Enhancing Spark ML Workflows for Malicious Network Traffic Identification with Google Cloud Dataproc

I. Introduction

In the current era of information technology, the widespread deployment of Internet of Things (IoT) devices has intensified the challenges of network security. Since IoT devices often lack adequate security protection, they are easy targets for cyber attacks. To effectively enhance the accuracy of malware detection and provide detailed data support for network security management, we have developed an application based on machine learning algorithms. This system can analyze the input network traffic data and accurately classify it into 10 different categories (Fig 1), each corresponding to different network traffic characteristics and potential threat levels, thereby greatly enhancing our ability to respond to and prevent network security threats.

Our project accepts user-uploaded network traffic data as input, which must be stored in CSV file format and comply with the arrangement format specified in Fig 2. At the backend, we utilize a pre-trained machine learning model to detect and classify the data. Once the computational tasks are completed, the system outputs the classification results, clearly indicating which category in Fig 1 the data belongs to. By using our project, users can effectively identify and categorize network traffic, thus accurately recognizing and addressing potential network security threats.

To ensure the timeliness and accuracy of the model, users can choose whether to provide data to assist us in updating the model when uploading network traffic data. The information uploaded by users who opt to provide data will be incorporated into our dataset. We will regularly retrain the model weekly using these updated datasets to maintain the model's timeliness and efficiency. This practice ensures that the model can continuously adapt to new network environments and threats, thereby providing more accurate analytical results.

Labels	Description
Attack	This label signifies the occurrence of an attack originating from an infected device directed towards another host.
Benign	This label denotes connections where no suspicious or malicious activities have been detected.
……	……
PartOfAHorizontalPortScan	This label is employed when connections are involved in a horizontal port scan aimed at gathering information for potential subsequent attacks.
Torii	This label is used when connections exhibit traits indicative of the Torii botnet.

Fig. 1. Labels

Field Name	Description	Type
Ts	The timestamp of the connection event.	time
uid	A unique identifier for the connection.	string
id.orig_h	The source IP address.	addr
id.orig_p	The source port.	port
……	……	……
resp_ip_bytes	The number of IP bytes sent from the destination to the source.	count
tunnel_parents	Indicates if this connection is part of a tunnel.	set[string]

Fig. 2. Field Name

II. Performance Analysis

Considering that our project is data-intensive, we identified the following three bottlenecks in our project.

A. Performance Bottleneck

1) **Hardware runtime resources limitations:**
   - *CPU:* In local environments with limited CPU resources, especially when working with large-scale datasets, Spark ML tasks can become CPU-intensive. The limited number of CPU cores may slow down the computation.
   - *Memory (RAM):* The limited memory capacity of the local machine may lead to frequent rubbish collection or even memory overflow errors. SparkML operations typically require large amounts of memory for data processing and memory storage.
2) **Data size and storage capacity limitations:**
   - *Data size:* When the data is too large, it is likely that the local machine will not be able to load all the provided data sets, which may cause the project to fail to run.
   - *Storage Capacity:* Local machines typically have limited storage capacity compared to cloud solutions such as Google Cloud Storage. Large datasets may not fit on local drives.
3) **Single-node execution limitations:**
   - Running Spark on a single node limits parallelism and scaling. Operations that could be distributed across multiple nodes in a cluster will be limited by the single-node execution environment.

B. Major Steps of Computation Tasks

Our project consists of the five steps as follows:

1) **Step 1: Data loading and preprocessing**
   In this step, we read the configuration file to extract numeric and categorical features, at the same time, we use Spark to read files and perform preliminary data processing, such as converting timestamps and replacing missing values. In this step, the Disk I/O usage and memory usage are high when large data is processed.
2) **Step 2: Feature processing**
   In this step, we preprocess the data, including removing invalid features, processing rare categories, generating new features, and splitting the data set into training and test sets, along with feature indexing and assembly. This step requires a large amount of memory to store intermediate data and processing results.
3) **Step 3: Model training**
   In this step, we use the training dataset to train the random forest model and tune the model parameters. In this step, the main pressure is on the CPU and memory, which is required to store training data, model parameters, and intermediate calculations during training.
4) **Step 4: Testing and evaluation**
   In this step, we use the testing dataset to predict the trained model, and at the same time, we calculate the evaluation indicators of the model such as ROC, AUC. In this step, prediction and evaluation metrics are calculated during the evaluation of model performance, and CPU utilization is high.
5) **Step 5: Model saving**
   In this step, we save the trained model to the specified path. In this step, saving the model involves writing operations, increases Disk I/O utilization, and takes up disk storage space.

