Real Time Call Monitoring System Using Spark Streaming and Network Intrusion Detection Using Distributed WekaSpark

Darshan V S1,*, Ria Raphael2
1 Department of Computer Science and Engineering
Amrita School of Engineering, Bangalore-Amrita Vishwa Vidyapeetham University, India
2 Department of Computer Science and Engineering
Amrita School of Engineering, Coimbatore-Amrita Vishwa Vidyapeetham University, India

Abstract—With the increase of calls in industries it is very difficult to identify the calls made in a huge organization. The study and developing analytics out of the call history generated in terms of real time or the information stored helps in the improvement of the quality of calls in terms of network failure analysis, analysing call usage pattern from minimal to maximum to increase server efficiency, analyse user level pattern. The capability to process, analyse and evaluate real time data in a system is a challenging task, the test of building up an adaptable, shortcoming tolerant and flexible observing framework that Can deal with information continuously and at a huge scale is nontrivial. We exhibit a novel framework for real time processing and batch processing by using spark streaming and spark, also an ensemble model is used with distributed weka-spark for intrusion detection.

Keywords—Big Data; Analytics; Machine Learning; Weka.

I. INTRODUCTION

Most of the industries will be having some call making app, ip phones, etc to make official calls. Some companies even may allow their employees to make personal calls with their app or company phones. And so, there are chances that the company might face many issues on the free calls that they provide as a benefit for their employees. Hence the company must monitor and analyze each call. To analyze each of the calls manually is not feasible. In large organization, where in there are substantially substantial number of employees and substantial projects, the number of calls made will be enormous. Each second there can be n number of calls made. Hence this becomes Big Data as the velocity is high. Therefore, in this paper we propose a method to handle call records of industries which is a big data and perform analytics on it and to integrate different big data tools which are needed for the processing. We also integrate with distributedWekaSpark for intrusion detection by using majority voting with ensemble mechanism.

II. RELATED WORKS

This paper deals with handling Big Data, also perform analytics on it. Usually big data is handled by using tools, we use 3 variants of big data tools to handle large volumes of data. First for streaming we use Apache Kafka [3] and Apache Spark Streaming. Second for analytics we use hdfs to store historic data and spark sql to perform analytics. Third for visualisation we use elastic search, kibana and d3.js. Now let's go in detail of these tools. First let's start with the streaming tools. We deal with real time data and the data is increasing every second, we are not dealing with a bulk data which is constant. Hence to process this real-time data we must stream the data first. For this purpose, we integrate Kafka with spark streaming. Kafka is a distributed message transferring tool. It acts as producer consumer. In kafka the data is stored and queued using topics, we create a topic with some

* Corresponding author can be contacted via the journal website.
random name and the data which is produced by any producer is stored in the kafka topic. Later on, the consumers can consume it from the topic. There can be n number of producers and n number of consumers for a particular topic.

Spark [5] is a big data processing tool. Spark is 100 times better compared to Hadoop in terms of efficiency and speed. Hence, we go for spark here. Spark streaming is a part of spark which does data streaming. In spark streaming, the streaming data is taken as input and it will also do some processing with the data if required and then given to output as batches of data.

HDFS [9] is Hadoop’s file storing system. It stores file in distributed manner across the cluster for fault tolerance. Any file with any size and any extension can be stores in HDFS.

SparkSQL [5] is another feature of spark where we can perform analytics on the data by simple querying method. The queries are similar to basic SQL, but only difference is that in SQL the query is performed on a limited dataset and its sequential. In spark, query is performed on large data, the data is split into chunks of data and the query is applied to it. The result of each chunk is considered to decide upon the final result. Each chunk is processed in parallel. Hence the results obtained are efficient in terms of time and its quick.

Elasticsearch [1] is a search based server. It is a tool where large chunks of data are stored. This data can be used later when required for different purposes. The data can be retrieved by some search filters or the whole data can be used. Even query operations can also be performed in elastic search with the data stored in it.

D3.js [2] and Kibana [4] is used for visualisation. Kibana has built in integration for Elasticsearch, it is user friendly. In kibana even if tile maps are there to visualise world maps, but there is no method to perform animations on the map. Hence for visualizing maps and to perform animations on it d3.js is the efficient.

J48 and Random Forest, decision trees work good with large datasets and gives high accuracy [6,7,20]. Distributed Weka-Spark [8], weka integration with spark which leverages the power of spark for distributed execution.

III. PROPOSED ARCHITECTURE

The proposed system consists 3 phases data acquisition, data analysis (batch processing) and visualisation. The architecture diagram(figure 1) has the following phases

1. Data Acquisition The first phase of the system is data collection. The data is stored in the cloud. Each time when a user makes a call it gets recorded in the cloud. Therefore, each second there can be a number of records recorded in the database. The connectMe+ app is used worldwide in the industry. There is frequent calls across different countries. Also the frequency of the data is high. To monitor the calls, streaming the data on real-time basis is done. Therefore, we use kafka integrated with spark streaming to read the data from database. The incoming data is processed in spark and is stored to both HDFS (for batch processing) and Elasticsearch (for visualization).

