7CCSMBDT

Coursework 1

1. **Big data collection using Apache Sqoop :** Describe three features of Apache Sqoop that help import data into a distributed file system efficiently. Your description should state the feature and a brief justification of what the feature does and how it can help efficiency.

The three features of Apache Sqoop that help import data into a distributed file system efficiently are :

- Compression import : This feature helps the system to save space by "compressing" the structure of the data in such a way that data uses less space on the distributed file system. This feature increases efficiency because the user would increase the distributed file system's allocated memory space less frequently.
- Parallel import : This feature transfers data to the distributed file system in parallel by having different mappers working at the same time. This feature increases efficiency because having mappers working together is faster than having a mapper working on its own.
- Incremental import : This feature updates the distributed file system by only transferring the modified data to the system rather than transferring the whole dataset. This feature increases efficiency because it requires to send less data to the distributed file system, thus to do data transfers faster.

2. **Task 2 :** Provide the output file (result) from applying the program to the first 10 lines of the dataset. Also, provide your code along with comments and description about what each step does.

Here is the output file from applying the program to the first 10 lines of the dataset :

```
898    "36"

888    "31"

886    "34"

877    "23"

876    "35"

875    "33"

867    "28"
```

```
861   "30"

858   "37"

841   "25"
```

Here is the code for task 2 :

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
WORD_RE = re.compile(r"[\w']+")

#-- list of global values I use either to store some lists so it is easier to
compute with them either some counters so things can happens only once
values = []
keys = []
counter = 0

class task(MRJob):

#-- basic mapper that get the age value from the inputted file
    def mapper(self, _, line):
      data=line.split(", ")
      if len(data)>=2:
        age=data[0].strip()
        yield (age, 1)

#-- basic reducer that count ages
    def reducer1(self, key, list_of_values):
      global values
      global keys
      sumA = sum(list_of_values)
      yield key, sumA
      keys.append(key)
      values.append(sumA)

#-- reducer that output the maximum number of age and remove it from the map
    def reducer2(self, key, list_of_values):
      global counter
      global values
      global keys
      if counter < 1:
        while len(values):
          yield max(values), keys[values.index(max(values))]
          keys.pop(values.index(max(values)))
          values.remove(max(values))
```

```
       counter = counter + 1

#-- basic step
      def steps(self):
         Return[MRStep(mapper=self.mapper,reducer=self.reducer1)
         ,MRStep(reducer=self.reducer2)]

#-- basic start
if __name__ == '__main__':

      task.run()
```

Here is the description of how the code works :

- The first lines are just imports so things could work
- Then, we have some global values. Either to store some lists so it is easier to compute with them either some counters so things can happens only once
- Then, the mapper that gets the age value from the inputted file by getting the first column of the inputted file.
- Then, the first reducer does two things, count ages and set the global lists.
- Then, the second reducer does two things, get the maximum number of age and remove it from the global lists. With the use of a counter I make it happen only once.
- Then, step so the order of execution is Mapper, 1st Reducer and 2nd Reducer.
- Finally, the start of the program.

3. **Task 3 :** Provide the output file (result) from applying the program to the first 10 lines of the dataset. Also, provide your code along with comments and description about what each step does.

Here is the output file from applying the program to the first 10 lines of the dataset :

```
"1"   [1, 5, 7]
"2"   [2, 3]
"3"   [3, 10]
"4"   [3, 10]
"5"   [4, 10]
"6"   [6, 8, 9, 10]
"7"   [9]
"8"   [9]
"9"   [10]
"10"  [10]
```

Here is the code for task 3 :

```
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
WORD_RE = re.compile(r"[\w']+")

#-- list of global values I use either to store some lists so it is easier to
compute with them either some counters so things can happens only once
counter = 0
counter2 = 0
datas = []
result = []

class task(MRJob):

#-- mapper that get the id value, the all the lines without the ids, and
initialize the output from the inputted file
      def mapper(self, _, line):

        global counter
        global datas
        data=line.split()
        id=data[0].strip()
        data.pop(0)
        datas.append(data)
        result.append([])
        if counter < 10:
          yield 1, 1
        counter = counter + 1

#-- reducer that setup and returns the output, here is an example of how it
works: if 6 is in row 8, then output[6].append(8)
      def reducer1(self,key, list_of_values):
        global counter2
        global datas
        global result
        counter3 = 1
        if counter2 < 1:
          for id in datas:
            for data in id:

              result[int(data)-1].append(counter3)

            counter3 = counter3 + 1
          for x in result:
            yield str(result.index(x)+1), [i for n, i in enumerate(x) if i not
            in x[:n]]
        counter2 = counter2 + 1
```

```
#-- basic step
      def steps(self):
        return [MRStep(mapper=self.mapper, reducer=self.reducer1)]

#-- basic start
if __name__ == '__main__':
      task.run()
```

Here is the description of how the code works :

- The first lines are just imports so things could work.
- Then, we have some global values. Either to store some lists so it is easier to compute with them either some counters so things can happen only once.
- Then, the mapper that gets the id value, all the lines without the ids, and initialize the output from the inputted file.
- Then the reducer that sets up and returns the output, here is an example of how it works: if 6 is in row 8, then output[6].append(8).
- Then, step so the order of execution is Mapper and Reducer.
- Finally, the start of the program.