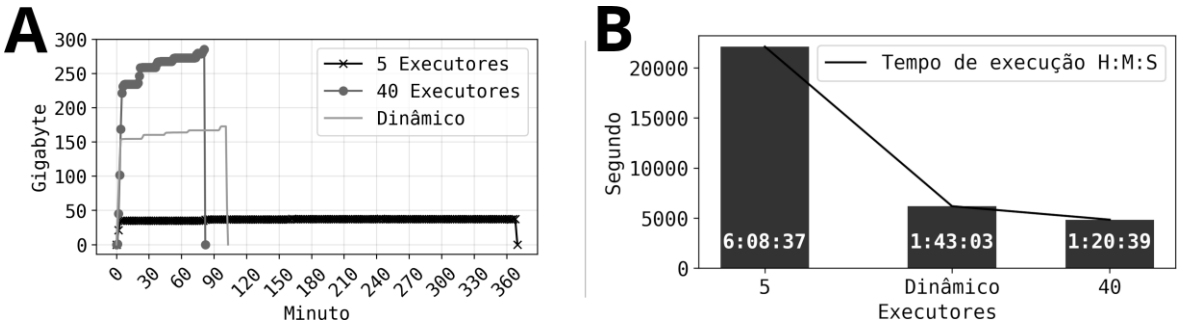

Figura 3. A. Consumo absoluto de memória RAM pelo N-IDS. B. Tempo de resposta do N-IDS

Na Figura 3B é apresentado o tempo que a infraestrutura levou para o N-IDS finalizasse o treinamento da rede, onde a alocação de mais *executors*, visivelmente melhorou o tempo de resposta da aplicação, porém a relação com número de executors não acontece de forma linear, o que pode impactar no consumo excessivo de recursos quando necessário um tempo de resposta bastante reduzido.

Em todos os cenários a rede neural MLP mostra-se como um mecanismo eficaz na detecção de intrusão (Tabelas 4, 5), conseguindo detectar até 98,03% dos casos analisados. Dessa forma, mostra-se uma excelente alternativa, em conjunto com paralelismo do Apache Spark e Kubernetes que o permite ser implementado nos mais diversos ambientes, principalmente naqueles que necessitam de um tempo de execução reduzido por tratar-se de ambientes de missão crítica.

Tabela 4. Métricas precision e recall da rede MLP aplicada nos três cenários

Nós	Precision	Recall	Accuracy
5	0.97566	0.97553	0.97553
40	0.97552	0.97538	0.97538
Dinâmico	0.98068	0.98066	0.98066

Tabela 5. Matriz de confusão da rede MLP aplicada nos três cenários

Nós	VP	FP	FN	VN
5	3450556	105890	156795	7023573
40	3449692	107323	157010	7026425
Dinâmico	3457265	97535	110101	7075598

7. CONSIDERAÇÕES FINAIS

A *Cloud* é uma alternativa para sistemas que necessitam de alto poder de processamento devido a capacidade de elasticidade (aumentar e reduzir a quantidade de nós de paralelismo), dessa forma, ambientes de missão crítica que necessitam de aplicações com tempo de resposta reduzido podem usufruir desse benefício quando implementam ferramentas como Apache Spark e Kubernetes. Estes ambientes por guardarem informações sensíveis, necessitam de camadas extras de segurança, capazes de auxiliar na detecção e bloqueio de tentativas de intrusão, onde a implementação de técnicas de *Machine Learning* mostram-se como excelentes alternativas, pois são capazes de detectar diversas tentativas intrusão, estas sendo catalogadas ou não, e reduzindo o número de falsos positivos, o que permite ambientes mais seguros e garantia de qualidade.

Diversos trabalhos (Kurnaz, 2019) (Wu, W. and Xu, S., 2020) (Hai and Khiem, 2020) utilizam tecnologias de paralelismo como intuito de reduzir o tempo de processamento de grandes bases de dados e acelerar a detecção de tentativas de intrusão. Porém, as soluções não buscam um dimensionamento adequado dos nós de paralelismo, sempre utilizando nós fixos, o que pode acarretar um gargalo à medida que os dados processados aumentam ou diminuem de tamanho, assim como quando os *cores* de processamento são requisitados, mas não utilizados.