2. Batch Processing There are many analytics the industry would look upon its meta data of call records. These analytics are processed in SparkSQL and written back to HDFS for also displayed in a dashboard.

3. Visualization Both real time streaming and batch processed data has to be visualized in a dashboard. Real time monitoring is displayed using D3.js and analytics is displayed using Kibana.

![Architecture Diagram for Analysis and Intrusion Detection](image)

**A. Intrusion Detection Design**

There are several occurrences of intrusion in the real world, to detect an intrusion we have collected data, with this data we apply classification algorithms then based on the best selected model, we reapply it to the real data and classify it as intrusion or anomaly. The proposed intrusion detection model is given in figure 2. In this model, we have applied J48, decision tree and Naive Bayes algorithms [21] to create an ensemble model [17,18] of the same to detection an intrusion using majority voting.
IV. IMPLEMENTATION AND RESULTS

A. Cluster Setup

This is a pre-requisite for the following system that a cluster has to be setup to work in a distributed environment. We created a 4-node cluster. The cluster setup has the following process, for each node we install community enterprise operating system and java. Then we install ambari server in one of the machine and install ambari agents in each of the machine. The installation process consists of selection of the nodes, assigning master nodes (figure 3), assigning slave nodes (figure 4) and to deploy and start the services (figure 5).

B. Implementation

The different phases of development of this work are presented in the following sections. This is a KDD process and hence it starts with data collection.

- Data Collection: We receive the call information from a MySQL database which is hosted in the Amazon cloud, as soon as we receive a call we have a record created in MySQL database, when the call ends we receive the session information for the call duration. The data is received from the App ConnectMe +

- Data Preprocessing: The initial step in every KDD process is data pre-processing. Preprocessing steps depends on the data. In our data, the only pre-processing that we have to do is data cleaning. The data is cleaned by handling the missing values using binning.
The whole system is designed for two phases:

a) Real time monitoring
b) Batching processing

The aim is to monitor the calls in real time and visualize the same. Whenever a call is made, the ongoing calls (source country to destination country) are displayed in a dashboard on which a 2D world map. In order to perform real time streaming first process the raw data. The processing is done by spark streaming. We integrate kafka and spark streaming[12] to pull the data from Mysql and push it for processing. Using an adapter named maxwell[11]( connector for Mysql to kafka), the raw data is pulled from the database and stored in a topic in kafka. The data in the particular topic is pushed to spark streaming for processing.

Data pre-processing and spark streaming is done in parallel because each time a record is inserted into the database, it has to be cleaned. Hence pre-processing cannot be done only at the start, instead it is done in parallel with streaming data. Maxwell adapter pulls the data from database and stores each record into the specified kafka topic in json format, extract the required fields and perform the processing. First identify the calling phone number and called phone number from the json and extract the country code to identify the country. Now convert it to corresponding latitude and longitude in the world map. Now this processed data fields(latitude and longitude of calling and called country) along with start time, end time and session time is being saved to elasticsearch. elasticsearch.js is used to load the data from elasticsearch and visualize it in a dashboard using d3.js. The monitored calls are shown in figure 6,figure 7 represents how the data has been stored in indexes. Next we perform analytics on the data stored. We go for batch processing All the inferences which are made purely depend on the data which is stored. The analytics performed are number of calls made by each user, call failure analysis, maximum number of calls received by a source or destination country is receiving displayed in Kibana (figure 8). Next we find out the Calls that have exceeded a time limit, we find out the difference in start time and end time of each call. Whenever a record is inserted, we make a check statement check whether the difference exceeds a specified time, the same is shown in figure 8 in the second top tab from the left. Next we try to find calls with small duration but made frequently. To find frequent calls, query the count of calls made by different users in a specified time interval but querying is not feasible in real time. It can be done only in batch processing. If the requirement was to find the frequent calls need not be in real time. Then we can use batch processing. But here we are doing it in real time, so we use a dynamic data structure ie is a hash map, where we store phone number, time and count. When a person makes a call for the first time, his phone number, the time when he made the call and count value = 1 will be saved to hash map. Now each time the same user makes a call, the count value will be incremented by 1. Also a condition check is made on time. That is if the new time when the recent call is made and the time when the first call is made is within a specified time interval and also if the count has exceeded a certain specified threshold (say, for example 30 calls in 5 minutes, etc) then that user is said to have made frequent calls. After finding the frequent calls and calls that have exceeded time limit, it has to be informed to concerned authority. One of the methods we used is to alert the concerned authority using a mail. Another method is to display it in a dashboard. Here we use kibana to display it in the dashboard (Figure8), right most tab. For batch processing we use historical data and make some predictions. In our system we are store historical data in HDFS. Our system handles real time data from real time data, there are 3 methods to store, First method is using spark streaming. While streaming real data, an in built function saveAsHadoop() can be used. Now the data will be stored in HDFS in table format. If we want to save in some specified format we can use an in built function saveAsTextFile(). Specify the format as the argument of the function. Second method is to use scoop[14]. It is a tool which directly writes the data from MySQL to HDFS in table format. Third method is to use an existing project called camus[13]. This writes the data from kafka to HDFS. This also stores the data in a table format. There is a disadvantage in using camus. Camus reads data from all the topics which is created in kafka and writes it to HDFS. This may be an advantage in some other application. But for our system, this is not preferred to use. Since we use a single topic for queuing the data. And there may or may not be some other applications which use another topic for queuing or transferring its data. Hence the first two methods are preferred for our system. In this system we go for the first method. Since we perform some preprocessing tasks and process the data according to our requirement. We use saveAsTextFile() and store the data in a specified format. We use sparkSQL for batch processing. SparkSQL is simple to use as it is just similar to querying in SQL. Only difference is that spark handles big data. Most of the existing call analytics performed are the calls made for longest duration, shortest duration, number of calls made by each person per day.
C. Intrusion Detection

