1.-Practice: Trending Topics & analysis sentiment (5.0%)

Implements the same application realized in activity 2 (mapreduce), but this time using Apache Spark:

1. Calculate de the N Treding Topics (3.0%)
   - Use the json twitter files.
   - Clean-up and filter the input data
   - Calculate the Trending topics (using the word-count and the Top-N models).
   - (optional) Analysis sentiment of hashtags (+1.0%)
   - Store the N Trending topics in a output text file
2. Implement the standalone version of the previous Trending Topics application and study its performance (2.0%).
   - Compare the performance with the mapreduce application.
   - Performance vs Number of executors
   - Performance vs Number of cores
   - Scalability vs input data size
   - Scalability vs number of executors/cores

Deliver the jupyter notebook, the standalone application and a pdf file with the performance analysis.

In [1]:
```
### 1.- Solution: Calculate dataset statistics

```

1a.- Results: Trending Topics & analysis sentiment (tweets2.json)

Input file: tweets2.json with 1000 tweets Language: es 2720 Negative words: [u'voluble', u'decepci\xf3n', u'vomitar', u'anzuelo', u'emboscada', u'dilaci\xf3n', u'inseguro', u'insuficientemente', u'calumnia', u'controversial'] 1553 Positive words: [u'invencible', u'asombro', u'limpiamente', u'consumado', u'sosiego', u'atrevimiento', u'divertir', u'igual', u'integrado', u'reforma'] Languaged filtered data count: 11 First tweet: {'text': u'rt @armyviciconte_: se tiene que hacer yaaaaa #con27estasparaelbailandomica', 'hashtags': [{u'indices': [46, 75], u'text': u'Con27EstasParaElBailandoMica'}]} Processed data count: 11 First tweet: {'text': u'rt @armyviciconte_: se tiene que hacer yaaaaa #con27estasparaelbailandomica', 'hashtags': [u'Con27EstasParaElBailandoMica']}

Result :

- Total number of Hashtags: 16

- First 10 Hashtags: [{'count': 1, 'positive': 0, 'length': 11, 'negative': 0, 'rate': 0.0, 'hashtag': u'FOTOGALER\xcdA'}, {'count': 1, 'positive': 1, 'length': 13, 'negative': 1, 'rate': 0.0, 'hashtag': u'MetGala'}, {'count': 1, 'positive': 0, 'length': 8, 'negative': 0, 'rate': 0.0, 'hashtag': u'Con27EstasParaElBailandoMica'}, {'count': 1, 'positive': 0, 'length': 8, 'negative': 0, 'rate': 0.0, 'hashtag': u'TuitUtil'}, {'count': 1, 'positive': 0, 'length': 8, 'negative': 0, 'rate': 0.0, 'hashtag': u'FrasesDePel\xedcula'}, {'count': 1, 'positive': 1, 'length': 13, 'negative': 1, 'rate': 0.0, 'hashtag': u'PilarMode'}, {'count': 1, 'positive': 0, 'length': 20, 'negative': 1, 'rate': -0.05, 'hashtag': u'Valladolid'}, {'count': 1, 'positive': 2, 'length': 18, 'negative': 0, 'rate': 0.1111111111111111, 'hashtag': u'SoyC'}, {'count': 1, 'positive': 0, 'length': 20, 'negative': 1, 'rate': -0.05, 'hashtag': u'Noticia'}, {'count': 1, 'positive': 0, 'length': 17, 'negative': 0, 'rate': 0.0, 'hashtag': u'OsorioChong'}]

- Trending Topics (Top-10 Hashtags):
- FOTOGALERÍA -> Count: 1, Positive: 0, Negative: 0, Length: 11, Rate: 0.000.
- MetGala -> Count: 1, Positive: 1, Negative: 1, Length: 13, Rate: 0.000.
- Con27EstasParaElBailandoMica -> Count: 1, Positive: 0, Negative: 0, Length: 8, Rate: 0.000.
- TuitUtil -> Count: 1, Positive: 0, Negative: 0, Length: 8, Rate: 0.000.
- FrasesDePelícula -> Count: 1, Positive: 0, Negative: 0, Length: 8, Rate: 0.000.
- PilarMode -> Count: 1, Positive: 1, Negative: 1, Length: 13, Rate: 0.000.
- Valladolid -> Count: 1, Positive: 0, Negative: 1, Length: 20, Rate: -0.050.
- SoyC -> Count: 1, Positive: 2, Negative: 0, Length: 18, Rate: 0.111.
- Noticia -> Count: 1, Positive: 0, Negative: 1, Length: 20, Rate: -0.050.

