Parallel Web Scraping

06.02.2022

Nizar El Mouaquit
EIDIA
UEMF

Overview

The project consists of creating a web scraping to fetch and gather to store the data locally for further analysis .

And then parallelize section of the code to lower the processing time

Goals

1. Create a sequential code for the scraping purpose
2. Parallelize the code .
3. Benchmark the different versions .

Specifications

Web Scraping is a super wide field for this project we focused on scraping the price of books within a csv file that contains over 1000 links .

The Website that we used for scraping is actually made for scraping purposes since web scraping is illegal because it violates the intellectual property of some of the websites and can also crash the server due to the high amount of requests in such a small period of time comparable to a DDoS Attack

Our code may not even work because it won't be able to go over the RECAPTCHA (used to differentiate between bots and actual humans surfing the web) and therefore would be stuck ,crash or even take a longer time even with a parallelisation .

Code Analysis

I. Sequential Version

Our Code is divided into three section each of it is responsible for a certain task :

- Section 1 :

```
with open('links.csv', 'r') as f:
        csv_reader = reader(f)
        for row in csv_reader:
        urls.append(row[0])
```

This part is responsible for getting the urls from the csv file to assign to the function responsible for the scrapping

- Section 2 :

```
def go_scrappy(url):
        #Gets the request with the url and passes to BeautifulSoup
        request =  requests.get(str(url))
        #Gets the HTML CODE/DATA FROM THE WEB PAGE
        codeH   =  BeautifulSoup(request.content, 'html.parser')
        # fetches the data from codeH to get the data we want
        price   =  codeH.find('p').text
        #passes the fetched data to the DATA_FRAME "PRICES"
        PRICES.append(price)
        #Prints the data within the console
        print(price)
        return
```

This part is the core of the code since the go_scrappy fetches the data from the url given to by putting it into a request form and then gives it to the BeautifulSoup Function to get the complete HTML code, following that is to find what we want from the HTML code for our it would be the <CLASS P> since this where the data that we want is stored so our function then try to find where the data is located and assign to the PRICE variable

- Section 3 :

```
DataFrame = pd.DataFrame(PRICES)
#Converts data frame to a csv File
DataFrame.to_csv('parralle_prices.csv', index=False)

#Prints the Message once everything is done
print('Done')
```

This part is responsible for getting the data generated from the go_scrappy function and writes it into a csv file to store the data locally for further analysis

II. Parallel Version

Since there's actually no differences apart from the way the function is executed where only gonna focus on the part that made the parallelization possible.

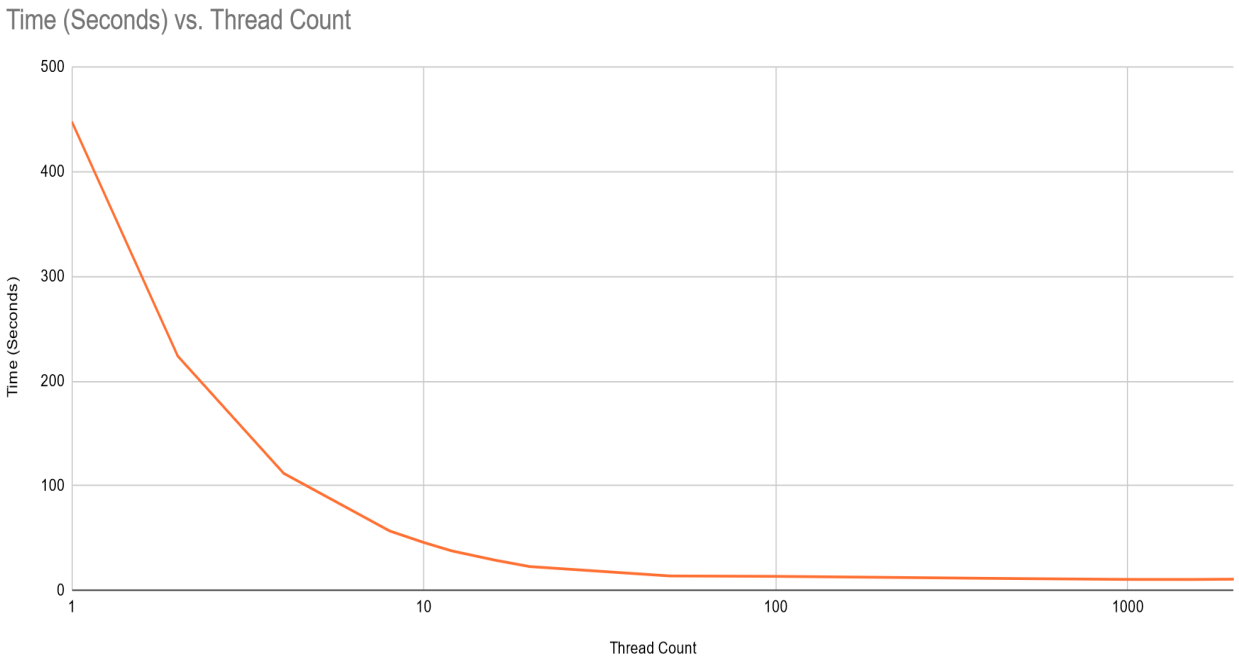
```
with concurrent.futures.ThreadPoolExecutor(Num_of_threads) as executor:
        #Executes in the functions
        executor.map(go_scrappy, urls)
```

Benchmarking

For our best shot with best conditions united we managed to squeeze down the time from 448 Seconds (Approx 8 Minutes 21 Seconds) to 10 Seconds only Using a machine running ArchLinux using an I7-9750H pulling approx 70W on a Thread Count equal to 1500 finding as the best Spot .

- ### Impact Of Thread Count

  The thread count passed to the function that parallelize out code matter as we will in the following graph



Conclusion

Parallelizing the code would help us considerably lower the time and therefore this technology should be used in every domain of the IT specter

For the Resources they would available within the following GitHub Directory :

https://github.com/Nizar04/WEBSCRAPING_PYTHON

THANK YOU