BIA 678-A Term Paper

APACHE FLINK

Archana Kalburgi

CWID : 10469491

Table of Contents

Apache Flink

Abstract

Flink, part of the Apache Flink project, is an open source platform for stateful calculations using massive, **unbounded and bounded streams** of data. Flink's main advantages are that it can run on a wide range of cluster environments, perform computations in memory, and run on a variety of platforms. Data can be processed as both bounded and unbounded streams. Unbounded data streams are the data streams that do not have a defined end but have a definite start. After termination the data obtained is not the same as when it was generated. Continuous processing must be done on unbounded streams, i.e., once they are ingested, they should be handled. Because all input data is unbounded, it is not a good idea to wait for the entire data to arrive. Unbounded data processing often requires that events are ingested in a specific order, probably in the order in which they occurred, to be able to determine the completeness of the result. On the contrary Bounded streams have a defined start and end. Data from bounded streams can be processed by ingesting all of it before any computations. For bounded streams, ordered ingestion is not necessary because they can be sorted.

In this paper I will try to explore how Flink can be incorporated by financial institutions in their dynamic fraud detection mechanisms.

Introduction

The number of data collected by companies has been increasing over the past few years. Taking advantage of this huge amount of data that is being collected, data scientists can generate business value. Models can be trained based on the historical data that is stored on HDFS and later these models can be scored based on the accuracy.

In case of financial institutions, the fraudulent transactions have to be blocked immediately to prevent damages to the institutions or customers for which the models responsible for detecting frauds are required to be scored in real time.

The real challenge is to add efficient models on a running system without downtime. In this paper I am going to discuss the implementation of a real time streaming analytics platform that also allows to dynamically change the behavior of the stateful Flink application. This will result will an environment that provides the users a DSL which can be used to dynamically stream the new models into the Flink

Building blocks of a streaming application

The efficiency of any streaming application is determined by how well the application framework can control streams, state and time.

Streams

Stream data is the data that consists of a stream of logs, events or any record along with the time as they happen. Stream data can be categorized based on its properties. Major categories include bounded and unbounded streams, real-time and recorded. Flink framework is designed to handle any kind of stream data, it has advanced features to process unbounded data and operators to process bounded streams. Flink applications are also efficient in processing file systems both in real time or for later processing.

Some of the examples of such streams are:

- Credit card transactions
- Sensor measurements
- Machine logs
- User interactions on website
- Mobile applications

State

Most of the streaming applications except for the ones that require the application of the transformations on an event are stateful. For applications that need to remember the events for running a logic needs to remember the state to access them in the future.

Flink is designed to cater to the state handling mechanisms. The flink framework provides multiple state primitives, pluggable state backends, exactly-once state consistency. Checkpoint algorithms enable flink to handle a very large state. It is also efficient in scaling stateful applications.

Time

Events are produced at a specific point and are not regular which makes tream applications dependent on the time semantics. Measuring time becomes an important feature for applications that perform operations like windows operation including all or any aggregation functions, pattern detection or time bases join operations. In such functions keeping track of the event-time and processing time is crucial for reliable results.

Flink makes it possible to keep such track relates to time because of its powerful time related features, which are event-time mode, watermark support, late data handling and processing time mode.

Advantages of Flink

Flink can be used to develop a wide range of applications since integrating flink with any of the cluster resource managers such as Yarn, Apache, Hadoop, Kubernetes or Mesos requires minimal effort. Flink can also be set up to operate as a standalone cluster manager. The resources specific mode in the flink allows it to interact with any of the resource managers in an idiomatic way removing the barrier of the varied programming language. The application resources required are identified by the flink automatically by analyzing the configuration of the application.

Most important feature of flink is it's failure mechanisms, flink handles failure by replacing the failed container with a new container up the.

Scalable stateful applications can be developed using flink, the status of the state is updated and controlled using REST api. Flink also has an inbuilt checkpoint system.

These features make flink a better choice for developing fraud detection applications, anomaly detection particularly in video streaming applications. Broader use cases include developing event-driven applications, data analytics applications and data pipeline applications.

