big data and cognitive computing

MDPI

Article

Comparative Study between Big Data Analysis Techniques in Intrusion Detection

Mounir Hafsa [1,*] and Farah Jemili [2,*]

[1] Higher Institute of Computer Science and Telecom (ISITCOM), University of Sousse, Hammam Sousse 4011, Tunisia
[2] MARS Research Lab LR17ES05, Higher Institute of Computer Science and Telecom (ISITCOM), University of Sousse, Hammam Sousse 4011, Tunisia
* Correspondence: mounir_hafsa@outlook.fr (M.H.); jmili_farah@yahoo.fr (F.J.)

Received: 12 November 2018; Accepted: 15 December 2018; Published: 20 December 2018

Abstract: Cybersecurity ventures expect that cyber-attack damage costs will rise to $11.5 billion in 2019 and that a business will fall victim to a cyber-attack every 14 seconds. Notice here that the time frame for such an event is seconds. With petabytes of data generated each day, this is a challenging task for traditional intrusion detection systems (IDSs). Protecting sensitive information is a major concern for both businesses and governments. Therefore, the need for a real-time, large-scale and effective IDS is a must. In this work, we present a cloud-based, fault tolerant, scalable and distributed IDS that uses Apache Spark Structured Streaming and its Machine Learning library (MLlib) to detect intrusions in real-time. To demonstrate the efficacy and effectivity of this system, we implement the proposed system within Microsoft Azure Cloud, as it provides both processing power and storage capabilities. A decision tree algorithm is used to predict the nature of incoming data. For this task, the use of the MAWILab dataset as a data source will give better insights about the system capabilities against cyber-attacks. The experimental results showed a 99.95% accuracy and more than 55,175 events per second were processed by the proposed system on a small cluster.

Keywords: intrusion detection system; machine learning; Apache Spark; Structured Streaming; Big Data; Decision Trees; Microsoft Azure Cloud

1. Introduction

In 2017, the world experienced some of the biggest cyber threats in the internet era. From WannaCry ransomware to the Equifax attack and other data breaches of services such as Uber and Yahoo [1], a phenomenon known as "burst attacks" grew in complexity and frequency. Burst attacks are short and can happen in a small-time frame, like a few minutes. The Cisco Cybersecurity Reports show that 42% of organizations experienced this type of Distributed Denial of Service (DDoS) attack in 2017 [2].

As of 2018, the amount of data generated each day is exceeding petabytes and this includes the traces that internet users leave when they access a website, mobile application or a network. These traces or "log data" are getting enormous each day as they are being produced by not only one, but sometimes many sources. The wise use of log data can give an advantage in identifying malicious connections, thus protecting the network from future attacks. However, the short time window that hackers are using can cripple even good systems as they attack in a short period of time. The need for a real-time detection system that can scale to the amount of data being ingested and act quickly in terms of response time can give an edge over these types of attacks.

To detect and signal anomalous activities, an intrusion detection system (IDS) is used. Sig Myers et al. [3] defined IDS as a system that delivers real-time, reliable analysis of network

traffic to determine whether the network is being (or has been) attacked. Cloud computing provides processing power, storage, services and applications over the Internet [4]. The most known cloud service providers are Amazon (AWS), Microsoft Azure and Google Cloud Platform. The use of on-premises and public cloud infrastructure is growing. Cisco Reports [2] show that security is seen as a key benefit of hosting networks in the cloud, as it gives an extra layer of security.

In order to ingest and process huge amounts of data, big data tools such as Apache Hadoop [5] and Apache spark [6] are used. Hadoop is an open source project that has been the leader in the Big Data world. It uses Map-Reduce to process huge amounts of data and store it in the HDFS (Hadoop Distributed File System). Spark proved to be 10× to 100× faster than Hadoop as it uses in-memory computation. Spark supports SQL queries, streaming data, machine learning and graph processing and it can run on Hadoop or use HDFS to store data.

Based on the methods and shortcomings of existing works, we propose a new approach that aims to provide an accurate classification of cyber-attacks in real-time. The design of the proposed approach will take into consideration the current trends in cloud computing and Big Data tools. To fulfill these requirements, there is a need for an Intrusion Detection System that can handle real-time streaming data. The final product will be a real-time, Big Data framework implemented within a cloud infrastructure and tested against real-world traffic data using a Machine Learning algorithm. The rest of the paper is organized as follows. In Section 2, there is a review of the related research which deals with IDS and Cloud Computing. Moreover, Section 3 presents the proposed system and its components. Section 4 illustrates the evaluation metrics and results of the tested system. Finally, Section 5 provides conclusions and offers new possibilities for the development of future work.

2. Literature Review

Intrusion detection has always been a major concern in scientific papers [7,8]. Since the evolution of the modern Internet, with Big Data coming to the surface along with Cloud Computing, researchers are more eager to find new solutions to this dilemma. Many methods were used to detect intrusions over a network, ranging from data mining approaches to Machine Learning algorithms. Different tools were also deployed. In this section, we discuss the state-of-the-art intrusion detection techniques in Big Data.