Table I demonstrates the performance analysis for each step.

III. Cloud Solution

A. Cloud Service Selected

The Google Cloud Platform was chosen as our cloud solution. The platform offers a range of managed services and powerful computing resources that that were particularly suited to our projects needs.

1) **Google Cloud Dataproc**:
   Dataproc is a Google Cloud Platform's managed service that could provide automated cluster management capabilities, designed to facilitate the execution of Apache Spark and Apache Hadoop clusters for handling large-scale data workloads. As it simplifies the process of cluster creation, management, and scaling, it can allow users to focus on data analysis rather than infrastructure management. Dataproc offers flexible pricing options, integrates with other services on Google Cloud, provides high-performance computing capabilities, and is compatible with popular open-source big data frameworks, making it an ideal platform for processing and analyzing large datasets.
2) **Google Cloud Storage(GCS)**: Google Cloud Storage (GCS) is a scalable, fully managed object storage service provided by Google Cloud Platform. It offers highly available, persistent, and secure storage for large-scale data across a global network, enabling quick data access and processing. GCS is ideal for storing unstructured data, such as images, videos, backups, and log files. It is commonly used for machine learning projects to store training data, models, and checkpoints within Cloud Storage buckets. The service supports storing any amount of data and allows for frequent retrieval, making it a versatile choice for various data storage needs.

B. Advantages of Google Cloud Platform:

As our project revolves heavily around pyspark, this service allows us to run our project with more resources spread over multiple machines. Dataproc's simpler setup, only requiring the cluster to be configured and our files to be uploaded before running the project.

Dataproc and GCS in Google Cloud Platform exhibits advantages and features in comparison to other prospective cloud services, which may help to solve our performance bottlenecks:

1) **Hardware runtime resources limitations**:
   Dataproc allows for the easy creation and management of clusters of various sizes. Dataproc also supports elastic scaling, enabling you to add more computing resources to the cluster on-demand. When faced with limitations in CPUs and RAM, the user can dynamically adjust the cluster size to allocate more resources to handle the workload. For example to increase the number of worker nodes in the cluster to distribute the workload across more CPUs and RAM, ensuring efficient processing of large machine learning workloads.

   Dataproc optimizes Hadoop and Spark configurations to efficiently use resources within the Google Cloud environment. This can ensure that the cluster utilizes CPUs and RAM effectively without unnecessary overhead. This optimization helps maximize resource utilization and minimizes the impact of hardware limitations on ML training performance.

   Built-in automation and optimization tools help maximize resource utilization, reducing job runtime and operational overhead. With automated resource management, you can efficiently allocate CPUs and RAM based on workload requirements, effectively addressing hardware runtime resource limitations. With Dataproc's optimized configurations and resource management, SparkML tasks run more efficiently, leading to faster model training and evaluation.
2) **Data size and storage capacity limitations:**
   GCS offers virtually unlimited storage capacity, allowing users to store and manage large datasets without worrying about running out of space. User can scale

TABLE I
RESOURCE USAGE FOR EACH STEP

	CPU	**Disk I/O**	**Memory (RAM)**	**Disk Storage**
Step 1: Data loading and preprocessing	Medium	High	High	Medium
Step 2: Feature processing	Medium	Low	Medium	Medium
Step 3: Model training	High	Low	High	Low
Step 4: Testing and evaluting	High	Low	High	Low
Step 5: Model save	Low	Medium	Low	High

storage needs seamlessly as data grows during the application. Moreover, for large-data-size, GCS provides high throughput and low latency for data storage and retrieval, enabling Spark jobs to access and process large-scale data more quickly. GCS supports parallel read and write operations, which significantly boosts data processing speeds, especially when handling large datasets.