Fraud Detection System Using Flink

A fraud detection system must be able to support various types of machine learning models and be able to score these models in real time. A successful fraud detection system must have one single codebase so that the solution can be deployed in different countries and have flexible decoupled architectures. The key part is being able to dynamically change the models on a running system i.e in any new fraud scenarios and a loophole in the models, new models must be updated and streamed into the flink without downtime or redeployments and must have ready to use system. Here is a brief discussion of how we can design such a system.

Support various types of models

To build the supporting models and to score these models an offline and an online environment are created. In an offline environment the domain experts are responsible for creating models using tools like Knime and spark. Historic data from the hadoop system can be utilized to update the models. Once the models are trained it can be streamed into Flink. Flink can later score these models in real time against the incoming events. To deploy the models from an offline environment to an online environment, the models have to be persisted for which is achieved by a sustainable environment called Predictive Model Markup Language, often abbreviated as PMML. PMML is an XML based format which enables the models to be saved. PMF files are lightweight and easy to interpret. For example if we were to save a neural network model in a PMF life, all the model parameters would be saved in the file as can be seen in figure1.

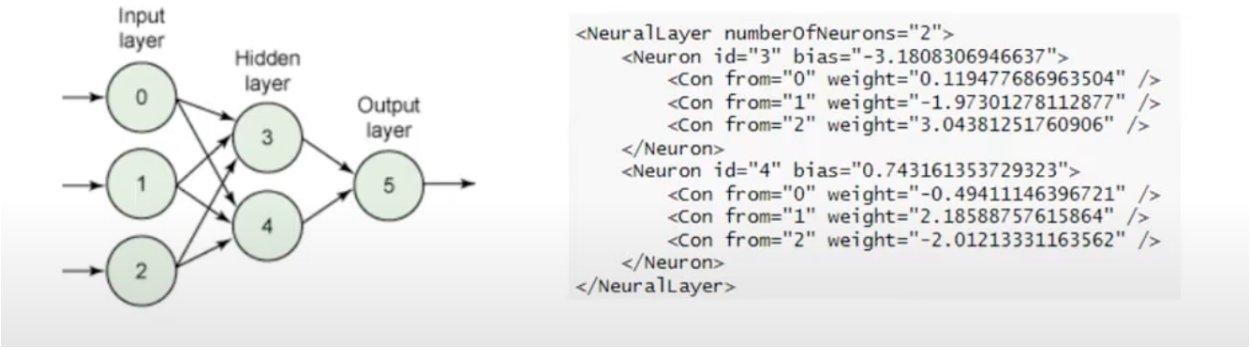

figure1

Most of the tools such as Spark, KNIME, R, python support PMML export mechanisms which open a plethora of options to be implemented.

In the online environment the PMML files are streamed in by KAFKA and they find their way into the control stream's mobile scoring operator. The OpenScoring.io library in the model scoring operator then converts the PMML file into a PMML object which exposes it into two functions. The first function passes the input to the model. Typically the inputs are the feature or the feature sets to the model. Once the model is fed the inputs the model is scored which can then be displayed on to the stream.

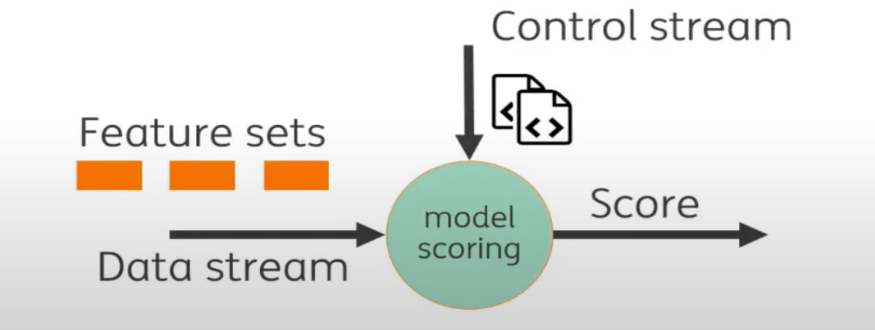

figure2

OpenScore.io supports association rules, cluster model, general regression, naive bayes, k nearest neighbor and ensemble models

Architecture

This flink system is flexible enough to be deployed in different environments and different geographical locations.