Muhammad et al. [9] introduced a real-time network intrusion detection system based on Apache Storm. This work consists of applying the Support Vector Machine (SVM) algorithm on streaming data coming from the Knowledge Discovery Database (KDD) cup 99 dataset. The system can process up to 13,600 packets in a second on a single machine with 92.60% accuracy on testing data. Although this work gave performance metrics for one machine, it has not been tested on a multi-node cluster to monitor its performance. The lack of a distributed environment is the missing key.

Mustapha et al. [10] used Apache Spark and MLlib to test the performance of intrusion detection using four Machine Learning algorithms, namely Support Vector Machine (SVM), Naïve Bayes, Decision Tree and Random Forest, against the UNSW-NB15 dataset. Their work shows that Random Forest yields the best performance in terms of accuracy (97.49%), sensitivity (93.53%) and specificity (97.75%). This is followed by Decision Trees, while Naïve Bayes gave the worst accuracy with 74.19%. This work used Apache Spark, although only with batch processing and no stream processing to classify data.

Pallaprolu et al. [11] applied Apache Spark Streaming to detect zero-day attacks. Here, the proposed system is tested with the KNN algorithm that showed a precision of 99.57% with a True Positive Rate (TPR) of 94% and a False Positive Rate (FPR) of 3%. In addition, the system has been tested in a distributed environment to test its scalability. Although this work shows that they classify incoming data in near real-time, they only used a relatively small test data set to evaluate their system, which does not quite fit the real-world scenario.

Gupta et al. [12] introduced a Spark-based intrusion detection framework. Their system has been implemented with two feature selection algorithms: correlation-based feature selection and

Chi-squared feature selection. To evaluate their performances, they used five Machine Learning algorithms (Logistic Regression, SVM, Naïve Bayes, Random Forest and GB Tree) on NSL-KDD and DARPA 1999 dataset. Despite using Spark's batch processing mode for their work, their results show that the Random Forest classifier yields the best accuracy but the worst prediction time, while Naïve Bayes holds the worst accuracy but has a faster training/prediction time. The use of the DARPA dataset, which is relatively old and contains duplicate and unreal network traffic data, unfortunately leads to false predictions. To the best of our knowledge, the distributed test environment is missing.

Terzi et al. [13] created a new unsupervised anomaly detection approach and used it with Apache Spark on Microsoft Azure (HDInsight) to harvest Spark's scalable processing power. The new approach was tested on CTU-13 (a botnet traffic dataset) and achieved a 96% accuracy rate. The drawback of this work is that it cannot detect anomalies that were similar to normal traffic. NetFlow data that have been used in their approach are often captured by ISPs for auditing and performance monitoring purposes. Sadly, this kind of sample is lacking the raw content of network packets [14].

Casas et al. [15] have used Apache Spark Streaming (RDD version) to detect anomalies and have compared its performance with other frameworks. Frameworks that are specifically built for anomaly detection have shown good performance; better than that of Spark Streaming. Unfortunately, they used the RDD deprecated version of Spark Streaming and not the new and faster Data Frame version of Spark Structured Streaming that we will be using in this work.

Our Spark-based network Intrusion Detection System will focus mainly on detecting anomalies in streaming data coming from a daily updated dataset (MAWILab) and applying a Decision Tree classifier using a multi-node distributed cluster within Microsoft Azure (HDInsight). Thus, we provide a fault tolerant, real-time and scalable system that adapts to the velocity of data by adding and shrinking the number of machines.

3. Research Methodology

Our work involves several steps that start with ingesting data all way to predicting anomalies in real-time. For each step, different tools are used to get the best of Microsoft Azure Cloud services. The proposed system is supposed to handle a continuous stream of data coming from the Fukuda Lab website. Two major sections are discussed here. The first part tackles the extraction of dataset files, followed by the transformation of these files and finally the preparing/cleaning. The second section will handle classifying data with the help of a Machine Learning pipeline. Figure 3 shows the overall process.

3.1. Data and Methods

3.1.1. Dataset Description

Both Casas et al. [15] and Callegari et al. [16] used the MAWILab dataset. Since 2001, 15 min network traffic traces are captured on a backbone link between Japan and the US. These network traffic traces are published in both CSV and XML format.

These network traffic data are collected by the Fukuda Lab and available for research purposes. This dataset uses a combination of four anomaly detectors (Hough transform, Gamma distribution, Kullback-Leibler divergence and Principal Component Analysis (PCA)) [17]. In addition, it contains four principal categories [18]: anomalous, suspicious, notice and benign. In addition, ten fields are ordered as follows: anomalyID, srcIP, srcPort, dstIP, dstPort, taxonomy, heuristic, distance, nbDetectors and a label column. The heuristic field inspects the port number, TCP flags and ICMP codes of anomalous traffic and assigns a code to each anomaly. If the code value is lower than 500, it means the anomalous traffic is using well-known suspicious ports or it contains an abnormally high number of packets with a SYN, RST or FIN flag. Table 1 below shows twelve anomalies with a code value below 500.

Table 1. Anomalies with a code value below 500.

Code	Anomaly
1	Sasser worm
2	NetBIOS attack
3	Remote Procedure Call (RPC) attack
4	Server Message Block (SMB) attack
10	SYN attack
11	Reset (RST) attack
12	FIN attack
20	Ping flood
51	File Transfer Protocol (FTP) attack
52	Secure Shell (SSH) attack
53	HyperText Transfer Protocol (HTTP) attack
54	Hypertext Transfer Protocol Secure (HTTPS) attack
else	Other

If the code value is between 500 and 900, it means that the anomaly is seen on well-known ports. Table 2 below presents five different anomalies with a code value between 500 and 900.

