A Simple Distributed Approach for Running Machine Learning Based Simulations in Intrusion Detection Systems

Rui Fernandes[1] and Nuno Lopes[2,3(✉)]

[1] School of Technology, IPCA, Barcelos, Portugal
a17618@alunos.ipca.pt
[2] 2AI - School of Technology, IPCA, Barcelos, Portugal
nlopes@ipca.pt
[3] LASI - Associate Laboratory of Intelligent Systems, Guimarães, Portugal

Abstract. Intrusion Detection Systems (IDS) that use Machine Learning (ML) are a must-have for success protection when thinking of network traffic. Classification algorithms within Machine Learning have already proved their value in this research field and they are already being used in real scenarios as a service.

However, analysing large quantities of data, with possibly multiple distinct algorithms takes a considerable amount of computing and time resources on the training phase that are required to decide which is the most efficient classification model. We propose the use of a distributed computing platform, using the Ray Python Library, to deploy a simple parallel execution of ML training algorithms with minimal source code change. We use the well-known CICIDS 2017 dataset to evaluate an ML based IDS as the testing case.

The results show that the Ray library is a simple and direct approach to the parallelism in training ML algorithms, while maintaining the same deterministic output results. The execution time of the experiments was improved by a speedup of up to 2.2 when running on an 8 core CPU.

Keywords: Distributed Computing · Machine Learning · Intrusion Detection Systems

1 Introduction

Internet of Everything (IoE) is now seen as a young term that is not widely known yet and can be described as the next stage of the Internet of Things (IoT) or a superset of IoT, which is a machine-to-machine phenomenon. It connects people to people, machine-to-machine and people-to-machine systems [8].

IoE ecosystems although bringing a lot of benefits for society it also present ongoing cybersecurity challenges [16] that must be addressed to maintain the CIA triad - Confidentiality, Integrity, and Availability [11]. People are used to the internet and it must be as easy to use as possible but with the evolution

T. Pereira et al. (Eds.): IOECON 2023, LNICST 551, pp. 66–75, 2024.
https://doi.org/10.1007/978-3-031-51572-9_6

of IoT and IoE, the concerns are bigger. The impact of data loss or any other intervention can be crucial for critical systems such as Medical systems where the data must be protected and accessed for only those who are authorized, uneditable, and available anytime to be used [11].

A solution in a variety set of techniques of cybersecurity is the use of Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS) that use Artificial Intelligence to find attacks from unauthorized third parties [3]. IDS have an important role in the network infrastructure because they represent a layer of security that is able to detect a potential attack going on through the analysis of passing network traffic, in real-time, and to raise a warning about a specific threat.

Jamalipour and Mourali divide IDS into three categories: IDS based on Intrusion Data Sources, IDS based on Detection Techniques, and IDS based on Placement Strategies [5]. IDS Based on Intrusion Data Sources can be divided into Host-based IDSs (HIDS) and Network-based IDSs (NIDS). The HIDS are useful to detect attacks that do not contain network traffic such as databases, operating systems, or software logs. The NIDS are used in the network environment in order to analyse the network traffic and detect attacks.

IDS Based on Detection Techniques can be divided into four categories that make use of four different techniques. They can be Signature-Based IDS that consists in comparing the network traffic with a pre-defined signature or pattern. Another technique is Anomaly-Based IDS which consists of pre-defined standards of what is benign or malign based on network protocols and it is very efficient in the detection of malign but it can't detect what type of attack is. Specification-Based IDS are based on rules and thresholds set by experts which is similar to anomaly-based but the, in this case, the rules are applied based on someone's expertise. Finally, Hybrid-Based IDS combines both the signature-based and the anomaly-based, that is, the trade-off between the storage cost of the signature-based technique and the computing cost of the anomaly-based technique which is the best of "two worlds".