3) **Single-node execution limitations:** Dataproc allows us to create scalable clusters with multiple nodes (both master and worker nodes). This distributes the workload across multiple machines, enabling parallel execution of tasks. The advantages are the same as 'Hardware runtime resources limitations' part above.

Google Cloud's Dataproc stood out against the other cloud services providers due to it's ease of use when working with pyspark. Other Cloud Platforms such as Tencent Cloud or Aliyun don't provide a simple spark or pyspark deployments, whereas AWS and Azure only provide spark deployments. Using any of these other platforms would require lots of manual configuration and debugging. Google cloud also provided straightforward integration with its cloud storage solution which makes accessing files as simple as providing a URI to the file. Dataproc also automatically scales our project for us so if we work on larger datasets we don't have to come back an manually update our configuration.

C. Details about Selected Cloud Service

We made use of Google's *Dataproc* service. This service allows us to configure, deploy, and scale Apache Hadoop, Spark and many other tools and frameworks. It's ability to deploy a Apache Spark cluster was the most appealing to us as our project makes extensive use of pyspark.

TABLE II
(VIRTUAL) MACHINE COMPARISONS

	RAM	CPU V	CPU Platform	Cluster Type
Local	16GB	4	Intel Core	None
Cluster1	32GB/node	8	AMD Milan	1 Master 3 Workers
Cluster2	8GB/node	1	Intel Cascade Lake	1 Master 2 Workers
Cluster3	32GB/node	8	Intel Cascade Lake	1 Master 3 Workers

1) VM configuration: Initially we deployed a single small cluster of 8vCPUs to ensure that *Dataproc* was a suitable service. Preliminary results showed promising performance improvements. Ultimately we decided on a 4 node cluster. 1 master and 3 worker nodes. Each node was configured with 8vCPUs and 32 GB of ram. This gave us the performance of 32vCPUs cores for our project.

We deployed 3 clusters, 2 of similar configuration. Due to cloud quotas and limitations we were unable to deploy 2 identical clusters to test our project on. The details can be see in the Table II.

2) Estimated Budget: Google advertises it's *Dataproc* services to be a low-cost solution costing only 1 cent per vCPU per hour. With our project running, the configured cluster would cost us an estimated 32 cents per hour. We expect our project to finish analysing all the data after 8 hours giving us an estimated cost of 2.56 dollars for our entire dataset.

IV. RESULT AND ANALYSIS

A. Performance Comparison

TABLE III
PERFORMANCE COMPARISONS

File Num	Local	Cluster 1	Cluster 2	Cluster 3
1	25min	12min54s	24min5s	-
2	DNR	17min2s	-	-
12	DNR	-	-	4h13min

Table III provides a performance comparison of datasets containing different numbers of files processed in a local environment and three different cluster environments (Cluster 1, Cluster 2, Cluster 3). The datasets are clearly distinguished as containing one file (File Num 1), two files (File Num 2), and a dataset of 12 files (File Num 12). In processing a single file, the local environment takes 25 minutes, while Cluster 1 significantly reduces this time to 12 minutes 54 seconds, and Cluster 2 takes 24 minutes 5 seconds; Cluster 3 did not perform this test.

For the dataset containing two files, the local environment did not complete the processing, marked as DNR (Did Not Run), due to insufficient system resources such as CPU computing power and RAM. Meanwhile, Cluster 1 was able to complete the processing in 17 minutes 2 seconds, while Cluster 2 and Cluster 3 did not conduct this test. In terms of processing the dataset containing 12 files, the local environment, Cluster 1, and Cluster 2 also did not complete the processing, similarly marked as DNR. Only Cluster 3 completed the processing, taking 4 hours 13 mins.

B. Analysis

The primary bottlenecks identified in the project are:

1) **Hardware Runtime Resource Limitations**: Limited CPUs and RAM especially for large-scale dataset.