Table 2. Anomalies with a code value between 500 and 900.

Code	Anomaly
501	File Transfer Protocol (FTP) traffic
502	Secure Shell (SSH) traffic
503	HyperText Transfer Protocol (HTTP) traffic
504	Hypertext Transfer Protocol Secure (HTTPS) traffic
else	Other

If the code value is higher than 900, it means the anomaly is seen on unknown ports. The code 901 represents unknown ports.

3.1.2. Apache Spark

Apache Spark is a fast-distributed engine for large-scale data processing and Machine Learning tasks. It was developed at UC Berkeley in 2009 [19]. The famous project has been adopted by many internet pioneers like Netflix, Yahoo and eBay. It is Scala based, but offers APIs for Java, Python and R. Spark has a diversified ecosystem which is shown in Figure 1. Spark Core provides the concept of resilient distributed datasets (RDDs) that enables caching data in-memory and not reading it from the disk every time. Spark Streaming enables Spark to be a real-time stream processing engine by turning incoming data into a Discretized Stream (DStream) to deliver insights in a fast and fault tolerant way. Spark uses micro-batches to process live data [20].

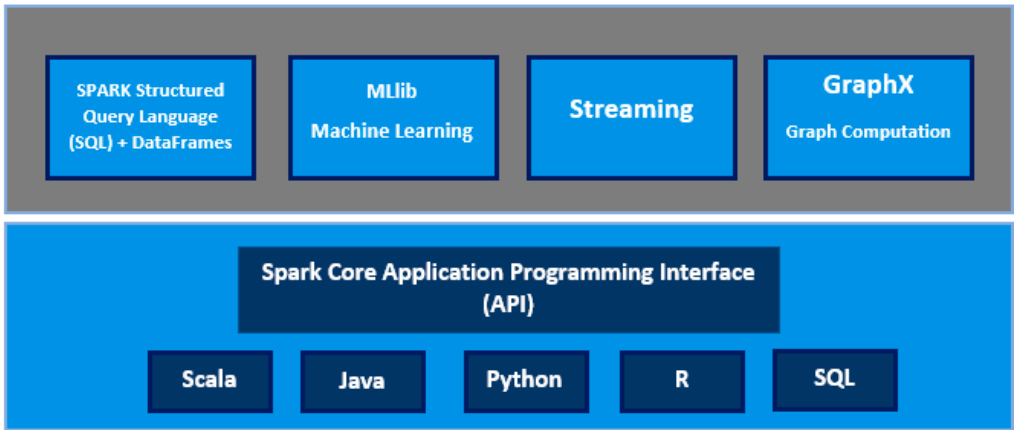

Figure 1. Apache Spark Ecosystem [19].

Spark Structured Streaming was introduced by Armbrust et al. [21] to tackle issues with streaming systems and provide an easy to use, real-time and high-performance stream processing engine built on top of the Spark SQL engine. The idea behind Structured Streaming is to treat any data feed as an unbounded table. New records added to the stream are like rows being inserted into a table [22]. Figure 2 explains the concept in a clear way. Structured Streaming outperformed other stream processing engines like Apache Flink by up to 2.9× and Apache Kafka Streams by 90× using the Yahoo! Streaming Benchmark [23,24].

Spark MLlib is a collection of popular Machine Learning and data mining algorithms. It enables predictions, recommendations and feature extraction at scale. Like Structured Streaming, Spark Machine Learning is a new data frame library. Spark Machine Learning is replacing the RDD-based MLlib due to its fast processing capabilities [22].

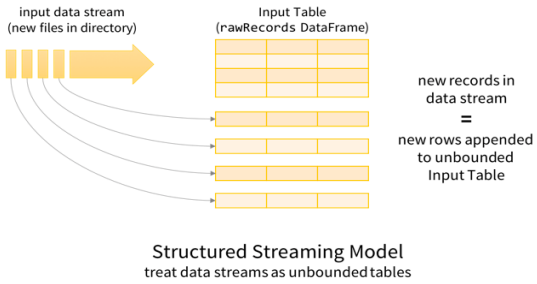

Figure 2. Structured Streaming Model [25].

3.1.3. Microsoft Azure

Microsoft Azure, formally known as Windows Azure, is a cloud computing platform for building, deploying and managing services and applications anywhere with the help of a global network of managed data centers located in 54 regions around the world [26]. It offers software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS). Customers can use any programming language, tool and framework with a growing marketplace of services [27].

Microsoft's HDInsight is a managed Hadoop service in Azure Cloud that uses the Hortonworks Data Platform (HDP). HDInsight clusters can be customized easily by adding additional packages and can scale up in case of high demand by allocating more processing power. The cluster consists of different types of virtual machines with the possibility of storing data separately in Azure Blob Storage or Azure Data Lake instead of HDFS. The data managed by Azure is protected by the Azure Active Directory and persists even after the cluster is deleted.