IDS Based on Placement Strategies can be divided into centralized IDS, distributed IDS, and hybrid IDS. Centralized IDS is placed either at the root node or a strategic node and analyzes the data traffic that is passing through to detect attacks. Distributed IDS Placement is a system of IDS where every node in the network will be configured with a full IDS taking advantage of a full analysis in each node preventing not only attacks from outside of the network and also inside the network. Hybrid IDS Placement combines both where there is a central IDS which is more capable of analyzing large amounts of data and a variety of lightweight nodes scattered in the network with less detection ratio but enough for the task needed.

When designing a Network Intrusion Detection System (NIDS) based on Machine Learning algorithms, the existence of previous data on attack types and their characteristics is of major importance to the success in the classification of the results, i.e., the capacity to distinguish between benign from malign (attack) packets. There are already some well-known datasets for NIDS, that cover a large

variety of attack types, such as CICIDS-2017 [14], KDD [6], NSL-KDD [17] and UNSW-NB15 [13] and the most recent, HIKARI-2021 [4]. The NIDS's datasets that focus on network traffic have a large number of entries and features. The metadata from a simple packet crossing the network can be considerable in size and most of the mentioned datasets start with 50 or more features and some of the datasets have more than 500 000 entries.

Besides the importance of having a good amount of data to train an accurate NIDS, it is also important to choose the best model. There are several previous studies using traditional Machine Learning [2,3] and also Deep Learning [7,10] algorithms that address these problems. Nevertheless, it is always a question of balance between computation power and time to get the best model possible. Given enough resources, a developer would like to train each ML or DL model on each dataset, multiple times (to make sure the model is reliable), and adapt the hyper-parameters to make the final model more accurate. This work presents a simple approach to improve the existing computing capability when training different classification models with large datasets, in the context of cybersecurity, through the use of a distributed computing framework, Ray [12].

Ray is a distributed framework based on the Actor-model, where actors (entities with behavior) are capable of communicating with one another by sending and receiving messages in a shared-nothing distributed environment. This design facilitates the deployment of actors either at a single host, possibly where each actor is mapped to a CPU core, or across multiple hosts, where each actor is placed at a different host. Since actors can only communicate through message exchange, either by sending or receiving messages, in a shared-nothing distributed setup, this design permits the flexible deployment of a solution that can be adapted to a single host multi-core deployment, or into a multiple host possible multi-core deployment.

The use of this distributed platform maintains the previous objective of selecting the best possible model from a large number of experiments. Considering the design flexibility of adapting the solution into a concurrent setup, either through multiple cores on a single host, or by running multiple hosts, one expects the total execution time of multiple experiments to decrease by taking advantage of the multi-core and possibly multiple host resources that are already available but may not be efficiently used. This framework is a clean and simple approach to parallel the execution of computing tasks with minimal source code change, using the Python programming language and its multiple ML libraries like SciKit.

The paper is organised as follows: the next section presents the Related Work, the Machine Learning Models section presents the classification algorithms and the architecture for the distributed framework, the Results and Discussion section shows the results obtained with the experiments and the last section concludes our work.

2 Related Work

With the advent of Big Data, distributed computing frameworks were developed for executing code in a distributed setting. Apache Spark [15] was one of the first frameworks, based in the Java programming language, that could distribute code across multiple hosts, so that datasets with a size larger than the capacity of a single machine could be stored on a cluster of machines, and its parts processed concurrently at each host simultaneously. Spark offers a library of ML algorithms, designed specifically for the platform.

Another framework based in the Python programming language is Dask [1], which like Spark, is dataset oriented, i.e., the data from the dataset is partitioned and distributed across multiple hosts, and code is run at each partition concurrently to obtain computation results. The development of this framework was specifically suited for ML libraries like SciKit and other numerical libraries like NumPy.