In most of the flink applications the input data is read from multiple sources into KAFKA, which is later processed and used depending on the use cases and then returned to KAFKA. In fraud detection the input is read from multiple sources such as the online banking application, mobile app, ATMs etc. The input from these sources come in a different datatype taking away the flexibility of processing it. This is overcome by introducing a second flink application known as preprocessor which is basically a technical filter seen in figure3. This preprocessor is majorly concerned with filtering out the events that are not needed by the feature selections and model scoring applications, can classify the raw events and business events and it also consolidates all the events.

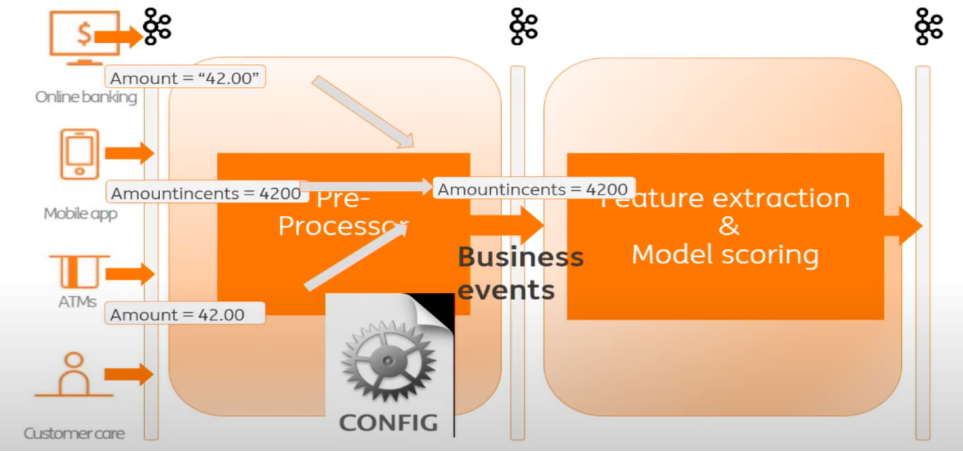

figure3

Stateless to stateful

When a model is updated the pattern must be stored. The model scoring prepended with the feature extraction is not always guaranteed to have all the required features or information about a transaction. In this case we must take the previous transactions into consideration. In flick it's achieved by introducing a state into the application to give the feature extractor a memory which will enable a better feature calculation for the future transactions. Memory overloading is avoided by storing just enough information required to calculate the feature. Here the model scoring is made efficient by creating a time series of the events which is achieved through updating the state. The model will now recognize the potential fraudulent transactions and create an alert.

Stream

Using stateful operators we can enrich a stream. Decisions can be made even if the input is insufficient. The keys in the transaction for examples are enriched by different keys. The input is split and assigned to the task managers. Each key has information related to it stored in different nodes, which is how the keys get enriched. The enriched keys in the cluster are then aggregated on one operator within the cluster. This is done by generating a new key, a random number for each new business event. And the key stream done on those same numbers will end up on the same operator. All the information will come together in the aggregation step, this information can be used to calculate the feature set for all the models and emit them on a stream. Each model looks at the different feature set and makes the decision of alert or ok status.

Conclusions

An end user might not be interested in the internal working of the flink. The user must be able to create a model and save it to a PMS file. But in addition to it the user must define how the feature set must be calculated and must define what data must be persisted since feature calculation is done based on the historic data. In order to enable this a Domain Specific Language is created for the end users for the customers to specify what is required out of flink.

The system can be extended to send customer notifications in case of fraud transactions or if the balance goes below a certain threshold.

References

1. https://flink.apache.org/flink-architecture.html
2. https://sf-2017.flink-forward.org/index.html%3Fp=1914.html
3. https://ieeexplore.ieee.org/document/9004318
4. https://flink.apache.org/news/2020/01/15/demo-fraud-detection.html