3.1.4. Decision Tree Classifier

Decision Trees are a supervised Machine Learning algorithm that is known to perform well if it is well configured. It supports both numerical and categorical data and it can handle large data sets. A Decision Tree can be represented with nodes and edges, and it is composed of a root node that performs the first split and leaf nodes in which we can find the predicted results. Generally, splitting is based on entropy and information gain.

In this work we used decision trees due to its simplicity and effectiveness. After testing off-the-shelf classification algorithms, we found the following results: Naïve Bayes gave around 70% accuracy, Random Forest gave 98.81% accuracy and Decision Trees gave 99.23% accuracy.

3.2. Extract, Transform and Prepare

Extracting data: This process will involve collecting dataset files from the Fukuda Lab website and loading it into Microsoft Azure Blob Storage [28]. This service is built to meet HDFS standards and use global and local replication with exabytes of capacity and massive scalability along with enterprise grade security. Two tools are used here: Microsoft Azure SDK for Python [29], a set of Python packages that make it easy to access components of Microsoft Azure, and Beautiful Soup, a Python library for pulling data out of HTML and XML files.

Beautiful Soup makes it easier to pull new data directly from the Fukuda Lab and upload it into Azure Blob Storage. Using a publish-subscribe messaging system like Apache Kafka or Azure Service Bus would be more practical, but unfortunately the website does not provide a single directory for published files, but instead each file is within a separate HTML page containing both CSV and XML files. The use of a web scraper to collect published files seemed more practical in the absence of a unified directory.

The remaining operations will be conducted using a Microsoft HDInsight cluster running Spark Structured Streaming. As mentioned before, Structured Streaming is built on top of the Spark SQL engine, which gives us many advantages including:

1. Exactly once delivery—this means that the message is only delivered once. It is neither getting lost or duplicated.
2. Providing end-to-end reliability with Structured Streaming and achieving fault tolerance is done by specifying a checkpoint directory where all metadata is saved. Besides, even if the entire cluster fails, work can be restarted on a new cluster. Spark supports many data sources such as file source, Apache Kafka, socket and rate source [25].

Transforming data: We proceed with the creation of an HDInsight Spark Cluster in which we can perform data manipulation in a distributed manner. We will be using the Python API (PySpark) and Jupyter Notebook with Apache Spark. In this step, we will be reading CSV files as a stream and converting them to Apache Parquet format. Listing 1 shows the code used to read CSV files as a stream. Since Apache Spark supports multiple operations on data, it offers the ability to convert data to another format in just one line of code. Developed by Twitter and Cloudera, Apache Parquet is an open-source columnar file format that is optimized for query performance and minimizing I/O, offering very efficient compression and encoding schemes [30]. Table 3 shows the efficiency and effectiveness of using Parquet format. We notice that by converting CSV to Parquet, both cost and performance are improved. This format minimizes not only the time wasted on waiting for data to be scanned and processed, but also storage costs. The MAWILab dataset is updated daily. Any new file added to their website will be immediately ingested by our web scraper. As different file sizes can be published, we used an average file size to determine an approximate size gain for each new file. Table 4 shows the old and new size after converting to Apache Parquet. Listing 2 shows the code used to export CSV to a specified sink with the new format 'Parquet'.

Table 3. Savings and speedup with Apache Parquet [30].

Dataset	Size on Amazon S3	Query Run Time	Data Scanned	Cost
Data stored as CSV files	1 TB	236 s	1.15 TB	$5.75
Data stored in Apache Parquet format	130 GB	6.78 s	2.51 GB	$0.01
Savings/Speedup	87% less using parquet	34× faster	99% less data scanned	99.7% savings

Table 4. Average file size before and after converting.

Average Size (CSV)	Average Size (Parquet)	Speedup
9.5 Kb	6.5 Kb	×1.46

Preparing data: This is a time-consuming task due to its importance to the overall performance. Choosing the right features for our machine learning model and dealing with missing fields can give a boost to the accuracy of our system. This task involves three different approaches consisting of feature selection, filling missing values and finally removing duplicates. We proceed by reading Parquet files as a stream like we did with CSV files. We can specify the option "maxFilesPerTrigger" at the start to limit how many files are collected with each trigger. It is set to maximum by default.

The first approach is feature selection, which limits the size of the dataset as well the computational power. According to Ullah et al. [31], using all features is inefficient. Therefore, specific features that contribute to the detection process are selected.

Unnecessary features like anomaly_id and label (empty column) are deleted to minimize the computational process. We did not use the feature selection algorithm for this task, but instead selected columns that were relevant. After removing anomaly_id and label, we end up with a total of eight columns. Since we have a low number of features and a high number of historical samples to train our Machine Learning algorithm, we will not be applying a feature selection algorithm to avoid unnecessary computations.

Listing 1. Spark read stream.

```
streamingInputDF = (spark.readStream.schema(FSchema).
option("header",True).csv(Source_path_CSV))
```

The second approach involves filling missing values. Data is naturally messy and sometimes missing fields are found in big datasets. To overcome this issue, we suggest a simple yet effective approach. Instead of deleting records containing missing fields, we proposed filling them with default values if known as an effort to make the dataset complete.