Ray [12], in turn, is a Python framework based on the actor-model which consists on a set of actors (entities with generic computing capabilities) that can process any type of data locally, and communicate exclusively through messages. Although an actor-model generic framework is not designed to run machine learning algorithms efficiently, its generic design and flexibility enables it to be adapted to any computing requirement. In the case of this work, the execution of some ML algorithms using (single host) Python based ML libraries.

The use of a distributed platform to improve ML or DL algorithms has been used previously by other authors. Zhu et al. [19] describe an approach of efficient training of deep forests, a Deep Learning algorithm, on distributed task-parallel platforms where they compared Tensorflow and Ray, and by choosing Ray as the best platform to use, they were able to outperform another algorithm described in the paper with 7x to 20.9x speedup using a cluster of 16 nodes running in the Ray architecture.

Teixeira et al. [18] propose a new architecture of a hybrid IDS consisting in a central IDS where multiple algorithms are trained with multiple datasets and a IDS per company which is a result of the best model trained by the central IDS. The edge node IDS receives updates to the model itself and returns logs of classified traffic back to the central node/IDS. Ray was used to distribute tasks by the available resources.

3 Machine Learning Models

The use of a Machine Learning model to support the IDS classification problem requires the selection of one of multiple classification algorithms that were previously proposed in the literature [2]. Additionally, datasets must be considered for the training of the model. Finally, the configuration used to distribute the computing of the training phase of the classification algorithms is presented.

3.1 Classification Algorithms

There are multiple classification algorithms available from popular ML libraries: the K-Nearest-Neighbors (KNN), the Multi-Layer Perception (MLP), the Support Vector Machine (SVM classifier), the Random Forest (RF), the Decision Tree, the Gaussian, and the Logistic Regression. The library used to train these models was the scikit-learn using default parameters. The choosing of the most suitable algorithm is made by experimenting the multiple possibilities and to compare the results.

These classification algorithms will be evaluated with the following metrics: Accuracy and Precision. Accuracy is calculated by dividing the number of correctly identified predictions, true positives and true negatives, by the total number of predictions which means that by having a high accuracy the algorithm can be more trusted to classify the traffic. Precision is calculated by dividing the number of correctly identified malicious predictions, true positives, by the total number of predictions that the algorithm classified as malicious, true positives + false positive, which means that high precision is valuable to say that we can correctly identify the different type of malicious traffic.

3.2 Dataset

The dataset chosen for this experiment was the CICIDS-2017 [14]. It was already widely studied showing that it can be used as a dataset of reference in IDS problems. It contains both benign and malign traffic which resembles true real-world data (PCAPs). The dataset already contains some analysis provided by a network analyzer, CICFlowMeter, and it was saved as a CSV file publicly accessed.

The dataset has a total of 80 features, both numerical and categorical, and more than 2 million entries and it is classified as benign or malign (Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet, and DDoS).

Several researchers have studied this dataset, Kurniabudi et al. affirm that the number of features used from the dataset affects the execution time [9].

In this case, we decided to use only 20% of the dataset, choosing 20% of each label so that we can use a balanced dataset, and using all the features so we can see the worst scenario in training time.

3.3 Distributed Training

The experiments necessary to perform this study consist in a number of classification algorithms applied to IDS specific datasets. Thorough studies make use of several algorithms with several datasets. For this demonstration, seven algorithms will be applied to one dataset. This setting, although small, is sufficient to make the proof of concept and demonstrate how a distributed framework using the actor-model can be applied to parallelise ML algorithms.