Detecting intrusion[15,19] is to identify the hackers using the app. This can be a intruder inside the company or outside the company. The task is to predict whether it is intrusion or not. With the static data we build a model. The evaluation criteria for different models were on accuracy, time taken to build the model and memory. Evaluation matrix for different models are shown in table 1. putting all these in mind the model finalized is an ensemble model of j48 naive bayes Random Forest. It showed that Random Forest and J48 is giving the highest accuracy. But it is resulting in over fitting[16]. Hence to reduce over fitting naive bayes is also included. Random Forest takes a time in building the model and also takes memory. But can be compromised as even if in building the model it takes time, what we care more is the time for predicting. Prediction time is also high when compared with other models but it is not very high hence can be compromised. Random Forest is best to handle big data and hence cannot be avoided. Intrusion Detection was done on Distributed Weka-Spark[8] which is the same weka tool build on spark for advanced analytics but we used this jar and created a model programmaticaly using majority voting mechanism and classified into intrusion or anomaly. Right now the distributedwekaspark jar supports only with .csv files. Hence we have the data in HDFS in csv format. We Applied Various Algorithms to our training data set by using DistributedWekaSpark for building the model on the dataset and we have achieved the following results show in table 1 and table 2 respectively. Table 1 shows the correctly classified instances(accuracy) based on the training set, Table 2 shows the correctly classified instances based on the Test Set.

<table>
<thead>
<tr>
<th>Table 1. Comparison of Models with Training Set</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sl.no</strong></td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

After Applying it to the testing set, ie when the model was built and the classes were generated from
the distributedwekaspark and then we got the following results shown in Table 2

<table>
<thead>
<tr>
<th>Sl.no</th>
<th>Algorithm</th>
<th>Correctly Classified Instances (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J48</td>
<td>81.5339</td>
</tr>
<tr>
<td>2</td>
<td>Random Forest</td>
<td>80.4516</td>
</tr>
<tr>
<td>3</td>
<td>Naive Bayes</td>
<td>78.1178</td>
</tr>
<tr>
<td>4</td>
<td>J48+Naive Bayes(Vote)</td>
<td>81.5605</td>
</tr>
<tr>
<td>5</td>
<td>J48+Random Forest(Vote)</td>
<td>81.5605</td>
</tr>
<tr>
<td>6</td>
<td>J48 +Random Forest + Naive Bayes(Vote)</td>
<td>81.3298</td>
</tr>
</tbody>
</table>

Time taken build the model and memory taken to save the model is shown in table 3.

<table>
<thead>
<tr>
<th>Sl.no</th>
<th>Algorithm</th>
<th>Time taken (ms)</th>
<th>Memory (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J48</td>
<td>79705</td>
<td>130</td>
</tr>
<tr>
<td>2</td>
<td>Random Forest</td>
<td>237950</td>
<td>8706</td>
</tr>
<tr>
<td>3</td>
<td>Naive Bayes</td>
<td>4803</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>J48+Naive Bayes(Vote)</td>
<td>77567</td>
<td>138</td>
</tr>
<tr>
<td>5</td>
<td>J48+Random Forest(Vote)</td>
<td>324285</td>
<td>8829</td>
</tr>
<tr>
<td>6</td>
<td>J48 +Random Forest + Naive Bayes(Vote)</td>
<td>171378</td>
<td>8837</td>
</tr>
</tbody>
</table>

V. CONCLUSIONS

A Real Time Call Monitoring System was developed. The analytics on the call meta data was done both in real time and also in batch processing. The performed analytics are:

- Number of calls made by each user
- Count on termination cause
- Identify the source/destination from/to maximum calls has been made
- Identify frequent calls and calls that has exceeded a time limit
- Intrasion Detection using ensemble modeling (majority Voting) by using distributedWekaSpark

REFERENCES

[16] Definition of overfitting www.investopedia.com/terms/o/overfitting.asp


