Apache Spark -- "Lightning-fast cluster computing"

Hou, Jue

[jue.hou@helsinki.fi](mailto:jue.hou@helsinki.fi)

* Solution
  + Problem 1

Map each data into a vector of 4 dimensions. The first dimension of all the vectors is assigned 1. The rest 3 dimensions of all the vectors are assigned to be the value of each origin data, which is transferred from each line of the text file. The form can be described as followed:

(1, (number), (number), (number))

Reduce the vectors with different operations on different dimensions. For the first two dimensions, get the summation of each dimension. For the third dimension, compare each third dimension of each vector and get the greater one. Operate like the third dimension, but get the smaller value. Return the new vector after reducing. Operation can be described as followed:

(i[0]+j[0], i[1]+j[1], max(i[2],j[2]), min(i[3],j[3]))

In the end, I can get a vector of each dimension stands for the length of the data, the summation of all data, the maximum value of all data and the minimum value of all data. I can hence get the average value as well

Re-map the data into value from text file and doing the calculation according to the formula for variance at the same time. Reduce them by doing the summation. After division, I can therefore get the variance value.

Alternatively, embedded functions mean(), max(), min() and variance() can also applied to achieve the required result.

* + Problem 2
    - Plan

Map the data into key-value pairs from text file. The key of each pair is the string of each line of the text file. And the value is uniformly assigned to be 1.

Reduce the pairs by key. Add the value of two pairs together if their keys are the same. Eventually, I will get pairs which stand for data number and how many times they appeared.

Reduce further to get the pair(s) with the greatest value. If two pairs have the same value, merge their key. After that, I will get a pair. And the mode is the key of that pair.

* + - Explanation of finding the mode poorly

In order to find the mode, the times of each data appeared have to be counted. Theoretically, no matter how is this achieved, the mode is always the data appears most. However, in the given dataset all the data might appear only a few times, which means those data, if compared to the whole amount, will be recognized as the mode. And there might be several data appear the equally same few times. In the case of sample dataset, every data appears exactly once. According to the definition, every item is a mode. Practically, according to my plan, I first aggregate pairs with the same key to form clusters. After clustering, there appears to be a large of amount of unique clusters and each of them only has a very small number of members. Then, processing the clusters can take a long time and is of low efficiency. Depending on the algorithm, function may not even be deterministic when there is several data which appear equally same amount of times.

In a word, it takes a long time to process the dataset and can only produce item(s) with low frequency. Also, the dataset contains several items with all the same low frequency, which may lead to an uncertain result or a large amount of results.

If let me select the dataset, I will pick something represents for finite classes or datasets which allows to aggregate and produce a relatively small amount of clusters. Also, data in the dataset should not be uniformly distributed. Maybe reducing the precision of the given dataset can also do the trick if there is no harm for the further computation.

* + Problem 3

Like the plan mentioned in Problem 2, map the data into key-value pairs. The value is uniformly assigned to 1. But the key will have to turn into decimal number, get divided by granularity and turn into ceiled integer. This process can turn each data into the ID of the bin which the data should be categorized in. The pseudo code for the whole operation is as followed:

int(ceil(float(data)/ granularity))

Perform ReduceByKey operation as Problem 2，I will eventually get pairs stand for the bin ID and the amount of data which can be categorized into that bin. After that, sort according to the bin ID.

However, in case some bins contain no data, bin ID have to be checked and zeroes have to be added when writing file.

Alternatively, embedded function histogram() can also achieve the same result with proper setting.

* Execution Process

I made three code files for three problems accordingly. All of them are modified from the given example. Detailed operations are illustrated above. I will briefly explain the process of my algorithms and omit initiating setting process for Spark system.

* + statistics.py for Problem1
    - Set the text file path
    - Map the data into 4 dimensional vectors
    - Reduce the vectors with three different operations on each dimension and return the new vector.
      * (i[0]+j[0], i[1]+j[1], max(i[2],j[2]), min(i[3],j[3]))
    - Get the average value by the division between the first two value
    - Re-map the data from text file to new values of

* + - Reduce the new values by summation
    - Divide the sum by the value of first dimension to get variance
    - Return the average value, the value of third dimension of the reduced vector as maximum value, the value of fourth dimension of the reduced vector as minimum and variance value
  + mode.py for Problem 2
    - Set the text file path
    - Map the data into key-value pairs
    - Reduce the pairs according to their key. Merge the pair with same key and get the sum of their value.
    - Reduce further.
      * If the value of two pairs is different, return the pair with the greater value.
      * If not, merge keys for those pairs. And then return.
    - Return the key(s) of the result as the mode
  + bin.py for Problem 3
    - Check the existence external argument. If not, program exit.
    - Calculate the granularity according to the number of bin and the whole range of data. Broadcast granularity to each node to speed up.
    - Set the text file path
    - Map the data into key-value pair
    - Reduce the pairs according to their key
    - Sort pairs according to their key in an increasing order
    - Traversal all the processed pairs and write file
      * If bin ID exists among the key of pairs, write its value
      * If there is a missing ID, write a zero
* Running Instruction
  + Environment: Python2, Ukko
  + After file uploading, file permission setting and logging in Ukko, run the file by input “./statistics.py”, “./mode.py” and “./bin.py [argument]” in the shell
    - The argument for bin.py is to specify the number of bins. For Problem 3, it should be 10,100 or 1000
  + The answer for the Problem 1
    - Avg.= 50.0009699185
    - Max.= 99.9999999
    - Min.= 1.1e-07
    - Var.= 833.340322243
  + The mode for Problem 2
    - Numbers listed below are the modes. And they all appears 6 times
      * 15.63279621
      * 47.80592216
      * 5.60493080
      * 71.78279202
      * 16.33257995
      * 95.93253164
      * 80.64623352
      * 74.47111395
      * 19.92614212
  + The files for Problem 3 is attached within the zip package