Sequential

```
def runRF():
    model = RandomForestClassifier(random_state=1)
    model.fit( CDS.X_train, CDS.Y_train)
    Y_predict = model.predict(CDS.X_test)
    accuracy = accuracy_score(CDS.Y_test, Y_predict)
```

Parallel (Ray)

```
@ray.remote
def runRF(num_CPU=1):
    model = RandomForestClassifier(random_state = 1)
    model.fit( CDS.X_train, CDS.Y_train)
    Y_predict = model.predict(CDS.X_test)
    accuracy = accuracy_score(CDS.Y_test, Y_predict)
```

Fig. 1. Code snippets for the Sequential and Parallel (Ray) versions

The usual classification pipeline starts with the training of the model with the dataset, which is a costly operation, and then testing to assess performance indicators. Our purpose is to speed up the training phase of each model. Two approaches are available to parallelise the training phase. We can approach the parallelisation of each algorithm internal execution (low-level approach), or we can parallelise the algorithm as an independent task regarding the others (high-level approach). We opted to follow the later design due to its simplicity and for not requiring knowledge on specific implementation details of the inner workings of each algorithm.

Each pair algorithm-dataset will form a task (following Ray terminology), which represents an actor. The distributed system will be made of multiple tasks, each one running on an available processor. Since each algorithm shares nothing with the other algorithms, the mapping between the algorithms and the tasks (actors) on the platform is perfect. Each task may then run at its own pace. The experiments are concluded when all tasks have finished their execution and all performance indicators can be gathered from all algorithms, so that a decision can be made regarding the best ML model.

Ray, following the actor-model, makes it very easy to adapt traditional sequential Python code for a parallel distributed environment. The Fig. 1 shows two code snippets, the first represents a typical sequential design, and the second presents the parallel enabled version. As it can be observed, the code changes are minimal: an annotation indicating the function will be deployed as a Ray task, and the launch of the Ray platform through a main call to ray.init(). The deployment of this code is automatically handled by the Ray platform, i.e., launching instances for each available processor at each host in the Ray cluster, and executing each task remotely.

4 Results and Discussion

Considering as the starting point the execution of seven classification algorithms with the CICIDS dataset, our objective is to compare the impact of multiple processors when running a set of experiments. In order to do so, the Ray framework will be used, and have each algorithm to be run as an (independent) task.

Our study will look at the impact of having multiple processors, starting with the single processor case (1P - 1 CPU), and increasing the number of processors (2P, 4P, 8P - 2,4,8 CPU), on performance indicators, namely the global execution time. All experiments were run on a single machine with an Intel(R) Core(TM)

i5-12400F CPU with 6 threads, 12 logical processors, with 16 Gb of RAM. The code was executed in Python 3.10, scikit-learn 1.2.2, and ray 2.3.1. Although just one host was used in this experiments, the design of the actor-model enables this same setup to be scaled into a cluster of hosts without any modification in the source code. The difference would be to setup a Ray cluster with all the member hosts.

In order to prove that the distributed experiments work well in multiple execution runs we designed the application to load the dataset and partition it using the same deterministic approach, so that executions are reproducible. By applying this setting we can run the same algorithm twice or more and get always the same metrics of evaluation.

Table 1. Results of the Distributed ML experience

Algorithm	Accuracy	Precision	Time (1P)	Time (2P)	Time (4P)	Time (8P)
KNN	0.990	0.990	254.045	270.987	306.704	311.024
Decision Tree	0.998	0.998	16.782	17.190	20.116	33.768
MLP	0.984	0.984	322.267	323.247	356.048	363.865
SVC	0.955	0.955	106.347	103.193	146.305	146.327
Random Forest	0.998	0.998	85.685	87.451	118.002	133.030
Linear Regression	0.948	0.944	23.561	25.004	30.781	35.644
Gaussian	0.712	0.971	3.176	3.277	3.987	4.375

Sum of the individual time	557.819	559.362	675.237	717.010
Total execution time	817.857	565.374	381.960	370.588
Speedup	1	1.447	2.141	2.207

Table 1 shows the results of the experiments described before. In these results, we can compare the algorithms with the accuracy, precision, and execution time needed to train (in seconds). The labels 1P, 2P, 4P and 8P refer to the number of CPU used.

By comparing the accuracy and precision in each execution run we can verify that the results are always the same for each algorithm which means that our implementation is deterministic. These results are not shown but were validated.

