MapReduceSmalFiles

March 1, 2022

[1]:
```
from SmallFilesContainer import SmallFilesContainer
from MapReduceEngine import MapReduceEngine
from VirtualBigFile import *

# general
import os
import time
import random
import warnings
from tqdm import tqdm
import pickle
from io import StringIO

# ml
import numpy as np
import scipy as sp
import pandas as pd

# visual
import seaborn as sns
import matplotlib.pyplot as plt

# notebook
from IPython.display import display
warnings.filterwarnings('ignore')
random.seed(0)
```

[2]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

```
<IPython.core.display.Javascript object>
```

[3]:
```
smallFilesContainer = SmallFilesContainer("MapReduceSmallFiles.csv")

smallFilesContainer.deleteAllFiles()
```

```
def get_input_filename(i:int):
    return "my_input_file_{:05d}.csv".format(i)
```

[4]:
```
def createDatasets(num_files=100000, rows_in_file=10):
    print("Creating {} input files. Each file contains {} rows. Each row␣
 ↪contains: firstname,city,secondname"\
          .format(num_files,rows_in_file))
    firstname  = ['John', 'Dana', 'Scott', 'Marc', 'Steven', 'Michael',␣
 ↪'Albert', 'Johanna']
    city       = ['NewYork', 'Haifa', 'Munchen', 'London', 'PaloAlto', ␣
 ↪'TelAviv', 'Kiel', 'Hamburg']
    secondname = ['Smith', 'Brown', 'Miller', 'Watson', 'Bain']

    filenames = []
    for i in tqdm(range(num_files)):
        first     = np.random.choice(a=firstname,  size=rows_in_file)
        cit       = np.random.choice(a=city,       size=rows_in_file)
        second    = np.random.choice(a=secondname, size=rows_in_file)
        df        = pd.DataFrame({'firstname': first, 'city': cit, 'secondname':
 ↪ second})
        file_name = get_input_filename(i)
        filenames.append(file_name)
        smallFilesContainer.createNewFile(file_name,df.to_csv(index=False,␣
 ↪header=True),deleteExist=True)
    return filenames
```

[5]:
```
filenames = createDatasets()
```

```
Creating 100000 input files. Each file contains 10 rows. Each row contains:
firstname,city,secondname

100%|                                        |
100000/100000 [01:01<00:00, 1626.29it/s]
```

[6]:
```
def read_df_from_csv(filename:str, delete:bool, header:bool):
    tuples  = smallFilesContainer.readFile(filename, type_=[tuple])
    if delete:
        smallFilesContainer.deleteFiles(filename)
    return pd.DataFrame(tuples[1:],columns=tuples[0]) if header else pd.
 ↪DataFrame(tuples)

def map_output_filename(threadID: int):
    return "map-output-{}.csv".format(threadID)

def map_process(threadID, input_filenames):
    tuples = [('key', 'value')]
    for filename in input_filenames:
        data = read_df_from_csv(filename, delete=False,header=True)
        # iterate through different columns to find location of each key-value␣
↪pair
        for col in data.columns:
            tuples.extend([(col + '_' + value, filename) for value in data[col].
↪values])
    # still using big files because each thread can create a huge partition␣
↪that consists from many small input files
    output_filename = map_output_filename(threadID)
    outputFile = VirtualBigFile(output_filename)
    outputFile.delete()
    outputFile.append(tuples)
    outputFile.flush()
```

[7]:
```
def shuffle_read_temp_from_input(threadID):
    filename = map_output_filename(threadID)
    bigFile  = VirtualBigFile(filename)
    tuples   = bigFile.readData(type_=[tuple])
    bigFile.delete()
    return pd.DataFrame(tuples[1:],columns=tuples[0])
```

[8]:
```
def reduce_process(threadID, shuffle_rows):
    tuples = []
    for shuffle_row in shuffle_rows:
        value, documents = shuffle_row[0], shuffle_row[1]
        '''This function takes a value pair and its locations and places them␣
↪in alist without duplicates'''
        #split docs into list and set them to to remove duplicates
        docs = sorted(list(set(documents.split(','))))
        #generate output list
        tuples.append((value, ':'.join(docs)))
    # still using big files because each thread can create a huge partition␣
↪that consists from many small input files
    output_filename = "part-{}-final.csv".format(threadID)
    outputFile = VirtualBigFile(output_filename)
    outputFile.delete()
    outputFile.append(tuples)
    outputFile.flush()
```

[9]:
```
MapReduceEngine.execute(filenames, map_process, shuffle_read_temp_from_input,␣
↪reduce_process, max_threads=8)

smallFilesContainer.flush(objectStorageFlush=True)
```

```
Starting Map stage with 100000 input objects splitted to 8 threads…
Map thread 0 is starting with 12500 objects …
Map thread 1 is starting with 12500 objects …
Map thread 2 is starting with 12500 objects …
Map thread 3 is starting with 12500 objects …
Map thread 4 is starting with 12500 objects …
Map thread 5 is starting with 12500 objects …
Map thread 6 is starting with 12500 objects …
Map thread 7 is starting with 12500 objects …
Map thread 0 is completed
Map thread 6 is completed
Map thread 2 is completed
Map thread 3 is completed
Map thread 1 is completed
Map thread 5 is completed
Map thread 7 is completed
Map thread 4 is completed
Map stage completed in 50.90843677520752 seconds.
Starting Reduce stage with 21 input objects splitted to 8 threads…
Reduce thread 0 is starting with 3 objects …
Reduce thread 1 is starting with 3 objects …
Reduce thread 2 is starting with 3 objects …
Reduce thread 3 is starting with 3 objects …
Reduce thread 4 is starting with 3 objects …
Reduce thread 5 is starting with 2 objects …
Reduce thread 6 is starting with 2 objects …
Reduce thread 7 is starting with 2 objects …
Reduce thread 0 is completed
Reduce thread 1 is completed
Reduce thread 7 is completed
Reduce thread 3 is completed
Reduce thread 2 is completed
Reduce thread 5 is completed
Reduce thread 4 is completed
Reduce thread 6 is completed
Reduce stage completed in 1.2888920307159424 seconds.
MapReduce Completed in 64.06488513946533 seconds.
```

[10]:
```
#VirtualBigFile.deleteFiles(filenames)
#VirtualBigFile.flushFiles(filenames, objectStorageFlush=True)
```