Listing 2. Spark write stream.

```
query = (streamingInputDF.writeStream.format("parquet")
.option("checkpointLocation","/checkpoint_location")
.option("Path","/Parquet_Files_Path/").start())
```

Getting rid of duplicates was our third approach. The removal of redundant records helps with attack detection as it makes the system less biased by the presence of more frequent records. This tactic makes computation faster as it must deal with less data [32]. Spark's Structured Streaming API provides a solution to remove duplicate rows from a continuous stream of data by using a unique identifier column. Spark will store the necessary amount of data from previous records, so it can filter duplicate records.

3.3. Real-Time Classification

After preparing the data, we proceed by loading our Machine Learning pipeline model that contains a Decision Tree classifier and we transform incoming data to obtain predictions for each record. Listing 3 below demonstrates the operation of loading the model.

Listing 3. Loading Machine Learning (ML) model.

```
model = PipelineModel.read().load("Path_to_ML_Model")
dflive = model.transform(LiveData)
```

The final step is writing predictions to a sink/output location, which is done by specifying the format of the output data (ex: Parquet, JSON, etc.), a checkpoint location to assure fault tolerance and finally an output sink. Other options can be added. Listing 4 below demonstrates the operation of writing predictions to Parquet format. The start() command must be specified to begin the streaming job. During the streaming job, Spark will ingest, classify and write each file to the chosen sink. The sink can be a file directory, a database or even another Spark job. To stop reading streaming data, we use the query.stop() command. In order to understand the concept of a Machine Learning pipeline, we demonstrate in the next part the steps used to create, train and save our pipeline using Apache Spark Machine Learning.

Listing 4. Loading ML model.

```
query = (dflive.select('srcIP', 'srcPort', 'dstIP', 'dstPort','label', 'label_ix','prediction') .writeStream.format
("parquet").option("checkpointLocation","Checkpoint_location_Path"). option("Path","Predictions_
Path").start())
```

3.4. Machine Learning Pipeline

In this section, we show the creation of a Machine Learning pipeline using the Apache Spark Machine Learning library. We first introduce the key components of the pipeline. Next, we introduce the workflow and steps taken to create the pipeline. Finally, we describe the detection schemes that we implemented and integrated in the system. A pipeline is a sequence of stages where each stage is either a Transformer or an Estimator. These stages are run in order. The input will be a data frame that is transformed as it passes through each stage. In our work, we used two transformers: a StringIndexer and a VectorAssembler.

1. The StringIndexer encodes a string column of labels to a column of label indices. The indices are ordered by label frequencies, so the most frequent label gets index 0.
2. The VectorAssembler is a transformer that combines a given list of columns in a single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector in order to train Machine Learning models like Logistic Regression and Decision Trees.

The final stage in our pipeline was adding a Decision Tree classifier. Following this, we used a portion of historic data collected from the datasets to train our pipeline model. The data used will pass through each mentioned stage above. Often, it is worth it to save a model or a pipeline to a disk for later use.

In Spark, pipeline import/export functionality is supported and this enables us to save a trained pipeline or Machine Learning model for future use. As shown in Figure 3, we have a pipeline model saved in Azure Blob Storage. The process of creating a pipeline model is done offline. Our model can be updated manually to train on new data and can be exported again to replace the old model. In the next chapter, we discuss the implementation of the proposed system in Microsoft Azure and evaluate its performance.

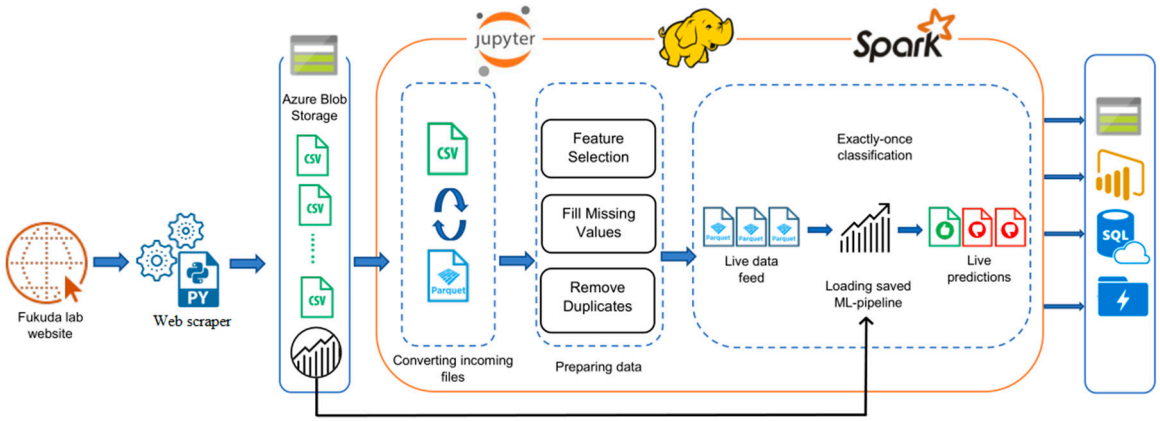

Figure 3. Diagram of proposed approach.

4. Experimentation Results

4.1. Experiment Setup

The environment chosen to perform the tests is Microsoft Azure as it provides HDInsight, a managed Hadoop distribution, so that we can test the system in a realistic, real-world environment and use multiple nodes in our Spark cluster to ensure distributed computing.