2) **Data Size and Storage Capacity Limitations**: Inability of local storage to handle large datasets.
3) **Single-node Execution Limitations**: Limited parallelism and scalability when using a single node.

We have successfully addressed the bottlenecks we have identified with GCP Dataproc Cloud Solution.

1) **Hardware runtime resources limitations:** Dataproc clusters allow dynamic adjustment of resources. With higher RAM and CPU configurations per node, Cluster 1 significantly reduced processing times compared to the local environment while both Cluster and Cluster 3 were able to run the task for 2 and 12 files that was not possible on the local environment. However, Cluster 2 has some slight performance improvement,
2) **Data Size and Storage Capacity Limitations:** This bottleneck is addressed as the clusters can access large datasets stored in GCS without local storage limitations. The ability to handle large datasets, as demonstrated by Cluster 3 processing the twelve-file dataset, indicates that GCS can support significant data sizes without local storage constraints.
3) **Single-node Execution Limitations:** Dataproc allows the creation of scalable clusters with multiple nodes (both master and worker nodes). This distributes the workload across multiple machines, enabling parallel execution of tasks. The in the multi-node setup in Cluster 2 demonstrates that even with limited individual node resources, clusters can provide better performance through parallel processing and scalability. Moreover, the significantly improved performance in Cluster 1 and Cluster 3 as well may illustrate the advantage of parallel processing and resource distribution.

C. Reflections

The performance of Cluster 2, despite being a cloud cluster, did not show a significant improvement over the local environment. This indirectly suggests that even cloud services may not offer substantial performance gains if resource limitations exist. From this observation, we can conclude and learn as follows:

1) **Resource Constraints in Cloud Environments**: Cluster 2, with only 8GB RAM and 1 vCPU per node, shows that limited node resources can significantly restrict performance improvements. Moreover, if individual node resources are too low, the overall performance gain from scaling across multiple nodes may still be minimal. The benefits of cloud scaling are best realized when each node has adequate resources to handle its portion of the workload efficiently.
2) **Cost-efficiency**: While cloud services offer flexibility and scalability, there is a need to balance cost and performance. Allocating more powerful nodes or increasing the number of nodes can lead to better performance but at a higher cost. Optimal performance often requires fine-tuning resource allocation to find a cost-effective solution.

D. Summarize the budget and actual money spent

The total actual cost of our project came out to be $5.34. This is higher then our estimated due to multiple reasons.

1) The project is billed while the servers are active, even if they're not running a job.
2) Multiples jobs were run when testing the clusters. We analyzed more data then what was strictly required to check if our project was running properly.

Broken down the cost of our project can be split into 3 services; Compute Engine, Networking and Storage as illustrated in Table IV.

TABLE IV
COST BREAKDOWN

Service	Compute Engine	Networking	Storage
Cost	5.20	0.11	0.03

Overall the largest cost was the compute time on the servers. This accounts for over 90% of the cost of the project.

V. CONCLUSIONS

In conclusion, our project aims to enhance the Spark ML workflow for malicious network traffic identification using Google Cloud Platform's Dataproc service. We have identified three major bottlenecks including hardware runtime resources limitations, data size storage capacity limitations and single-node execution limitations. The project has successfully addressed the bottlenecks as follows. Dataproc's dynamic cluster management and elastic scaling allowed us to efficiently distribute workloads and allocate sufficient resources, optimizing Hadoop and Spark configurations for better resource utilization. Google Cloud Storage provided unlimited, high-throughput storage, enabling quick access to large datasets. Enhanced parallelism and scalability through multi-node clusters ensured efficient processing of large ML workloads. Experiment results of the local machine and clusters demonstrated significant improvements in processing time and scalability, validating the effectiveness of Dataproc in mitigating performance bottlenecks and enhancing the Spark ML workflow. We have also learned from the limitation of cloud services under resource constraint condition that simply moving to the cloud does not guarantee substantial performance enhancements. The adequate resource allocation and optimization in cloud environments is also of great importance.