CS 585 Big Data Management
Project Report

Deep Learning on SUSY dataset using Kafka and TensorFlow

Name: Muralidharan Kumaravel, Amey Dilipkumar More

Abstract:

The main aim of this project is to create a containerized data pipeline to stream the data from the SUSY dataset, train a deep learning model and testing the performance of the deep learning model. The SUSY dataset is simulated particle accelerator data, thus we used Kafka to stream the data and consume the data using TensorFlow. Further, build a deep learning model and train it using the consumed data. The trained model is saved to MongoDB and used Pyspark and Pandas to test the saved model on the test data. The entire data pipeline is containerized using Docker.

Problem Statement:

The SUSY dataset is a particle accelerator data made using Monte Carlo Simulation. The key aim is to identify whether the incoming signal can produce hyper symmetric particles or a just a background process. Generally, in real life particle accelerators are real time streaming data. Therefore, the objective is to stream the data from the dataset to Kafka and further consume the data from Kafka and create deep learning model for the classification objective. The models needed to be saved to MongoDB such that it can be accessed anytime for testing with the test data and the trained model won't get lost once the containers are down. And the structure of data streamed and consumed should be compatible with TensorFlow and all applications should be independent of one another.

Technologies Used:

Kafka – The data from the local machine is streamed into a Kafka topic and the data from the topic is consumed and used for training the model.

TensorFlow – It is used to consume the data from the Kafka topic using TensorFlow IO. Further, it is used to train and test the model.

Spark, Pandas – It is used to load the testing data and process it for testing.

MongoDB – It is used to save the trained models JSON with IDs and timestamps.

Docker – All the process are containerized using docker individually as a microservices. Each operation can individual operate in the data pipeline.

Dataset Description:

This dataset is generated using Monte Carlo simulations and contains **2.22 GB** of particle accelerator data. The dataset contains 5 million records of a simulated particle accelerator. First 8 features are kinematic properties. The next 10 features are functions of the first 8 features. Labels are 0 for Background signal and 1 for Signal that produce hyper symmetric particles.

Further the data is scaled to **23.2 GB.** And the scaled data is processed using Pyspark, Pandas for testing the model performance.

Challenges:

Finding an effective way to stream the data and the data transforms between Kafka and TensorFlow. Used Key for labels and message for the features and used TensorFlow IO operations for data transform from Kafka to TensorFlow.

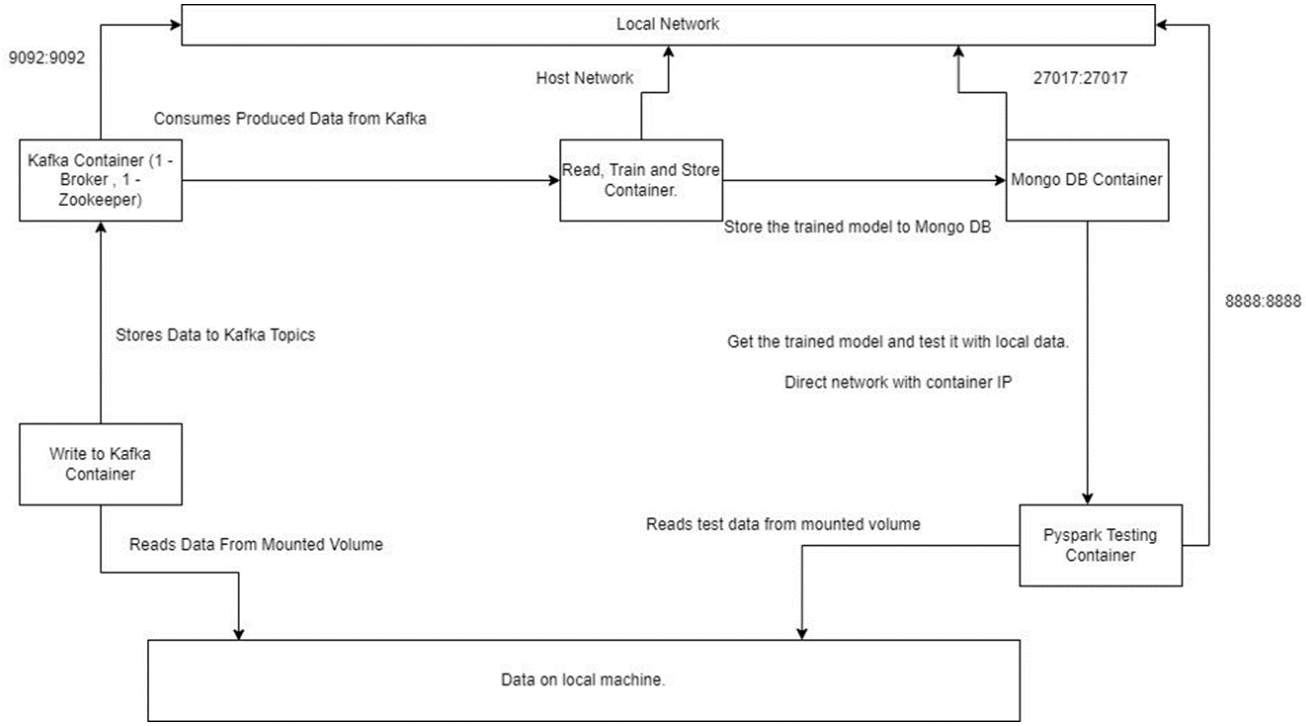
Migrating the applications from local machine to the docker. When migrating it creates an independent network for the container and the ports and enabling the container to connect to host network was taken care.

Inter container communications. Containers have dynamic IP when building from a image. Even though some ports are exposed to localhost, it still created problem to connection between containers. Dynamic IP of container is used for connection when testing the model using scaled data.

Process large data for testing. Custom methods and functions where put in place to avoid memory overflow while testing the data.

Saving the model and accessing it. Model is converted to JSON object and stored in MongoDB for testing its performance.

Methodology and Architecture Diagram:



Write to Kafka – This is containerized python application, where it reads the data from the local machine and processes it and the streams the data to a Kafka topic "trainData". The trainData consists of 90000 records that are streamed into Kafka with comma separated for the features and the label is encoded into the key. The data is processed using Pandas and Scikit Learn and written into the Kafka topic. The directory to the local data is mounted as a volume to this container. And this container is connected to the localhost network.

Kafka (Broker, Zookeeper Containers) – Two containers one for the Kafka Broker and one for Zookeeper are running independently and they ports are exposed to localhost ports at 9092. The topic "trainData" is created using docker execute on to the container.

MongoDB – This is a MongoDB containerized server where this port is connected to localhost port at 27017. A database called "bigdata" is created with a collection named "MLModels" to store the generated models.

Read, Train and Store – This also a containerized python application, where it consumes the data from the Kafka topic "trainData" and process the data into

TensorFlow tensors for training. The model is defined in this container itself. This application container is also connected to the localhost network.

Model Architecture:

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_36 (Dense)            (None, 128)               2432

 dropout_27 (Dropout)        (None, 128)               0

 dense_37 (Dense)            (None, 256)               33024

 dropout_28 (Dropout)        (None, 256)               0

 dense_38 (Dense)            (None, 128)               32896

 dropout_29 (Dropout)        (None, 128)               0

 dense_39 (Dense)            (None, 1)                 129

=================================================================
Total params: 68,481
Trainable params: 68,481
Non-trainable params: 0
_________________________________________________________________
None
```

The model is trained using the consume data for one epoch. As it is a binary classification, binary cross entropy loss is used and this loss is optimized using Adams optimizer. The batch size for training the model is also defined within in the container. The trained model is converted to JSON object. A client is created and connected to the MongoDB database which is also containerized. The JSON object of the model is saved with a unique ID and timestamp of the model generated to the MongoDB database.

Pyspark Testing Container – This is dependencies preinstalled jupyter notebook container exposed to localhost network at 8888. This is container create a client to connect to MongoDB using the container IP address. Load the JSON model and complies the model for testing. The model is loaded from MongoDB either by the timestamp created or the unique ID it is stored. The testing data is scaled to 23.2 GB. The entire data cannot be loaded into memory for testing. The data is scaled 15 times the original data. The per iteration of testing, 1 millions records are loaded and tested on the model. That is 1 million records are loaded into memory. The accuracies were stored in list and once all the records have been tested the entire model accuracy is predicted. The testing data from local machine is mounted as a volume to the container.

Execution Guidelines:

- Download the SUSY dataset from UCI machine learning repository.
- Build the image for writekafka using the docker file in the file named “writekafka.
- Build the docker image for readtrainkafka using the docker file in the file named “readtrainkafka”.
- Use the docker compose file docker_compose_kafka.yml to create and launch the containers for kafka and zookeeper. Use the command “docker exec broker kafka-topics --bootstrap-server broker:9092 --create --topic trainData” to create the topic trainData.
- Pull the docker image for MongoDB 6.0 and build the image.
- Use “docker exec -it MongoDB_Container_Name” bash” command to get to the container shell. Run “mongosh” inside the container to get into MongoDB. Create a database called “bigdata” and a collection called "MLModels”. And exit the container.
- Use the command “docker run -it --volume "Dataset Location in Local Machine":/writekafka --network="host" writekafka” to launch the writekafka container.
- Use the command “docker run -it --network="host" readtrainkafka” to run the readtrainkafka container.
- Use the commands in MongoDB step to view the saved model.
- Pull the image for pyspark jupyter notebook from docker hub.
- Use the command “docker run -it -p 8888:8888 --volume "Test Data Location in Local Machine":/readTest --name pyspark jupyter/pyspark-notebook” to launch the pyspark notebook and access the jupyter notebook in the local machine browser.
- Use the command “docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' MongoDB container name” to get the IP address of the MongoDB container.
- Run the notebook, but change the timestamp or model ID that needed to be chosen and view the results. And use the IP address acquired in the previous step to connect to MongoDB client.

Results:

Operation	Time in Seconds
Loading a batch of 90000 data and processing it for sending to Kafka.	1.89
Reading the data from data frame and producing it to a Kafka Topic as a key and message. (Including connecting to a Kafka broker)	3.56
Consume the data from Kafka and load it into batch for training	0.145
Training of the model for 1 epoch	9.98
Storing the model to MongoDB.	Almost instantly (in MS)
Testing the model for 23.2 GB data	7772.76

As we can see loading a batch of it not that time consuming as when loading a single batch there is no memory constraints. But if there is not enough memory to load the entire data (RAM: 16GB, Data: 23.2GB) processing data in a single machine batch wise takes a long time. This can be optimized using thread-based operations such that each thread tests the model for some batch of data. Finally, all the individual accuracies can be merged from each thread. So that the testing the model happens distributedly instead of batch wise testing.

Conclusion:

This project gave exposure to creating data pipeline using docker, Kafka, MongoDB and TensorFlow. We learned about Kafka offsets and implementation of Kafka topics to store data and implementation of TensorFlow for creating, training and testing models. We gained insight on docker, it's working mainly networking in docker. The architecture works well when the data is processed in batch wise. As the models are stored in the MongoDB, lots of models and be made, even ensemble learning can be performed. As new data comes in new model for the data can be created, which acts like a version of continual learning. MPI or Open MP can be used to process the batch of data for testing parallel with the trained model. Because each batch of data testing is completely independent. As each application is independently containerized, the data pipeline can be managed effectively.

Github Repository Link: **https://github.com/murali22chan/CS-585-Big-Data-Management-Final-Project**