***A Project Report***

*on*

**Parallelization Of Python Web Scraper**

*carried out as part of the* ***Parallel Programming Lab*** *Submitted*

by

**Sughosh Deshpande**

**(209309046)**

**(Section- A)**

*in partial fulfilment for the award of the degree* *of*

**Bachelor of Technology**

in

**Data Science and Engineering**

Under the Guidance of

**Guide Name**



**School of Computing and Information Technology**

**Department of Data Science and Engineering**

**MANIPAL UNIVERSITY JAIPUR**

**JAIPUR-303007**

**RAJASTHAN, INDIA**

**April 2023**

**TABLE OF CONTENTS**

1. Problem Statement.…………………………………………………….3
2. Algorithm …………………...…………………………...………..........4
   1. Serial Web Scraper ………….………………………………...4
   2. Parallelization of web scraper ……………….……………...…5
3. Time Comparison…………………………………………...………….7

**PROBLEM STATEMENT**

Web scraping is a computer software technique of extracting information from websites. This technique mostly focuses on the transformation of unstructured data (HTML format) on the web into structured data. It means extracting data from a website. A Python-based web scraper can efficiently and quickly retrieve data from multiple websites / links simultaneously multiple lines of data from a single website, webpage, database, or dataset. Since web scraping is a computationally intensive process, it is essential to leverage the power of multiprocessing to parallelize the scraping process and speed up the data retrieval process. This is very useful while working on big data as dataset being enormous will take a long time to be parsed thus halting the entire analysis process. A multiprocessor will fairly reduce the time taken to parse the data sometimes to mere seconds. When we are working with just one URL, then Multiprocessing may not be very helpful, but one can observe significant improvement in performance when multiple URLs are scraped to get the data.

The main objective of the project is to design a scraper that takes -

(1) a list of data from a single Wikipedia webpage as input

(2) a list of data (over 1000 names of books) from a text file as input,

and then spawns multiple processes, each responsible for scraping single data concurrently. The scraper is built for text-based data.

The serial web scraper is built using Beautiful Soup which is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping. The scraper is also built using the requests module that allows to send HTTP requests using Python to a website. The HTTP request returns a Response Object with all the response data be it content, encoding, status or anything else. requests and Beautiful Soup make extracting the URLs easy.

The parallel web scraper along with modules of the serial scraper also uses the multiprocessor and concurrent.futures module to implement parallelization to the serial web scraper. multiprocessing is a package that supports spawning processes using an API similar to the threading module. The multiprocessing package offers both **local and remote concurrency,** effectively side-stepping the Global Interpreter Lock by using sub-processes instead of threads. Due to this, the multiprocessing module allows the programmer to fully **leverage multiple processors** on a given machine.

**ALGORITHM**

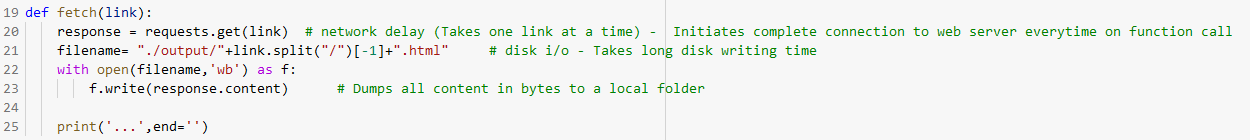
**Serial Web Scraper**

To import the required libraries, and defining a function to download the HTML from a single URL(to get the webpage into a local disk and parse the data into a list.)

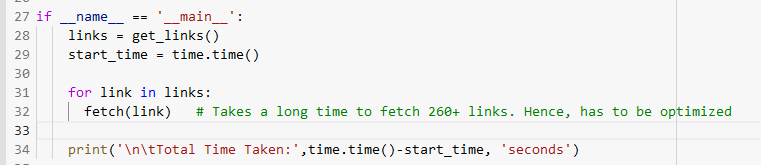
Text

Description automatically generated with low confidence

Function to extract data from all the pages.

****

Function to get all the links and print the time taken to parse.



**Parallelization of the web scraper**

To find out the number of CPUs so that each process will run on a different CPU.

Graphical user interface, application

Description automatically generated

To import the required libraries and function to parse all the links into a list.



When we must process a lot of URLs from the same server or a lot of pages from the same website, we use sessions instead of requests.

Text

Description automatically generated

Function to create a pool of resources, map them to the list of links instead of using a for loop and print the processing time.

Text

Description automatically generated

Making the GET request and receiving the response is pretty concerning if I need to make thousands of requests. Multiprocessing is still a bottleneck, as I only have two physical cores on my machine. Scraping thousands of files will still take thousands of seconds. The benefits of multiprocessing are basically capped by the number of cores in the machine, and multiple Python processes come with more overhead than simply using multiple threads. In Python, I/O functionality releases the Global Interpreter Lock (GIL). This means I/O tasks can be executed concurrently across multiple threads **in the same process,** and that these tasks can happen while other Python bytecode is being interpreted. We can release the GIL in our own library code, too. This is how data science libraries like cuDF and CuPy can be so fast. We can wrap Python code around blazing fast CUDA code (to take advantage of the GPU) that isn’t bound by the GIL. Multithreading with concurrent.futures gives a significant boost here. The ThreadPoolExecutor from concurrent.futures has been used to execute the function across many independent threads.

Text

Description automatically generated

**TIME COMPARISON**

For Serial Web Scrapper:

Total Time Taken: 2.6226043701171875e-06 seconds.

For Web Scrapper Using Multiprocessing:

Total Time Taken: 0.039621829986572266 seconds.

For Web Scrapper Using Multithreading:

Total Time Taken: 9.131431579589844e-05 seconds.

Clearly time taken after parallel processing is much faster and on using multiple threads it becomes even more fast. Now, the time difference grows very quickly when number of URLs increase which means that performance of multiprocessing script improves with large number of URLs.