1b.- Results: Trending Topics & analysis sentiment (tweets.json)

Input file: tweets.json with 204282 tweets Language: es 2720 Negative words: [u'voluble', u'decepci\xf3n', u'vomitar', u'anzuelo', u'emboscada', u'dilaci\xf3n', u'inseguro', u'insuficientemente', u'calumnia', u'controversial'] 1553 Positive words: [u'invencible', u'asombro', u'limpiamente', u'consumado', u'sosiego', u'atrevimiento', u'divertir', u'igual', u'integrado', u'reforma'] Languaged filtered data count: 3495 First tweet: {'text': u'@disneyspain @tinistoessel libera logooo \n#tini', 'hashtags': [{u'indices': [42, 47], u'text': u'TINI'}]} Processed data count: 3495 First tweet: {'text': u'@disneyspain @tinistoessel libera logooo \n#tini', 'hashtags': [u'TINI']}

Result :

- Total number of Hashtags: 2981

- First 10 Hashtags: [{'count': 1, 'positive': 0, 'length': 14, 'negative': 0, 'rate': 0.0, 'hashtag': u'PaisajeCulturalCafetero'}, {'count': 11, 'positive': 0, 'length': 230, 'negative': 25, 'rate':

2.- Practice: Calculate dataset statistics (3.0%)

Using the Datasets/ml-100k/u.item data:

1. Read the file and separate all the fields. textFile, map, split, ... (0.5%)
2. Calculate the following stastistics (0.5%):
   - Number of Movies. [count, ...]
   - First record. [first, ...]
   - First 5 records. [take, ...]
3. Calculate Movies of year X (not take in consideration the blank years) [conver_year, map, filter, reduceByKey, *sortByKey*, collectAsMap, ...] (1.0%)

   - Calculate the max, min, average and standard desviation of the movies by year. [max, min, map, mean,s tdev, ...]
   - Plot a histogram with the number of movies by year (*optional*) [np.arange, map, filter, distinct, plt.hist, plt.gcf,...]
4. Working with partitions (0.5%)
   - Get the Dataset's automatic number of partitions. [getNumPartitions, ...]
   - Change the number of partitions to 10. [repartition, ...]

The u.item file has information about the items (movies); this is a tab separated list with the following fields. movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western | The last 19 fields are the genres, a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several generes at once. The movie ids are the ones used in the u.data data set.

In [] there are the spark funtions required to implement the exercise.

To get the movie year from the date and convert it to integer, you can use the following convert_year function:

In [2]:
```
def convert_year(x):
     try:
       return int(x[-4:])
     except:
       return 0 # there is a 'bad' data point with a blank year, which we set
```

In [3]:
```
### 2.- Solution: Calculate dataset statistics

```

2.- Results: Calculate dataset statistics

Datasets/ml-100k/u.item MapPartitionsRDD[11] at textFile at NativeMethodAccessorImpl.java:-2

1.- Movies fields: PythonRDD[12] at RDD at PythonRDD.scala:48 ()

2.- Number of movies: 1682

2.- First record: [u'1', u'Toy Story (1995)', u'01-Jan-1995', u'', u'http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)' (http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)'), u'0', u'0', u'0', u'1', u'1', u'1', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0']

2.- First 5 records: [[u'1', u'Toy Story (1995)', u'01-Jan-1995', u'', u'http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)' (http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)'), u'0', u'0', u'0', u'1', u'1', u'1', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0'], [u'2', u'GoldenEye (1995)', u'01-Jan-1995', u'', u'http://us.imdb.com/M/title-exact?GoldenEye%20(1995)' (http://us.imdb.com/M/title-exact?GoldenEye%20(1995)'), u'0', u'1', u'1', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'1', u'0', u'0'], [u'3', u'Four Rooms (1995)', u'01-Jan-1995', u'', u'http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)' (http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)'), u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'1', u'0', u'0'], [u'4', u'Get Shorty (1995)', u'01-Jan-1995', u'', u'http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)' (http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)'), u'0', u'1', u'0', u'0', u'0', u'1', u'0', u'0', u'1', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'0'], [u'5', u'Copycat (1995)', u'01-Jan-1995', u'', u'http://us.imdb.com/M/title-exact?Copycat%20(1995)' (http://us.imdb.com/M/title-exact?Copycat%20(1995)'), u'0', u'0', u'0', u'0', u'0', u'0', u'1', u'0', u'1', u'0', u'0', u'0', u'0', u'0', u'0', u'0', u'1', u'0', u'0']]

3.- Movies by year histogram: [(1922, 1), (1926, 1), (1930, 1), (1931, 1), (1932, 1), (1933, 2), (1934, 4), (1935, 4), (1936, 2), (1937, 4), (1938, 3), (1939, 7), (1940, 8), (1941, 5), (1942, 2), (1943, 4), (1944, 5), (1945, 4), (1946, 5), (1947, 5), (1948, 3), (1949, 4), (1950, 7), (1951, 5), (1952, 3), (1953, 2), (1954, 7), (1955, 5), (1956, 4), (1957, 8), (1958, 9), (1959, 4), (1960, 5), (1961, 3), (1962, 5), (1963, 6), (1964, 2), (1965, 5), (1966, 2), (1967, 5), (1968, 6), (1969, 4), (1970, 3), (1971, 7), (1972, 3), (1973, 4), (1974, 8), (1975, 6), (1976, 5), (1977, 4), (1978, 4), (1979, 9), (1980, 8), (1981, 12), (1982, 13), (1983, 5), (1984, 8), (1985, 7), (1986, 15), (1987, 13), (1988, 11), (1989, 15), (1990, 24), (1991, 22), (1992, 37), (1993, 126), (1994, 214), (1995, 219), (1996, 355), (1997, 286), (1998, 65)]

