**Document Finder Using Spark**

Introduction: -

Document Finder finds the relevant document from a list of documents in a directory depending on the query. Querying multiple documents and getting relevant results involves two steps, where in the first step the inverted index with TF-IDF scores is calculated and is saved in the HDFS and in second step relevant results are retrieved with maximum TF-IDF scores. The Document Finder is very fast in returning the relevant documents. The best part of my code is that it first creates the TF-IDF table and saves it in a parquet format. And in the second step it retrieves the relevant document using that table. This reduces the unnecessary computation for calculating the TF-IDF scores.

Creating Inverted Index: -

Inverted index with TF-IDF scores are calculated using the pyspark. Spark Data Frames are used to compute the TF-IDF scores. The code below explains how TF-IDF scores are calculated using command line on pyspark.

Script for Calculating Inverted Index: -

**from** pyspark.sql.functions **import** split,explode,countDistinct,log  
  
*#reading all the files from cricket folder with their locations*text = sc.wholeTextFiles(**'file:///root/Lab/cricket/\*'**)  
*#Counting the total number of files in the folder from the rdd*total\_doc = text.distinct().count()  
*#Converting RDD to DataFrame*df = text.toDF([**'location'**,**'words'**])  
*#Performing word count on the DataFrame*df1 = df.select(**'location'**,explode(split(df[**"words"**],**" "**)).alias(**'word'**))  
df2 = df1.groupBy(**'location'**,**'word'**).count()  
*#Calculating Total words in each file to calculate TF scores*df\_total = df1.groupBy(**'location'**).count()  
df\_total = df\_total.withColumnRenamed(**'count'**,**'total'**)  
df3 = df2.join(df\_total, **'location'**)  
*#Dividing count of word at each location with total number of words in that locations  
#To calculate TF Scores*df4 = df3.withColumn(**"TF"**, df3[**'count'**]/df3[**'total'**])  
*#Counting total number of locations where a word appears*df\_loc = df1.groupBy(df1.word).agg(countDistinct(**'location'**))  
*#Calculating the IDF scores for each word*df\_IDF = df\_loc.withColumn(**"count(DISTINCT location)"**, log(total\_doc/df\_loc[**'count(DISTINCT location)'**]))  
*#df\_IDF DataFrame stores the IDF scores*df\_IDF = df\_IDF.withColumnRenamed(**'count(DISTINCT location)'**,**'IDF'**)  
*#Dropping Unnecessary Columns from DataFrame df4*df4 = df4.drop(**'count'**,**'total'**)  
df\_TF\_IDF = df\_IDF.join(df4, **'word'**)  
df\_TF\_IDF = df\_TF\_IDF.withColumn(**"TF-IDF"**, df\_TF\_IDF[**'TF'**] \* df\_TF\_IDF[**'IDF'**])  
*#Removing unnecessary columns from the final table*df\_Final = df\_TF\_IDF.drop(**'TF'**,**'IDF'**)  
df\_Final.coalesce(1).write.save(**'/user/root/cricket/output/'**, format=**'csv'**,mode=**'append'**)

Explanation for the above script: -

The above code reads the file using wholeTextFiles command which reads all the files in the directory and the best part is it also copies its location. After reading the files the first job is to calculate the word count which would be used to calculate the TF and IDF scores. Then the TF and IDF scores are calculated in different Data Frames and are joined in the later steps. TF scores are calculated in DF4 Data Frame and the IDF scores are calculated in the df\_IDF Data Frame. Finally, both Data Frame are joined after dropping the unnecessary columns from the respective data frames and multiplying the TF and IDF gives us the TFIDF scores.

Querying from The Inverted Index: -

Two methods are used to query the inverted index. Map Reduce and Spark SQL context are used to make the queries. I found that it was faster to make the queries using the spark SQL than using Map Reduce.

Querying Using Spark SQL (Near Real-Time Query): -

Spark SQL is used to make the SQL like queries in spark. The results retrieved using this method are much simple and fast than the map reduce technique. The Data Frame with TF-IDF scores is stored in the parquet format on the HDFS since it is really fast to retrieve the results from the parquet format.

Script to Make Spark SQL Query: -

**from** pyspark.sql.functions **import** desc  
*#Seq is the sequence of query words*seq = **"Nagpur test"***#Number of documents to be returned*n = 3  
*#reading the parquet stored tfidf scores table*df = sqlContext.read.format(**'parquet'**).load(**'/user/root/cricket/output2/'**)  
*#Creating SQL context view for tfidf scores*df.createOrReplaceTempView(**"tfidf"**)  
*#Splitting the words in the query*q = seq.split()  
*#Creating an rdd of words*words = sc.parallelize(q)  
*#Counting total number of words in the query*tot = words.count()  
d = words.map(**lambda** x : (x,tot)).toDF([**'word'**,**'total'**])  
*#Creating SQL context view for query view*d.createOrReplaceTempView(**"query"**)  
*#creating an inner join on query and tfidf scores*df\_combined = sqlContext.sql(**"select a.location,a.word,a.score,b.total from tfidf a inner join query b on a.word = b.word"**)  
*#Creating SQL context view for combined*df\_combined.createOrReplaceTempView(**"combined"**)  
*#Counting the sum of scores and count of words*df\_combined\_with\_count = sqlContext.sql(**"select location, sum(score), count(word) as count from combined group by location"**)  
df\_combined\_with\_intersection = df\_combined\_with\_count.withColumn(**"count"**, df\_combined\_with\_count[**'count'**]/tot)  
df\_final = df\_combined\_with\_intersection.withColumn(**'finalscores'**, df\_combined\_with\_intersection[**'count'**]\* df\_combined\_with\_intersection[**'sum(score)'**])  
df\_final = df\_final.drop(**'sum(score)'**,**'count'**)  
*#Sorting he top results beased on their final scores*df\_final = df\_final.sort(df\_final[**'finalscores'**].desc())  
*#Showing top n results*df\_final.show(n)

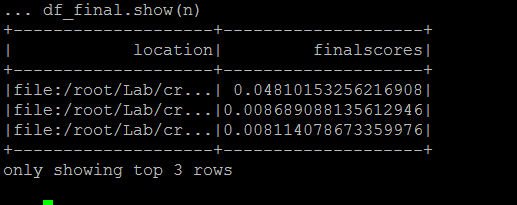
Explanation for The Script: -

In this code I am using spark SQL to get the relevant documents based on their TF-IDF scores. I am creating query data frames with the words from the query. After that I am creating an inner join on the TF-IDF scores and query to get the relevant documents from the table.

Results: -

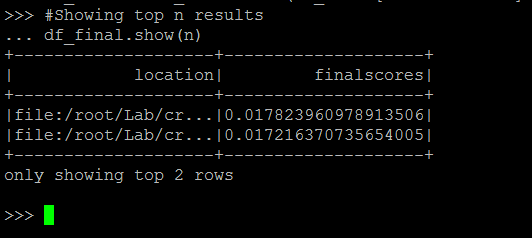
1. For N = 3 and word = Indian star batsman Sachin Tendulkar

Output -



1. For N=2 and word = Nagpur Test Australia

Output -



1. For N=4 and word = India Pakistan One Day

Output -