To conduct our experiments, we used Microsoft Azure HDInsight running on Linux virtual machines and Spark version 2.2.0 running on top of YARN, using Jupyter Notebook and Python API (Pyspark). This cluster is equipped with two 2 head nodes and two 2 worker nodes. Characteristics are listed in Table 5.

Table 5. Setup used to test our approach.

	Head Node	**Worker Node**
Name	D3 V2 optimized	D4 V2 optimized
Number	2	2
CPU	4 vCPUs	8 vCPUs
Memory (RAM)	14 GB	28 GB
Storage	200 GB SSD	400 GB SSD
Operating System (OS)	Linux (CentOS) ×64 bit.	Linux (CentOS) ×64 bit.
Cost	$0.229/h	$0.458/h

4.2. Performance Metrics

Apache Spark Machine Learning provides a suite of metrics to evaluate the performance of Machine Learning models [33]. The metrics used in our work to measure the performance of the pipeline are as shown in Table 6.

Table 6. Evaluation metrics.

Measure	**Description**	**Formula**
Accuracy	Accuracy measures precision across all labels	$AC = \frac{(TP+TN)}{TP+FP+TN+FN}$
Precision	Proportion of correct labels that were classified over all labels	$P = \frac{TP}{TP+FP}$
Recall	Proportion of correct labels that were classified correctly over all positive labels	$R = \frac{TP}{TP+FN}$
F-measure	Harmonic average of Precision and Recall	$FM = 2\,\frac{P*R}{P+R}$

Where TP = True Positives, TN = True Negatives, FP = False Positives and FN = False Negatives. The results generated by the pipeline can be seen in Table 7 below:

Table 7. Proposed system results.

Accuracy	Precision	Recall	F1 Score
99.95%	99.91%	99.95%	99.93%

Using a Decision Tree in this experiment showed great results as indicated in the previous table. For each trace in the MAWILab dataset, there are two CSV files. One file is for traffic labeled as anomalous or suspicious; the other file is for traffic labeled as notice.

In this evaluation we will analyze the MAWILab traffic traces to determine and find some insights. The MAWILab dataset contains IP addresses for both source and destinations. For this, we will try to find out the top IP addresses that generate attacks as well as the top IP addresses that were attacked. In addition, the dataset provides information about the source and destination port. We will then try to determine which port was used to produce attacks and which port was attacked the most.

As discussed in Section 3.1.1, the MAWILab dataset contains a taxonomy field that provides a detailed classification of each captured event in the dataset. The taxonomy can be divided into two major categories, anomalous and normal, where anomalous concerns any event that includes denial of service (DoS), distributed denial of service (DDoS), port scan and network scan, whether it is ICMP, UDP or TCP. Normal traffic contains heavy hitter, point-multipoint and others.

The number of events in the overall dataset collected from 2007 to September 2018 contains more than 1,873,545 events. After running our stream job and Spark Structured Streaming and applying a deduplication operation to remove duplicate records, we ended up with a total of 560,808 events. Anomalous events present more than 41.68% of the dataset, while suspicious events present 58.31% of the dataset. Table 8 shows the top taxonomies found in the anomalous traffic. These top six represent 93% of the overall found taxonomies. Other attack types such as DoS and DDoS represent only 1.74% of anomalous traffic. We can see that DDoS and DoS attacks are not frequent, but HTTP attacks and Network Scans occurred frequently. In Table 9 we notice that the most used ports in this dataset for both anomalous and suspicious traffic are: (1) unknown ports, then followed by (2) port 80 (HTTP), (3) 53 (DNS) and (4) 443 (HTTPS).

Table 8. Number of attacks in anomalous traffic. DoS = denial of service; DDoS = distributed denial of service.

Attack	Multi-Points	HTTP	Network Scan (TCP)	Alpha Flow	DDoS	DoS
Count	88,429	67,063	23,281	12,427	4162	1928

Table 9. Top 5 source ports with anomalous events.

Port Number	0 (Unknown)	80	443	53	6000
Count	100,656	65,368	18,569	9531	5906

Ivanov et al. [34] shows important metrics to evaluate Apache Spark streaming performance. Since our work is continuously collecting data generated by the Fukuda Lab, two metrics are used: processed records per second and input rows per second. Table 10 shows the description of each metric.

Table 10. Metrics used to evaluate our system.

Metric	Description	Result
Input Rate	Describes how many rows were loaded per second.	555,470 rows per second
Processing Rate	Describes how many rows were processed per second.	55,175 rows per second

When first launched, Spark processed all files published since 2007 to the time when the tests were performed on 24 July 2018. More than 555,470 rows were collected, each file containing an average of

163 rows. Our proposed system was able to process 55,175 records in a second and collected all files in one batch. This was possible due to setting the maxFilesPerTrigger to max, which can be limited to one file or many collected at a time. On 25 July 2018, our system automatically ingested a newly published file and predictions were made with the same accuracy score. To test the scalability of our system, we added another worker node with the same hardware. The performance of our cluster increased to achieve a processing rate of 80,517 rows per second. The performance of our Spark Cluster depends on two factors:

1. The first factor is the size of the cluster. Although it still varies with the workload, the more processing power we allocate, the more data the system can process.
2. The second factor is the incoming rate of data. If it is more than what the system can process, it creates a bottleneck and an intervention is needed to limit the size of the input rate.

