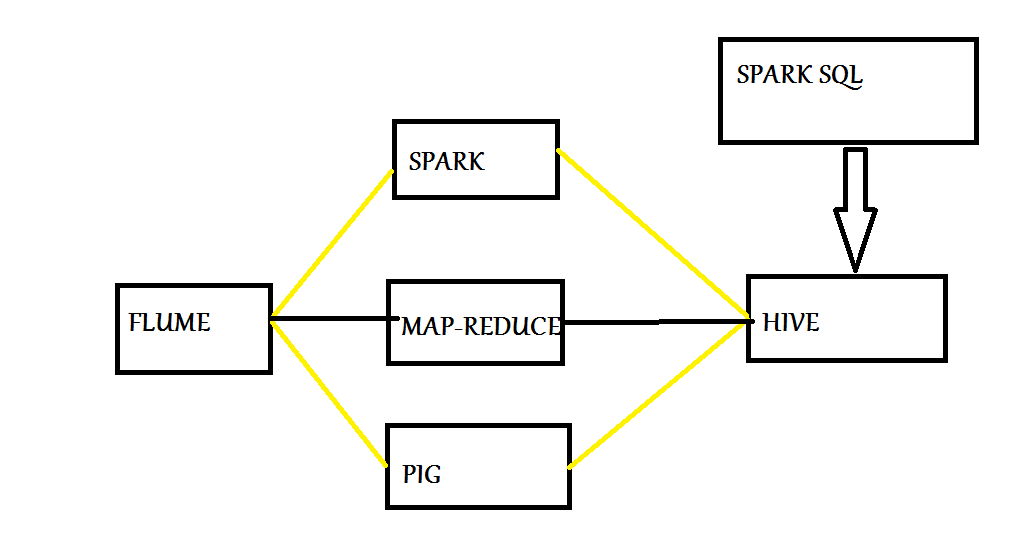
BACK-END BIG DATA PIPELINE(by Aayush Grover):

The back end pipeline consists of following components:



FLUME : I’ve implemented a custom Flume Source in Java for which Twitter streaming API is acting as the Source, Flume is acting as the channel, and the sink is HDFS. Thus, I am fetching tweets from Twitter via a custom flume source and storing these raw tweets in HDFS. Now, it is very important to write a Custom Flume Source over here, instead of using the default Flume, because by using default Flume we would end up fetching the same tweets over and over again. Thus, I created an application on http:// dev.twitter.com and I used the tokens of this app in my Custom Flume source to fetch the tweets and also provided a timestamp interceptor in the source code so that Flume didn’t fetch the same tweets over and over again. Please find the Flume Java Source code in CODE folder.

Once the tweets have been fetched, now in order to convert the raw unstructured tweets into structured format, I have used 3 different components:

SPARK : I’ve written a Spark Core program in Scala that performs batch processing to convert the tweets to structured format. Here the input path is the path of the tweets that were stored in HDFS by FLUME. While implementing this pipeline in real world, Spark should be chosen when data is around 200-220GB for a production cluster of around a 100 nodes. Beyond this limit of data using Spark wouldn’t be much efficient as we would not be able to perform efficient caching operations, and thus the performance of Spark would be just a bit better than that of Map Reduce, but that too after spending a lot of cost. For our 3 node cluster, this limit is of 12 GB. Please find the Spark Scala Source code in CODE folder.

PIG : Secondly, I’ve written a PIG script in pig latin language to convert the tweets into structured format. Now, in real world scenario PIG would be chosen over Spark for Batch processing for large amount of data of the order of around 200TB. Beyond this limit of data, the performance of the Pig Script would deteriorate, as when a PIG job runs it gets converted into a logical job, then a physical job, and finally a M-R job and then this M-R job is executed by YARN. Now this Map-Reduce job is the default Map-Reduce job and we can’t optimize it further to improve the performance. Thus, for data of less than 200TB, in order to skip the large code development time for Map-Reduce, Pig should be used. In our 3 node cluster this limit for PIG was around 28GB. Please find the Pig Latin Script in CODE folder.

MAP-REDUCE : Thirdly, I’ve written a Map-Reduce program in Java to convert the tweets into structured format. In real world scenario, this Map-Reduce program can even be used for Batch processing of Petabytes or Exabytes of data. In this Map-Reduce program, in order to improve the performance, I’ve used a combiner per mapper, I’ve used 11 reducers where each reducer’s output contains the structured output for each category of products, and in order to send a particular category output to a particular reducer, I’ve implemented a Custom Partitioner. Please find the Map-Reduce JAVA source code in source folder.