Although deterministic on the accuracy and precision metrics, the execution time of each run changes slightly for the different CPU settings. The code executed was always the same, and this fluctuation on the execution time can be attributed to Ray's internal management of tasks. The total sum of each individual experiment was slightly different from the others, and includes this internal Ray management.

We can see that if we run the entire experiment in sequence (1P experiment) the time needed to train all the algorithms is 817 s. By comparing the four values of the total execution time needed to train in each experiment we can see that

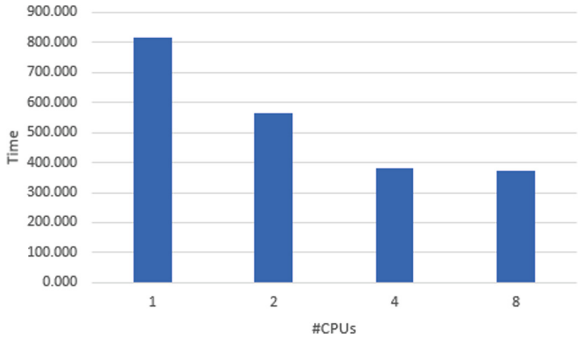

Fig. 2. Execution time vs Number of CPU)

there is a correlation between the number of CPU and the time needed. By increasing the number of CPU we reduce the time needed to train.

This result is expected since tasks are parallelised, when adding one more CPU, we add the simultaneous execution of one more actor. Although the sum of all individual times does not reduce, since the calculations are the same, the final execution time reduces, because we increased parallelism.

On the other side, we can notice a reduction of time between 2P and 4P which is 183.414 s, and between 4P and 8P which is 11.372 s. The difference between the 4P and 8P cases is smaller than the one obtained between the 2P and the 4P cases. This is due to the fact that we have only 7 tasks, but have available 8 processors. When the level of parallelism reaches the number of actors, the execution time converges to the highest individual task execution time, i.e., all tasks finish before the longest lasting but the application only finishes when all tasks are over.

In this experiment, the individual execution times differ significantly between one another. The fastest task takes 3 s (for the P1 case) and the longest task takes 322 s (for the P1 case). This high variation becomes noticeable when the number of processors increases up to the number of tasks, making the total execution time equals the time of the longest execution individual task.

The metrics of comparison of those experiments was speedup. Speedup is determined by dividing the sequential time of execution by the parallel time of execution for each experiment. We can confirm that distributed computing works well in this case because we see an increase in speedup when increasing the number of CPU. The best result occurs when using 8 CPU (8P), achieving a speedup of 2,2.

The impact of the number of CPU on the reduction of time can be seen in Fig. 2. It shows a descent tendency on the global execution time as the number of processors grow, more accentuated in the first experiments and less in the last, as explained before.

5 Conclusion

Overall this work represents an approach to distributed computing, using Ray, for running ML algorithms in IDS problems. With this approach, we were able to train seven different algorithms concurrently and by comparing the experiments with different numbers of CPU we were able to say that by using 8 CPU we can achieve a speedup of 2.2.

Our results show that it is indeed very simple and straightforward to adapt sequential Python code to run in a distributed setup. Secondly, the performance results show that even a single desktop machine with multi-core processor can speedup the execution of multiple simulations. Finally, this approach reveals itself adequate to scale a high number of experiments into a cluster of hosts with relative simplicity.

The findings of this research allow researchers to improve the process of training ML algorithms with a high number of experiments to be made, where time is crucial to make decisions.

As future work we plan to introduce more datasets and more algorithms, increasing the number of tasks to parallelise, and to quantify the effective speedup at a large scale cluster.

Acknowledgments. This work was funded by the project "Cybers SeC IP" (NORTE-01-0145-FEDER-000044), supported by Northern Portugal Regional Operational Programme (Norte2020), under the Portugal 2020 Partnership Agreement, through the European Regional Development Fund (ERDF).