In general, this system achieves good classification results along with a good processing speed that not only meets real-world scenarios, but also can be increased by adding more machines to the cluster. Apache Spark Structured Streaming provides important functionalities such as dealing with streaming data. These features make it possible to modify the structure of data, fill missing fields or drop duplicate records on the fly. Aside from being fault tolerant and distributed, Apache Spark was able to read and write data from different sources in different formats.

The Decision Tree algorithm showed good classifications. This was possible due to the Spark Machine Learning library that made the implementation of Machine Learning algorithms easier. Decision Trees proved to be reliable when dealing with binary classification against streaming data. While using a Decision Tree in this experiment showed great results, unfortunately one of the disadvantages of Decision Trees is that each tested object will have only one leaf associated with it. Also, missing or inaccurate data results lead to false classifications. Accordingly, Fuzzy Representation is becoming popular in dealing with the problems of missing and inexact data. This leads to Fuzzy Decision Trees (FDTs) that combine both the readability of Decision Trees with the Fuzzy Representation of missing data [35]. FDT helps in situations where data is missing or inexact, for example in intrusion detection.

5. Conclusions and Future Work

In this paper, we discussed the need for a fast, real-time Intrusion Detection System to handle evolutive traffic and provide classifications to protect a network. Our proposed system uses Apache Spark Structured Streaming to process and detect anomalies in real-time. Besides being distributed, scalable and fault tolerant, this system is used with Microsoft Azure to showcase its performance within a distributed cloud infrastructure. We used the MAWILab dataset to evaluate the proposed system against cyber-threats. Based on our experimentation, our Spark-based IDS yields a 99.95% accuracy using a Decision Tree classifier. Achieving good accuracy was not our only top priority; this system can also process more than 55,175 records in one second using only a two worker-nodes cluster. Hence, the processing rate can achieve higher rates by allocating more nodes to the cluster.

There are some issues in our approach that could be improved or fixed in the future. First, the speed of data processing could be improved by using the newly introduced feature of Spark Structured Streaming: continuous streaming. This makes it possible to process data in milliseconds (1 ms) and not just seconds. Another improvement would be using Fuzzy Decision Trees (FDTs) [35] to classify data and not traditional Decision Trees. Another issue we noticed is the advantages of using more than one evaluation data set. Another potentially useful new feature introduced in Apache Spark is called Stream-to-Stream joins. This makes joining two different streams of data into one stream possible. Using this feature to merge data sets will improve the evaluation schemes [36].

For future work, we aspire to test another Big Data framework such as Apache Beam. It is expected to give interesting results. Using Deep Learning is trendy and it is widely used because it has the potential to extract better representations from the data to create much better models, and it is

inspired by recurrent neural networks [37]. Combining both to produce a Deep Learning approach based on Big Data technologies for intrusion detection will give an edge over the commonly used Machine Learning techniques.

Author Contributions: M.H. performed literature review and experiments including data collection, preprocessing and implementation of proposed approach. This work was supervised by F.J. whom also verified the writing of the original draft.

Funding: This research received no external funding.

Acknowledgments: The authors would to thank the reviewers for their valuable suggestions and comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Top-5-Cybersecurity-Concerns-for-2018. Available online: https://www.csoonline.com/article/3241766/cyber-attacks-espionage/top-5-cybersecurity-concerns-for-2018.html (accessed on 23 June 2018).
2. Cisco Cybersecurity Reports. Available online: https://www.cisco.com/c/en/us/products/security/security-reports.html#~{}stickynav=2 (accessed on 9 August 2018).
3. Myers, S.; Musacchio, J.; Bao, N. *Intrusion Detection Systems: A Feature and Capability Analysis*; Baskin School of Engineering: Santa Cruz, CA, USA, 2010.
4. Stergiou, C.; Psannis, K.E.; Byung-Gyu, K.; Brij, G.B. Secure integration of IoT and Cloud Computing. *FuTure Gener. Comput. Syst.* **2018**, *78*, 964–975. [CrossRef]
5. Apache Hadoop. Available online: www.apache.com/hadoop (accessed on 8 December 2018).
6. Apache Spark. Available online: www.apache.com/spark (accessed on 8 December 2018).
7. Ar, L.; Levent, E.; Vipin, K.; Aysel, O.; Jaideep, S. A comparative study of anomaly detection schemes in network intrusion detection. In Proceedings of the SIAM Conference on Applications of Dynamical. Systems, Snowbird, UT, USA, 27–31 May 2003.
8. Massimiliano, A.; Erbacher, R.F.; Jajodia, S.; Persia, M.C.F.; Picariello, A.; Sperli, G.; Subrahmanian, S.V. Recognizing unexplained behavior in network traffic. *Netw. Sci. Cybersecur.* **2013**, *55*, 39–62.
9. Manzoor, M.A.; Morgan, Y. Real-time Support Vector Machine based Network Intrusion Detection system using Apache Storm. In Proceedings of the IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 13–15 October 2016.
10. Belouch, M.; el Hadaj, S.; Idhammad, M. Performance evaluation of intrusion detection based on machine learning using Apache Spark. *Procedia Comput. Sci.* **2018**, *127*, 1–6. [CrossRef]
11. Pallaprolu, S.C.; Sankineni, R.; Thevar, M.; Karabatis, G.; Wang, J. Zero-Day Attack Identification in Streaming Data Using Semantics and Spark. In Proceedings of the IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 25–30 June 2017.
12. Gupta, G.P.; Kulariya, M. A Framework for Fast and Efficient Cyber Security Network Intrusion Detection Using Apache Spark. *Procedia Comput. Sci.* **2016**, *93*, 824–831. [CrossRef]
13. Terzi, D.S.; Terzi, R.; Sagiroglu, S. Big data analytics for network anomaly detection from netflow data. In Proceedings of the International Conference on Computer Science and Engineering (UBMK), Antalya, Turkey, 5–8 October 2017.
14. Cisco Systems NetFlow Services Export Version 9. Available online: https://tools.ietf.org/html/rfc3954 (accessed on 2 June 2018).
15. Casas, P.; Soro, F.; Vanerio, J.; Settanni, G.; D'Alconzo, A. Network security and anomaly detection with Big-DAMA, a big data analytics framework. In Proceedings of the IEEE 6th International Conference on Cloud Networking (CloudNet), Prague, Czech Republic, 25–27 September 2017.
16. Callegari, C.; Giordano, S.; Pagano, M. Statistical Network Anomaly Detection: An Experimental Study. In Proceedings of the International Conference on Future Network Systems and Security, Paris, France, 23–25 November 2016.
17. Fontugne, R.; Borgnat, P.; Abry, P.; Fukuda, K. MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT), Philadelphia, PA, USA, 30 November–3 December 2010.