O dimensionamento inadequado de soluções em *Cloud* pode acarretar diversos prejuízos, que são desde o desgaste de equipamentos, gastos excessivos com recursos não utilizados, indisponibilidade de máquinas, desempenho indesejado, entre outros, além de impactar no funcionamento adequado de determinados ambientes, portanto, ter soluções capazes de se auto modelarem como a combinação do Apache Spark e Kubernetes mostra-se uma solução viável capaz de reduzir o tempo de convergência significativamente (em até 72%) de uma rede MLP, quando comparado a cenários mal dimensionados, além de equilibrar o *tradeoff* entre desempenho da aplicação e recursos utilizados. Em trabalhos futuros, busca-se estudar a alocação dos recursos implementada pelo Kubernetes com o intuito de otimizar os níveis de processamento e utilização da largura de banda, além do impacto financeiro gerado pela utilização inadequada de ferramentas de segurança em Cloud.

REFERÊNCIAS

Ali, M., Khan, S. U., and Zomaya, A. Y. (2016). Security and dependability of cloud assisted internet of things;. IEEE Cloud Computing, 3(2):24–26

Ahmad, Zeeshan et al. (2020). “Network intrusion detection system: A system- atic study of machine learning and deep learning approaches”. en. In: Transactions on Emerging Telecommunications Technologies n/a.n/a. eprint: https://onlinelibrary.wiley.com/d e4150. issn: 2161-3915. doi: https://doi.org/10.1002/ett.4150. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4150 (visited on 12/30/2020).

Almalawi, Abdulmohsen (2021). SCADA security: machine learning concepts for intrusion detection and prevention. Hoboken, NJ, USA: Wiley. Isbn: 1119606039.

Belouch, Mustapha, Salah El Hadaj, and Mohamed Idhammad (Jan. 2018). “Performance evaluation of intrusion detection based on machine learn- ing using Apache Spark”. en. In: Procedia Computer Science. PROCEED- INGS OF THE FIRST INTERNATIONAL CONFERENCE ON INTELLI- GENT COMPUTING IN DATA SCIENCES, ICDS2017 127, pp. 1–6. issn: 1877-0509. doi: 10 . 1016 / j. procs. 2018 .01 .091. url: http:// www.sciencedirect.com/science/article/pii/S1877050918301029 (visited on 10/22/2020).

CNCF, Cloud Native Computing Foundation - (2015). Production-Grade Con- tainer Orchestration. https://Kubernetes.io.

Drabas, Tomasz (2018). PySpark Cookbook : Over 60 Recipes for Implementing Big Data Processing and Analytics Using Apache Spark and Python. Birm- ingham: Packt Publishing Ltd. Isbn: 1788835360.

Hai, T. H. and N. T. Khiem (2020). “Architecture for IDS Log Processing using Spark Streaming”. In: 2020 International Conference on Electrical, Com- munication, and Computer Engineering (ICECCE), pp. 1–5. doi: 10.1109/ ICECCE49384.2020.9179188.

Hussain, Mohammed Shabaz and Khaleel Ur Rahman Khan (2020). “A Survey of IDS Techniques in MANETs Using Machine-Learning”. In: Proceedings of the Third International Conference on Computational Intelligence and Informatics. Ed. by K. Srujan Raju et al. Singapore: Springer Singapore, pp. 743–751. isbn: 978-981-15-1480-7.

Khraisat, Ansam et al. (July 2019). “Survey of intrusion detection systems: techniques, datasets and challenges”. In: Cybersecurity 2.1. doi: 10.1186/ s42400 - 019 - 0038 - 7. url: https:// doi. org/ 10 . 1186 / s42400 - 019 - 0038-7.

Kurnaz, Sefer (2019). “Intrusion Detection System using Apache Spark Analytic System”. In: IOSR Journal of Computer Engineering (IOSR-JCE).

Sarhan, M. et al. (2020). “Netflow datasets for machine learning-based network intrusion detection systems”. Em: .arXiv: 2011.09144

Silva, Ivan Nunes da (2016). Redes Neurais Artificiais para Engenharia e Ciência Aplicadas. Artliber. isbn: 8588098873.

Rehman, Rafeeq (2003). Intrusion detection systems with Snort: advanced IDS techniques using Snort, Apache, MySQL, PHP, and ACID. Upper Saddle River, N.J: Prentice Hall PTR. Isbn: 978-0131407336.

Wu, W. and Xu, S. (2020). Application of mapreduce parallel association mining on ids in cloud computing environment. Journal of Intelligent & Fuzzy Systems, 39(2):1915–1923.

Zhenwei (2011). Intrusion detection: a machine learning approach. London Singapore Hackensack, NJ: Imperial College Press Distributed by World Scientific Pub. Co. isbn: 1848164475.