Parallelizing Web Crawler using Multithreading

J-COMPONENT REPORT

Submitted by

Sayak Maji (18BCE0261)

Bhavesh Sahu (18BCE0789)

Anurag Kashyap (18BCE0776)

Sapan Kumar Shrivastava (18BCE0779)

Course Code: CSE4001

Course Title: Parallel and Distributed Computing

Under the guidance of

Dr. S. Anto

Associate Professor, SCOPE, VIT

Vellore.

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

October, 2020

INDEX

1. ABSTRACT

A parallel crawler is a crawler that runs multiple processes in parallel. The goal is to maximize the download rate while minimizing the overhead from parallelization and to avoid repeated downloads of the same page. The number of Internet pages is extremely large; even the largest crawlers fall short of making a complete index. The number of possible URLs crawled being generated by server side software has also made it difficult for web crawlers to avoid retrieving duplicate content. Endless combinations of HTTP GET (URL-based) parameters exist, of which only a small selection will actually return unique content. Crawlers usually perform some type of URL normalization in order to avoid crawling the same resource more than once. The term URL normalization, also called URL canonicalization, refers to the process of modifying and standardizing a URL in a consistent manner. There are several types of normalization that may be performed including conversion of URLs to lowercase, removal of "." and ".." segments, and adding trailing slashes to the non-empty path component. Through this project, we will demonstrate web crawling of multiple child node in one go from the parent node using the concept of multithreading in python program. In this project we perform a real life application of multithreaded web crawler where we scrape and display stock information about various companies.

2. INTRODUCTION

2.1 Problem Definition:

The things such as web scraping and the web crawling process is done using python language, but the problem arises as python is by default a single threaded and an interpreter based language, which ultimately slows down the site using python Crawler. Also the number of the internet pages is extremely large; even the largest Crawler fall short of making a complete index.

2.2 Existing solution:

Serial web crawler:- Here a single seed URL is taken as input and pushed into the frontier queue. Then that URL is parsed for further URLs and same process is repeated for those URLs. To do this either BFS or DFS is used on the frontier URLs.

Distributed web crawler:- A single frontier queue is used and URLs are filled in frontier queue via multiple crawlers in parallel.

2.3 The issues faced by the existing solution: The above distributed web crawler does not have proper mechanism to detect duplicates.

3. LITERATURE SURVEY

Web crawlers follow links in webpages and automatically download the page. Crawling is the most basic as well as the most powerful web search procedure. A design and framework of a multithreaded web crawler has been proposed. The end result of the process is a collection of web pages. The experiment demonstrates how this process has better performance. It is possible to increase the performance of a search by understanding the user's need and the relevance of the document. The computer network is vast and act as a single huge network for data and message transportation across vast distances. The World wide web has millions of pages linked and is ever growing. Hence, they continuously keep track of the web and find new webpages. This system called the semantic web works on shared meaningful knowledge representation. It proposes to make an AI application that will make the content of the web meaningful to the system. This will make the crawler must more efficient by crawling only those pages which has relevance to the users requirements. [1]

Initially a seed URL is given input by the user. Crawler then fetches the webpage from the interne. Contents of the page are parsed and the URL are scraped and stored in Database. If the links are to be traversed, the links are clicked and passed onto the web crawlers. Crawler then searches the keyword if present in the page fetched using the URL. If present, it calculates the number of times it is present in the page. A brief evaluation study of the crawler in an uncontrolled and practical environment is the main objective of this research paper. The World Wide Web (or simply the web) is a vast, prosperous, superior, readily available and suitable source of information and its users are growing very fast nowadays. To extract information from the web, Search engines are used that access web pages as per the requirement of the users. Most of the data in the web is unmanaged so it is not possible to use the entire web at once in a single attempt, so search engines use the web crawler. Web crawler is an important part of search engine. This is a program that navigates the web and downloads reference to web pages. [2]

Search engines run multiple instances of crawlers on wide spread servers Get various information from them. Web crawler crawls from page to page in world wide. Get the webpage, load the page content and index it to find the engines database. This paper presents a systematic study of the web Crawler. The study of web crawler is very important because properly designed web crawler always gives good yield. Web crawler are an efficient component of the search engine. It is the core member responsible for searching the web and indexing the traversed webpages. It starts with a few seed pages, travels the web using the links in the webpages and inserts new link in the system database. The web pages found are hashed for easy lookup later. [3]