References

1. Dask. https://www.dask.org/
2. Fernandes, R., Lopes, N.: Network intrusion detection packet classification with the hikari-2021 dataset: a study on ml algorithms. In: 2022 10th International Symposium on Digital Forensics and Security (ISDFS), pp. 1–5 (2022). https://doi.org/10.1109/ISDFS55398.2022.9800807
3. Fernandes, R., Silva, J., Ribeiro, O., Portela, I., Lopes, N.: The impact of identifiable features in ML classification algorithms with the HIKARI-2021 dataset. In: 2023 11th International Symposium on Digital Forensics and Security (ISDFS), pp. 1–5 (2023). https://doi.org/10.1109/ISDFS58141.2023.10131864
4. Ferriyan, A., Thamrin, A.H., Takeda, K., Murai, J.: Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic. Appl. Sci. **11**(17), 7868 (2021). https://doi.org/10.3390/app11177868
5. Jamalipour, A., Murali, S.: A taxonomy of machine-learning-based intrusion detection systems for the internet of things: a survey. IEEE Internet Things J. **9**(12), 9444–9466 (2022). https://doi.org/10.1109/JIOT.2021.3126811
6. KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
7. Khan, R.U., Zhang, X., Alazab, M., Kumar, R.: An improved convolutional neural network model for intrusion detection in networks. In: 2019 Cybersecurity and Cyberforensics Conference (CCC), pp. 74–77 (2019). https://doi.org/10.1109/CCC.2019.000-6

8. Kiesler, N., Impagliazzo, J.: Perspectives on the internet of everything. In: Pereira, T., Impagliazzo, J., Santos, H. (eds.) Internet of Everything, pp. 3–17. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-25222-8_1
9. Stiawan, D., Idris, M.Y.B., Bamhdi, A.M., Budiarto, R.: CICIDS-2017 dataset feature analysis with information gain for anomaly detection. IEEE Access **8**, 132911–132921 (2020). https://doi.org/10.1109/ACCESS.2020.3009843
10. Latif, S., Zou, Z., Idrees, Z., Ahmad, J.: A novel attack detection scheme for the industrial internet of things using a lightweight random neural network. IEEE Access **8**, 89337–89350 (2020). https://doi.org/10.1109/ACCESS.2020.2994079
11. Longras, A., Pereira, T., Amaral, A.: Cybersecurity challenges in healthcare medical devices. In: Pereira, T., Impagliazzo, J., Santos, H. (eds.) Internet of Everything, pp. 66–75. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-25222-8_6
12. Moritz, P., et al.: Ray: a distributed framework for emerging AI applications (2018)
13. Moustafa, N., Slay, J.: UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 Military Communications and Information Systems Conference (MilCIS), pp. 1–6 (2015). https://doi.org/10.1109/MilCIS.2015.7348942
14. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: International Conference on Information Systems Security and Privacy (2018)
15. Apache spark. https://spark.apache.org/
16. Stavrou, E.: Guidelines to develop consumers cyber resilience capabilities in the ioe ecosystem. In: Pereira, T., Impagliazzo, J., Santos, H. (eds.) Internet of Everything, pp. 18–28. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-25222-8_2
17. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A.: A detailed analysis of the KDD CUP 99 data set. In: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, pp. 1–6 (2009). https://doi.org/10.1109/CISDA.2009.5356528
18. Teixeira, D., Malta, S., Pinto, P.: A vote-based architecture to generate classified datasets and improve performance of intrusion detection systems based on supervised learning. Future Internet **14**(3), 72 (2022). https://doi.org/10.3390/fi14030072
19. Zhu, G., Hu, Q., Gu, R., Yuan, C., Huang, Y.: ForestLayer: efficient training of deep forests on distributed task-parallel platforms. J. Parallel Distrib. Comput. **132**, 113–126 (2019). https://doi.org/10.1016/j.jpdc.2019.05.001