```
In [ ]:  TF-IDF Search Using Spark
```

```
In [ ]:
```

```
In [ ]:  # Working with the cricket text files folder
```

```
In [3]:  #importing the data from HDFS into Spark
         #mapping into a, b where a=text file name, b= content of the text file
         from pyspark.sql import SQLContext, Row
         cricket_text = sc.wholeTextFiles('/user/root/crc').map(lambda (a,b): Row(
         le =a.replace('hdfs://sandbox.hortonworks.com:8020',''), text=b) )
```

```
In [5]:  number_of_docs = cricket_text.count()
         number_of_docs

         # output shows its dealing with 124 text files in the cricket folder

Out[5]:  124
```

```
In [28]: import re
         def tokenize(s):
           return re.split("\\W+", s.lower())




         # definition that splits each word of the document and also keeping trac
         k of the file name
```

```
In [13]: # Calculating frequency of each word per document.
         #Used flat map values function
         #Pass each value in the key-value pair RDD through a flatMap function wi
         thout changing the keys;
         #this also retains the original RDD's partitioning.

         term_frequency = tokenized_text.flatMapValues(lambda x:
         x).countByValue()
         term_frequency.items()[:5]

Out[13]: [((u'/user/root/crc/067.txt', u'team'), 5),
          ((u'/user/root/crc/106.txt', u'taking'), 1),
          ((u'/user/root/crc/071.txt', u'ago'), 1),
          ((u'/user/root/crc/014.txt', u'now'), 1),
          ((u'/user/root/crc/103.txt', u'proved'), 1)]
```

In [51]:
```
document_frequency = tokenized_text.flatMapValues(lambda x: x).distinct()
filter(lambda x: x[1] != '').map(lambda (title,word): (word,title)).coun
tByKey()
document_frequency.items()[:5]

#Step 1: taking all the unique words in all the documents
# Step 2: filtering / discarding any null values
# Counting all the unique words in all the docs by creating dictionary
# count by key()
# the idea is is the any word's count is more than 1, it appeared in mor
e than 1 doc
```

Out[51]:
```
[(u'nudges', 1),
 (u'limited', 7),
 (u'devilliers', 1),
 (u'bidding', 1),
 (u'khalil', 9)]
```

In [52]:
```
document_frequency['nudges']
```

Out[52]:
```
1
```

In [ ]:
```
# explaining the tf_idf function

Step 1: taking each element of term_frequency which is in the format[((f
ile,word),TF)]
as a key value Pair

Step2: assigining (filename,word) into list doc and term
Step 3: collecting document frequency of each term of the docuemnt_frequ
ency
function already created
Step4: calculating tf-idf for each word in each document,along with term
 frequncy
Step 5: appending to result
```

In [57]:
```
# Calculating TF-IDF
import numpy as np
from __future__ import division
def tf_idf(N, tf, df):
    result = []
    for key, value in tf.items():
        doc = key[0]
        term = key[1]
        df = document_frequency[term]
        if (df>0):
          tf_idf = float(value)*np.log(number_of_docs/df)

        result.append({"doc":doc, "term":term, "score":tf_idf})
    return result
tf_idf_output = tf_idf(number_of_docs, term_frequency, document_frequenc
y)
tf_idf_output[:4]
```

Out[57]:
```
[{'doc': u'/user/root/crc/067.txt',
  'score': 3.2294714785469991,
  'term': u'team'},
 {'doc': u'/user/root/crc/106.txt',
  'score': 1.9299098077088723,
  'term': u'taking'},
 {'doc': u'/user/root/crc/071.txt',
  'score': 2.6230569882688175,
  'term': u'ago'},
 {'doc': u'/user/root/crc/014.txt',
  'score': 1.0590814499114745,
  'term': u'now'}]
```

In [ ]:
```
Defining a search function
(1) Tokens= taking the query as string an splitting into words
(2) Word search of the  each word in the query in the rdd to create a jo
ined rdd
which gives word, no of times it appeared in that document and tf-idf sc
ore
(3) scount aggregates by key and returns sum of tfidf based on query for
 each document
(4) scores multiplies the sum multiplied with query doc existences in ea
ch document / len(query)
(5) Also does an inverted index
finally returns top score and document name
```

In [83]:
```
tfidf_RDD = sc.parallelize(tf_idf_output).map(lambda x: (x['term'],(x['d
oc'],x['score']) )) # the corpus with tfidf scores

def search(query, topN):
  tokens = sc.parallelize(tokenize(query)).map(lambda x: (x,1) ).collect
AsMap()
  bcTokens = sc.broadcast(tokens)

  joined_tfidf = tfidf_RDD.map(lambda (k,v): (k,bcTokens.value.get(k,'-')
) ).filter(lambda (a,b,c): b != '-' )

  scount = joined_tfidf.map(lambda a: a[2]).aggregateByKey((0,0),
  (lambda acc, value: (acc[0] +value,acc[1]+1)),
  (lambda acc1,acc2: (acc1[0]+acc2[0],acc1[1]+acc2[1])) )

  scores = scount.map(lambda (k,v): ( v[0]*v[1]/len(tokens), k) ).top(to
pN)

  return scores
```

In [84]:
```
# returns the result in less than 5 seconds
search('bangladesh win',5 )
```

Out[84]:
```
[(19.454654995321306, u'/user/root/crc/115.txt'),
 (13.308703690412814, u'/user/root/crc/077.txt'),
 (8.1330195264573124, u'/user/root/crc/039.txt'),
 (7.3474990257663526, u'/user/root/crc/065.txt'),
 (6.9547387754208723, u'/user/root/crc/057.txt')]
```

In [85]:
```
search('australia plays india',3)
```

Out[85]:
```
[(11.36766318009561, u'/user/root/crc/045.txt'),
 (9.1542441822941516, u'/user/root/crc/044.txt'),
 (7.080891939236551, u'/user/root/crc/026.txt')]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```