In described paper research shows the experimentation to harvest bibliography data using multi-threading in a fast safe and ethical manner. It uses a focused web crawler to download particular data from certain sites. One of the challenges in searching/query is deciding upon which criteria to use so that we can get the maximum number of bibliography data. Mainly the query is done using OCLC number to receive the maximum amount of bibliography by performing the process recurrently. One more important point was to keep in mind the netiquette convention of Prohibition to flood websites with requests that hindered their performance to serve other users .Since the algorithm has only one loop therefore the complexity is O(n) which is pretty less. It was able to create a focused web harvesting program on bibliography data from the WorldCat database. This led to downloading over five million bibliography data without getting our IP blocked. [4]

Crawler is a program that can auto collect information from internet. This Crawler has features of high accuracy, strong adaptability and high efficiency. Specific resource crawler can be divided into four modules: (1) The module of webpage downloading. The main work of this module is to download the page. (2) The module of URL filtering. This module determines which URL should be crawled by analyzing the URL of the webpage, and then adds the URL to the URL queue. (3) The module of webpage analysis. This module analyzes the webpage information through information extracting. (4) The module of data persistence. This module will transform the data model in the memory to storage model. Through the use of sharing queue model to communication, we make data structure operated by multiple threads existing in this cohort. This makes communication easy to control. It is better than the thread synchronization distribution everywhere. Specific resources of crawlers are based on multi-thread identity, mainly through setting the priority of a thread and thread dormant time. A Crawler's system evaluation depends on two aspects: crawler's accuracy rate and system crawler's rate. [5]

Another research paper has tried to propose a different web community crawler since many of the crawlers are not good at gathering web document due to the URLs of web articles are frequently changed and redundant, which will make the crawling job difficult and Second, the amount of articles is significantly large that the crawler should be designed in a scalable manner. The paper has tried to provide a crawler much different from the apache Nutch crawler as , it is not suitable for collecting information from online communities because it gathers duplicated or meaningless webpages. The design and the implementation of distributed web crawler is done using a master and multiple workers model. A master node monitors and manages workers and their tasks. Workers consists of three modules Login Module, HTTP Client Module and Main Module, and they run independently without any coordination among other crawlers. Crawled web pages are stored in the local storage. The implementation of the distributed community crawler comprises three modules:

• The login module is responsible for logging into online communities when a valid account is needed before crawling starts.

• The HTTP client module sends HTTP request messages and processes HTTP response messages.

• The crawler core module manages whole process of the distributed community crawler.

Paper's implementation and Nutch web crawler launch 18 crawling processes concurrently. The maximum number of threads per each task for Nutch is set to 20. For instance, the number of threads in one node is 80, which executes four crawling processes. On the other hand, in Paper's implementation, one crawling process has one thread. Finally the result has shown in the paper that Research paper implementation has higher throughput and higher accuracy. [6]

As information on internet is growing at an explosive rate , most of the companies are shifting towards distributed web crawling from standalone web crawling process. The Research paper has categorized the distributed system architecture in 4 layers called Infrastructure, platform layer , Business logic layer and UI layer. The infrastructure layer contains the cluster of servers for physical infrastructure. At the platform level, distributed processing architecture based on Mesos, Marathon and Zookeeper is used to ensure the crawling system to run stably and efficiently. The logic layer is the core of the whole system, which is divided into two sub-systems, namely, distributed crawling management system and crawlers supporting distribution. At the top level of the entire architecture is the UI display layer, which provides a user interface or connection.

According to paper the distributing system consist of two modules one is monitor and other one is Daemon . the monitor module provides the Web interface and Daemon process is responsible for task scheduling, deployment and management of task containers.
The Comparison is done between the standalone and the Light weight crawler in the paper on the basis of pages crawled per unit time and on the utilization ratio of hosts CPU's , memory and other aspects. By Observation we got to know that , better crawling efficiency is achieved for getting web pages in cluster rather than in single computer's environment. In addition to that ,the light weight distributed crawler has lower CPU utilization and less memory. [7].

In eighth paper, the authors present client server architecture based smart distributed crawler for crawling web. In this architecture load between the crawlers is managed by server and each time a crawler is loaded, load is distributed to others by dynamically distributing the URLs. The Paper introduces some web crawlers like Mercator , which is an extensible WebCrawler written in java, In this content seen test is used to avoid adding duplicate URLs. In addition to that paper introduces Igloog WebCrawler, which   download web pages on large scale where distribution is provided by grid service. Similarly DCrawler , IBM web fountain and many more distributed web crawler has been introduced in it.