3.- Number of movies year 1994: 214

3.- Maximun movies/year: (1996, 355)

3.- Minimun movies/year: (1922, 1)

3.- Average movies/year: 23.676056338

3.- Standard desviation movies/year: 63.5534098658 ()

4.- Num Partitions: 2

4.- New Num Partitions: 10

Hist Image

3.- Practice: pair RDDs (3.0%)

Using the Datasets/ml-100k/u.data dataset:

1. Read the ratings data and convert to numeric values (ids to integer and rating to float). [textFile, map, split, int(), float(), ...] (0.5%)
2. Create a pair RDD with the following information: (user id, (rating, item id)) [map, ...] (0.5%)
3. Calculate for each user the sum of the ratings and the number of movies rated. [aggregateByKey, ...] (0.5%)
4. Calculate the average of ratings by movie for each user [mapValues, ...] (0.5%)
5. Calculate how many ratings did each movie receive. map, reduceByKey, ... (0.5%)
6. High Rating Movies: How many movies had a higher than average (3) rating [map, filter, mapValues, reduceByKey, ...] (1.0%)
   - Map the data to movie ID and rating.
   - Filter the data only for those records with ratings 4 or higher.
   - Map the data to movie ID and the number 1.
   - Add each row of data together.
7. Print the Top 5 the Last 5 rated movies [top, takeOrdered, sortBy,...] (0.5%)
8. Join the two movie_counts and high_rating_movies datasets using a leftOuterJoin [leftOuterJoin, tale, sortByKey, collect,...] (1.0%)
   - Print the first 5 elements from the orignal and joining datasets.
   - Print the data for the movid with id = 314.9
9. Calculate the percent of ratings that are higher. [mapValues, top,...] (0.5%)
   - Print the Top 10.

You can choose the apartats you want to implement.

The u.data file has full u data set, 100000 ratings by 943 users on 1682 items.

- Each user has rated at least 20 movies.
- Users and items are numbered consecutively from 1.
- The data is randomly ordered.

This is a tab separated list with the following fields.

```
user id | item id | rating | timestamp.
```

In [4]:
```
### 2.- Solution: pair RDDs (3.5%)
```

3.- Result: pair RDDs (3.0%)

1.- User ratings (user id, item id, rating, timestamp):[u'196', u'242', u'3', u'881250949']

1.- Numerical user ratings (user id, rating, item id): (196, 3.0, 242)

2.- Pair RDD (user id, (rating, item id)): (196, (3.0, 242))

3.- Agregate user ratings and movies ((user id, (rating sum, number of movies)): [(2, (230.0, 62.0)), (4, (104.0, 24.0)), (6, (767.0, 211.0)), (8, (224.0, 59.0)), (10, (774.0, 184.0))]

4.- User average ratings ((user id, rating sum/number of movies): [(2, 3.7096774193548385), (4, 4.333333333333333), (6, 3.6350710900473935), (8, 3.7966101694915255), (10, 4.206521739130435)]

5.- Movie number of ratings (movie id, ratings number): [(2, 131), (4, 209), (6, 26), (8, 219), (10, 89)]

6.- 1447 high rating movies (movie id, high ratings number): [(2, 51), (4, 122), (6, 15), (8, 155), (10, 59)]

7.- Top 5 rating movies: [(50, 501), (100, 406), (181, 379), (127, 351), (174, 348)] [(50, 501), (100, 406), (181, 379), (127, 351), (174, 348)] [(50, 501), (100, 406), (181, 379), (127, 351), (174, 348)] 7.- Last 5 rating movies: [(36, 1), (440, 1), (548, 1), (556, 1), (600, 1)]

8.- Movie rating counts dataset (movie id, ratings number): [(2, 131), (4, 209), (6, 26), (8, 219), (10, 89)]

8.- High rating movies dataset (movie id, high ratings number): [(2, 51), (4, 122), (6, 15), (8, 155), (10, 59)]

8.- Join (movie id, (ratings number, high ratings number): [(2, (131, 51)), (4, (209, 122)), (6, (26, 15)), (8, (219, 155)), (10, (89, 59))]

9.- Info for movie id 314: (314, (5, None))

9.- Top higher rates movie (movie id, (high ratings number/ ratings number): [(814, 1.0), (1064, 1.0), (1080, 1.0), (1122, 1.0), (1130, 1.0), (1396, 1.0), (1398, 1.0), (1452, 1.0), (1458, 1.0), (1482, 1.0)]

In [5]:
```
from IPython.core.display import HTML
def css_styling():
    styles = open("./styles/Exercise.css", "r").read()
    return HTML(styles)
css_styling()
```
Out[5]: