Distributed Systems: Big Exercise 4

Jakub Kubišta

Code is written in python version 2.x and you can start program by following execution:

python spark_example.py

Program is also generally documented inside of code. The results of the third task, are in files h-data-10.txt, h-data-100.txt and h-data-1000.txt. I used for creation of program ukko004.

Requirements

1. Calculate minimum, maximum, average, and variance for file data-1.txt:

    a. Minimum = 0.00000011
    b. Maximum = 99.99999990
    c. Average = 50.00096992
    d. Variance = 833.34032224

    I created these functions separately (not in one function calculate all), because we can use them separately every time for different data set in this format. At first, I tested difference between average functions (simple against Spark one). And results were on the simple file (data-1-sample.txt):

```
Time of execution for avg with Spark (original): 5.1286740303 seconds
Time of execution for avg with Spark (alternative): 1.33336901665 seconds
Time of execution for avg without Spark: 0.00135684013367 seconds
Average = 50.19059339
Minumum = 0.02928808
Maximum = 99.97232931
Mode = 39.71426400
kubista@ukko004:~$
```

    Where “original” function was implemented from previous example, “alternative” using mean function and the last function is without any use of Spark.

    Functions looks after mapping as following:

```
data = data.map(lambda s: float(s))
avg_value = data.mean()
min_value = data.min()
max_value = data.max()
variance_value = data.variance()
```

    Then I tried to use same functions on file data-1.txt and it brought the following results:

```
Time of execution for avg with Spark (original): 640.904625177 seconds
Time of execution for avg with Spark (alternative): 1225.79683995 seconds
Time of execution for avg without Spark: 375.461700916 seconds
```

    As we can see time of execution for different average functions is absolutely different on bigger data set. If we use mapping and then execute all function with Spark and without Spark. We can find, how much is Spark useful. I create also function which brings all results (avg., max., min.,

var.) together. Function on average computer takes:

```
Time of execution for functions: 2742.92136502 seconds
Average = 50.00096992
Minumum = 0.00000011
Maximum = 99.99999990
Variance = 833.34032224
```

(ie. 45 minutes)

Results have been already said, but this function was created only for testing in this topic. In practise is better to use these functions separately.

2. Compute the mode for the data set:

The mode of a set of data is the value in the set that occurs most often. A set of data can be bimodal. It is also possible to have a set of data with no mode.

Data which we used are unsorted. For mentioned function, would be better to have sorted data by the most frequent value or at least partially sorted data. I also think, that for this function file contains too redundant data – it is maybe good for detecting and correcting mistakes of data, but for these interval functions it is worse.

To improve effectivity of functions like mode would be very useful transform data set into well-structured data set, sometimes it's not always possible, so we can just improve this area.

3. Calculate three histograms:

This task represents function calculate_histogram(fn, hfn,n), whre fn represents input data file, hfn output data file and n is number of bins. At first was necessary to create array for number of bins, where every node represents edge of the range for histogram. The first attempt was as follows:

```
..
if (n == 100):
    for x in range(0, n + 1):
        histo_array.append(int(x))
..
```

But then I realized, that python could have some library for this. So, I used NumPy, which is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Then I tried to use arange from np library, but it was looking too complicated to understand for the first view, as we can see on the following row.

```
..
histo_array = np.arange(0, n + 1, 0.1)
..
```

I finished at function Linspace, which returns evenly spaced numbers over a specified interval. It's basicly decimal range() step value function, which lets you have control over what happens at the endpoin. But after that is necessary to change linspace into array.

```
..
histo_lin = np.linspace(0,100,n+1)
..
```

After I mapped data, I used function histogram, which takes prepared histo_array and created buckets, where first array represents histo_array and second is count of numbers between each range:

```
..
histogram_data = data.map(lambda s: float(s)).histogram(histo_array)
..
```

Then are data written into files as it was in requirements.

I also used Excel program for sure, that histogram function count numbers between intervals correctly, like this (view with sample data):

5	66,56262					
6	0,280959				94	
7	3,46182				107	
8	55,99361				90	
9	70,03421				111	
10	82,17355				94	
11	33,84224				106	
12	80,89065				98	
13	44,11505				105	
14	19,61453				98	
15	99,12124				97	
16	83,02874				8	
17	80,2871				11	
18	10,17377					
19	77,76391					
20	88,88586					