Since smart web crawler is distributed so best first and breadth first search algorithms are considered to be practical for it. For assigning new URL there are two types of strategies given . In static assignment strategy assignment of URLs is fixed from start. In dynamic assignment server dynamically assigns new URLs to the crawlers in the system. In this way server can balance the load of crawlers.

For dataset that are large in size data structure such as a b-tree can be used. B-Trees can help in insertion, update and removal of items. For extracting hyperlinks from document either HTML parser or regular expressions can be used. Since Size of the web is growing and from user point of view, user expects the crawler to retrieve the results as soon as possible, it turns out to be profoundly attractive to utilize distributed web crawlers when search engine fails to do so. [8]

There are cases when multiple threads may crawl the same URL. In such cases the crawler becomes more vulnerable to falling in an infinite loop trap. That is why duplicate detector forms an important part of web crawler. Traditionally brute force approach can be used to detect duplicates, i.e. scraped urls can be stored in a list and every url to crawl can be checked in it via linear search. Another method can be keeping record of timestamps of urls as they are crawled and allow only urls with timestamps less than current timestamps to get scraped. However brute force leads to high time complexity whereas storing timestamps may lead to inaccurate results if system's time is wrong. A better approach is using bloom filters for storing hash of urls and accessing them in O(1) time and checking them with urls to crawl. The only disadvantage of using bloom filters is that it gives some False Positive results during duplicate detection, i.e some uncrawled urls are often declared as duplicates.[9]

In the next paper, the authors put forth some ways to reduce the false positive results of traditional bloom filters. The urls are first preprocessed by MD5 to solve hash conflict caused by different URL lengths. Also multidimensional array is used to optimize classic bloom filter to reduce error rate. [10]

4. OVERVIEW OF WORK

4.1 Objectives of the Project:

A serial crawler parses a given URL and stores the URLs present in the frontier queue. For each of the URLs present in the queue we perform the above operation. The chief problem here is lack of speed. To solve this problem we use multiple threads to perform the task of crawling the stock data. The main objective of our project is to compare the performance of a serial crawler with multithreaded crawler.

4.2 Software Requirements:

VSCode as text-editor, Python libraries BeautifulSoup for crawling, Requests, URLlib

4.3 Hardware Requirements:

8 GB RAM, x64 processor

4.4 Solution Proposed:

A seed URL is taken as input. Initially the URL is pushed in the frontier queue. The URL is parsed for more URLs. Multithreading is done for fetching of the URLs. In this process, the URLs fetched by each thread is checked over a set of pages already scraped, if not scraped before its added to the set, else the page is discarded. It is made to work like an ackerman function where key methods of crawler are packed inside a function that is called for request

4.5 Scenario implemented:

Two operations are performed, first stock prices of different companies is scraped from respective sites directed from main URL. Secondly news headlines URLs are crawled from the main URL. E.g. https://in.finance.yahoo.com/topic/latestnews is passed as input and all URLs present in the page is crawled and each of those URLs are further parsed for more URLs. For stock prices, a text file is created containing symbols for companies. Using those symbols our crawler is directed to respected pages of companies and required data is scraped.

5. SYSTEM DESIGN

5.1 Algorithms:

Web Scraping:

```
Open txt file & convert elements to List
Create List data structure for storing threads
For each companies in List:
    Create thread & call Scraping Price Function to Scrape Price from web page
    Start thread & add to ThreadList

SCRAPE PRICE FUNCTION( ):
    Get text html of url Web Page
    Use BeautifulSoup & find Stock Price of company from Web Page
    Print Stock price
```

Using 20 workers to scrape the pages.

Web Crawling:

```
Initializing Global Queue
Initializing Global Set
Initializing root URL
Create pool of threads using ThreadPoolExecutor
Using bloom filter to store scraped urls and fast duplicate detection
While True:
    url = pop from Queue
    if url not in Filter:
        print url
        submit the  Scraping function in ThreadPoolExecutor to Scrape URL
        from web page for multiple threads to call the function.
        Add url to filter
        Sleep for 0.05 seconds
```

```
SCRAPING FUNCTION( URL ):
    Get text html of url Web Page
    Use BeautifulSoup & find all links on Web Page
    For each link
        If its similar to root url:
            If its not in Set:
                Add link to queue
```

Using 20 workers to execute crawl operation.