HIVE: Once I got the data in structured format, I’ve stored this data in a Hive table, i.e. for 70 days I kept on storing the structured data in a Hive table so that in the end I can query this historical data to gain fruitful results. Now while storing data in Hive, I’ve performed dynamic partitioning on the data. Say, if I wouldn’t have performed Dynamic partitioning per category and then queried the data and say, my query is “Select \* from table where category = ‘female bags’”, then to perform this query all the records will be parsed and then the ones with category as female bags will be returned; If I perform another query, say, for category = ‘Cosmetics’, then again all the records will be parsed and ones related to Cosmetics will be returned. But, upon performing dynamic partition for 11 categories, 11 different partitions would take place inside the table with 11 categories, and everytime I fire a query, only these partition names will be parsed to fetch records, i.e. only 11 names will be parsed instead of whole data getting parsed over and over again. Please refer to the output folder and have a look at Hive Dynamic Partition output to get a clear view of how partition directories are formed. Please find the HIVE Script in CODE folder.

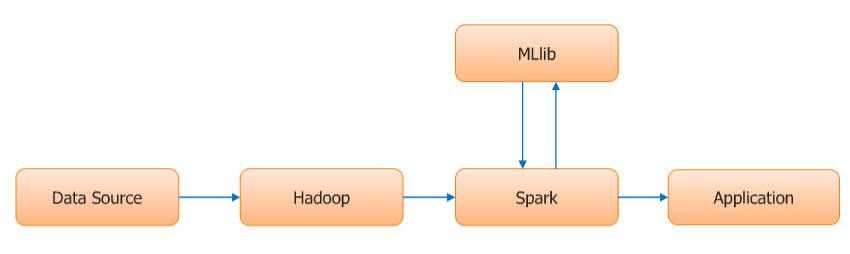
SPARK SQL : Once all the data got stored in Hive, I queried the Hive table and partitions using Spark SQL in order to obtain the total count of reviews for each of the product of every category. Now, Spark SQL works on the concept of dataframes and performs the querying computations in memory, that’s why it is much faster when compared with querying the data using Hive(which works on the concept of Map Reduce). Point to be noted that in case of 100s of Terrabytes of data, Spark SQL would be of no use to query the Hive table, and in that case HIVE QL should be preferred over Spark SQL.

OOZIE : Oozie is basically a job scheduler and a job coordinator for scheduling Hadoop jobs. I’ve written Oozie scripts and XMLs to configure Flume, Map-Reduce, PIG, and HIVE jobs, such that everyday my Flume Job runs from 7pm to 10pm, my Map-Reduce run from 10:15pm, PIG job runs at 10:30 pm, and the HIVE job runs at 10:45 pm. All these jobs have been running for 65-70 days now, and I gathered a total of 33GB of Twitter tweets. Please find the Oozie script and configured xmls in CODE folder.

That was all about the backend pipeline implemented by me.

EVALUATION OF PIPELINE

Now, the evaluation part of this pipeline. In order to check the efficiency of this pipeline, I have implemented the Alternating Least Squares algorithm leveraging the MLLib library of Spark.



I extracted the review frequency(count) information from the tweets. Then i created a tweet object out of it using the tweet case array, where in i've passed the parameters product and the review frequency. Then using the train method i trained the model using the ALS library. This train method expects the tweet object that i obtained earlier, to be passed as one of the parameters, along with the other parameters like no. of iterations, depth, and precision. Thus, we call the train method on ALS library, and it gives us the desired model for the given dataset. Then the model that i obtained after training, using this model i have called the prediction method, where in i am asking the model to predict the reviews for all the products, and then I compared the relative no. of reviews between different products with the relative no. of reviews between that I obtained from the pipeline. For example, from the backend big data pipeline I obtained the no. of reviews for beer as 382,430 while for coffee I got the no. of reviews as 593561. And then predicting the no. of reviews for beer and coffee using the ALS implementation on a smaller scale, I got the results as 7839 for beer and 11048 for coffee. Thus, the efficiency of the pipeline based upon Alternating Least Squares Algorithm can be given as (382430/593561)/(7839/11048) = 0.915. Similarly, efficiency could be obtained for various combination of products and then the mean of efficiencies can be taken to find the eventual efficiency of the pipeline as per Alternating Least Squares algorithm. In order to understand the training and prediction of this model better, please find the ALS algorithm implementation Scala source code in CODE folder.