Implementing MapReduce Programming Framework using ZeroMQ sockets

CS403/534 - Distributed Systems
Assignment #6 for Spring 2020

E. Savaş
Computer Science & Engineering
Sabancı University
İstanbul

April 3, 2020

Abstract

In this assignment, you will implement a MapReduce framework using ZeroMQ socket comunication. Using this framework, you will develop an application that reads a text file, and computes the number of words in the file and the number of occurences of a given "keyword".

1 Background

MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment. For this assignment, you will implement a toy MapReduce framework. For detailed information about map-reduce programming strategy, please check:

`https://www.edureka.co/blog/mapreduce-tutorial`

For implementation, you are required to use ZeroMQ Pipeline Pattern with Python programming language. For pipeline pattern, please check:

`https://learning-0mq-with-pyzmq.readthedocs.io/en/latest/pyzmq/patterns/pushpull.html`

2 Description

The MapReduce framework contains a single abstract class, with various functions as follows:

```
abstract class MapReduce:
  def MapReduce(NumWorker:Int) # constructor
  abstract def Map(map_input):Partial Result
  abstract def Reduce(reduce_input):Result
```

```
private def Producer(producer_input)
private def Consumer()
private def ResultCollector()
def start(filename , keyword=None)
```

3 Explanations

- The constructor sets the number of workers, `NumWorker`.
- `Map` and `Reduce` functions are abstract, therefore must be implemented by a subclass. The `Map` function should map its input to a partial result. Then, the `Reduce` function merges those partial results and outputs the result of the computation. In real world applications, the heavy work is done in the `Map` phase. `Map` and `Reduce` functions are implemented by sub-classes which define the work as in the examples given in Section 4.
- `Producer`, `Consumer` and `ResultCollector` functions are implemented in the `MapReduce` abstract class and should not be reachable from outside of the class.
- `Start` function takes external input(s) and prints out the results at the end. First, it starts the following functions as *separate processes*:
  - `ResultCollector`,
  - `Consumer`: as many as `NumWorker`,
  - `Producer`.

  The result produced by the `ResultCollector` is the result of the execution. Please see

  `https://docs.python.org/3/library/multiprocessing.html#multiprocessing.Process`

  for more information on multi-processing. Check also usage samples in the file `zmq_with_processes.py`, provided in the homework package.
- `Start` function passes `map_input` to `Producer`, which partitions and distributes the work to `Consumer` processes. Each `Consumer` process works on its (partial) input using the `Map` function; then its sends its output as a partial result to `ResultCollector`. `ResultCollector` merges the partial results using the `Reduce` function. The final result is printed out.
- Data transfer between `Producer` and `Consumer` as well as `Consumer` and `Resultcollector` must be done using json objects on zmq sockets. On the other hand, you can assume `Resultcollector` and main process (parent process) can communicate via shared memory map using `Value` or `Array` from `multiprocessing` module.

4 Examples

This section contains the pseudo code of example sub-classes of the abstract class `MapReduce` and their usage. `FindWordCount` class takes the name of a text file and the start and end line numbers in the file, and outputs the number of words in it. An instance of `Map` function by a consumer is supposed to take a string and returns its word count while `Reduce` function takes the outputs of `Map` functions and outputs their sum.

```
class FindWordCount extends MapReduce:
  def Map({"filename": fname, "startno": n1, "endno":n2})
                    # returns the number of words in String
  def Reduce(reduce_input)
                       # returns the sum of integers
                         returned by Map

finder = FindWordCount(5) # instantiate with 5 workers
finder.start('sample_01.txt') # prints out 375
```

Another example is to find the number of occurences of a keyword in a text file:

```
class FindWordFrequency extends MapReduce:
  def Map({"filename": fname, "startno": n1, "endno":n2,
                              "keyword":kword})
                              # returns the number of
                                occurences of keyword
  def Reduce(reduce_input)  # returns the sum of integers
                              returned by Map

counter=FindWordFrequency(7) # instantiate with 7 workers
counter.start('sample_01.txt', 'or']) # prints out 30
```

5 Subclasses to implement

You are required to implement the subclasses for the following applications:

1. **FindWordCount**: A subclass of `MapReduce`, which finds the number of words in a text file. The name of the text file is the parameter of the start function.

2. **FindWordFrequency**: A subclass of `MapReduce`, which finds the frequency of a given keyword. The name of the text file and the keyword are parameters of the start function.

6 Program Flow

You should create a `main.py` with the following command line arguments:

- The first parameter is either COUNT or FREQ which switches between **FindWordCount** and **FindWordFrequency**.

- The second parameter is the number of workers. You should support as many as 10 of them.
- The third parameter is the name of the file that you will process.
- If the **FindWordFrequency** example is chosen by the first parameter, the fourth parameter is the query keyword. Otherwise, it is ignored if provided in the command line.

Please check,

`https://www.tutorialspoint.com/python/python_command_line_arguments.htm`

In the following, we have two example runs of the program:

```
python main.py COUNT 5 sample_01.txt
python main.py FREQ 10 sample_01.txt or
```

In the first example, the program prints out 375; it prints out 30 in the second example.

7 Notes

- The deadline is March 12, 2020 @11:55pm.
- You can work in groups of two.
- Implement the framework in `map_reduce.py`
- Implement **FindWordCount** in `find_word_count.py`
- Implement **FindWordFrequency** in `find_word_frequency.py`
- Implement your main in `main.py`
- Submit your assignments through SUCourse and name ALL files using the format "`CS403_Assign2_SUusername1_SUusername2.zip`" or "`CS534_Assign2_SUusername1_SUusername2.zip`" etc.
- One submission per group is sufficient.
- For assistance write to Tolun Tosun (toluntosun@sabanciuniv.edu), Şeyma Selcan Mağara (sselcan@sabanciuniv.edu) or Erkay Savaş (erkays@sabanciuniv.edu).
- **Office hours:** Thursday 17:40-18:30 @ZOOM (Tolun Tosun) (Only for programming assignment); Thursday 11:40-13:30 @ZOOM (Şeyma Selcan Mağara); Wednesday 14:40-16:30 @ZOOM (Erkay Savaş). Please send and e-mail if you plan to come, so we can set the ZOOM meeting.