18. Fukuda Lab. Documentation. Available online: http://www.fukuda-lab.org/mawilab/documentation.html (accessed on 8 December 2018).
19. Datariicks. About Databricks. Available online: https://databricks.com/spark/about (accessed on 6 May 2018).
20. Zubair, N. *Pro Spark Streaming the Zen of Real-Time Analytics Using Apache Spark*; Apress: Berkeley, CA, USA, 2016.
21. Armbrust, M.; Das, T.; Torres, J.; Yavuz, B.; Zhu, S.; Xin, R.; Ghodsi, A.; Stoica, I.; Zaharia, M. Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In Proceedings of the International Conference on Management of Data, Houston, TX, USA, 10–15 June 2018.
22. Real-time Streaming ETL with Structured Streaming in Apache Spark 2.1. Available online: https://databricks.com/blog/2017/01/19/real-time-streaming-etl-structured-streaming-apache-spark-2-1.html (accessed on 6 May 2018).
23. Benchmarking Structured Streaming on Databricks Runtime against State-of-the-Art Streaming Systems. Available online: https://databricks.com/blog/2017/10/11/benchmarking-structured-streaming-on-databricks-runtime-against-state-of-the-art-streaming-systems.html (accessed on 7 May 2018).
24. Yahoo Streaming Benchmarks. Available online: https://github.com/yahoo/streaming-benchmarks (accessed on 8 December 2018).
25. Apache. Structured Streaming Programming Guide. Available online: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html (accessed on 7 June 2018).
26. Microsoft. Azure Regions. Available online: https://azure.microsoft.com/en-us/global-infrastructure/regions/ (accessed on 5 May 2018).
27. What is Microsoft Azure and Why Use It? Available online: https://www.sumologic.com/resource/white-paper/what-is-microsoft-azure-and-why-use-it/ (accessed on 5 May 2018).
28. Microsoft. Azure Storage Blobs. Available online: https://azure.microsoft.com/en-us/services/storage/blobs/ (accessed on 8 December 2018).
29. Microsoft. Azure SDK for PYTHON. Available online: https://github.com/Azure/azure-sdk-for-python (accessed on 8 December 2018).
30. Apache Parquet vs. CSV Files—DZone Database. Available online: https://dzone.com/articles/how-to-be-a-hero-with-powerful-parquet-google-and (accessed on 6 February 2018).
31. Ullah, F.; Babar, M.A. Architectural Tactics for Big Data Cybersecurity Analytic Systems: A Review. *arXiv* **2018**, arXiv:1802.03178.
32. Verma, R.; Kantarcioglu, M.; Marchette, D.; Leiss, E.; Solorio, T. Security Analytics: Essential Data Analytics Knowledge for Cybersecurity Professionals and Students. *IEEE Secur. Priv.* **2015**, *13*, 60–65. [CrossRef]
33. Mllib Evaluation Metrics. Available online: https://spark.apache.org/docs/2.1.0/mllib-evaluation-metrics.html (accessed on 3 June 2018).
34. Ivanov, T.; Taaffe, J. Exploratory Analysis of Spark Structured Streaming. In Proceedings of the International Conference on Performance Engineering, Berlin, Germany, 9–13 April 2018.
35. Gaied, I.; Jemili, F.; Korbaa, O. Intrusion detection based on Neuro-Fuzzy classification. In Proceedings of the 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA), Marrakech, Morocco, 17–20 November 2015.
36. Essid, M.; Jemili, F. Combining intrusion detection datasets using MapReduce. In Proceedings of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Budapest, Hungary, 9–12 October 2016.
37. Li, Z. *A Neural Network Based Distributed Intrusion Detection Sysem on Cloud Platform*; The University of Toledo: Toledo, OH, USA, 2013.