5.2 Block Diagram:

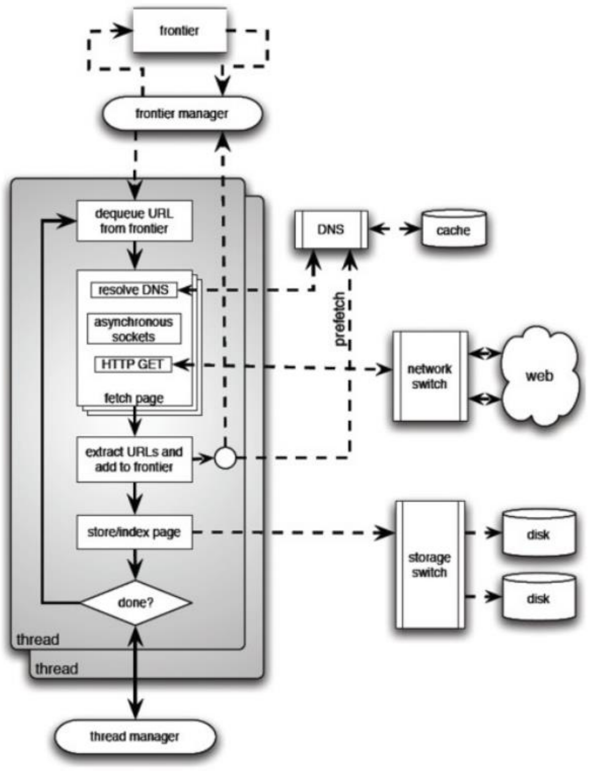


Fig 1. 1 Serial scalable web crawler

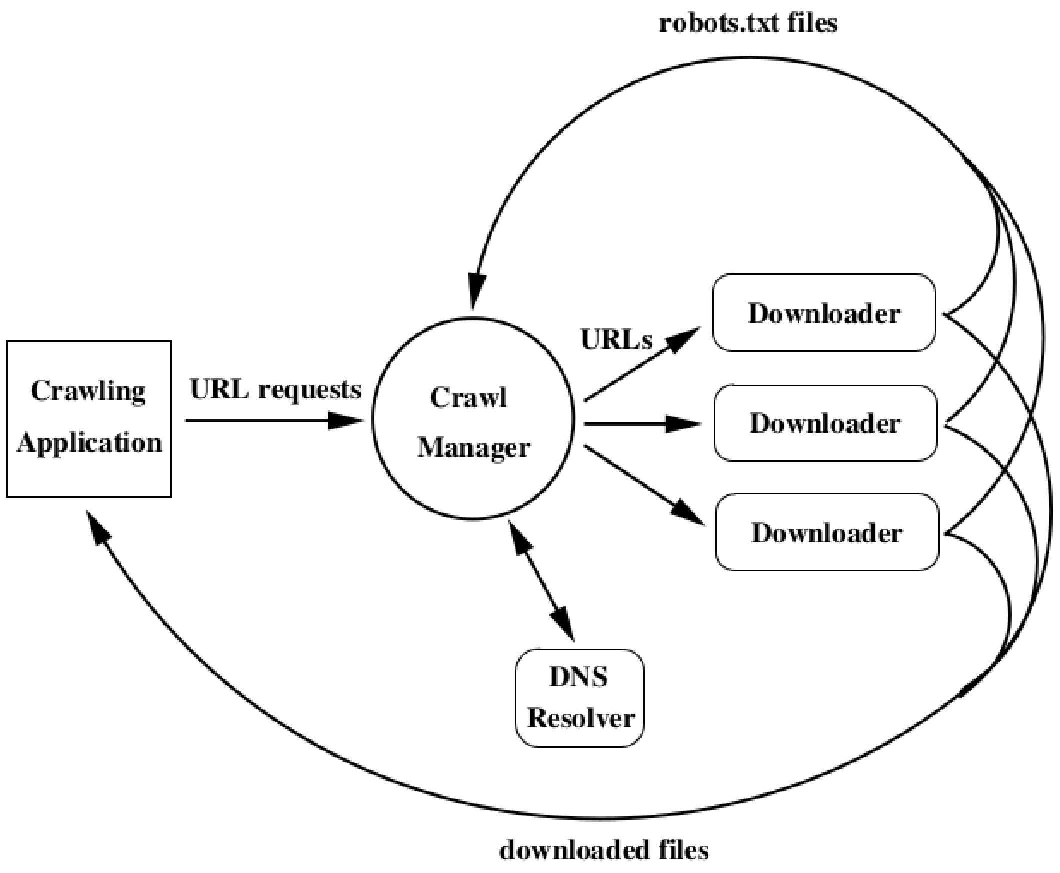
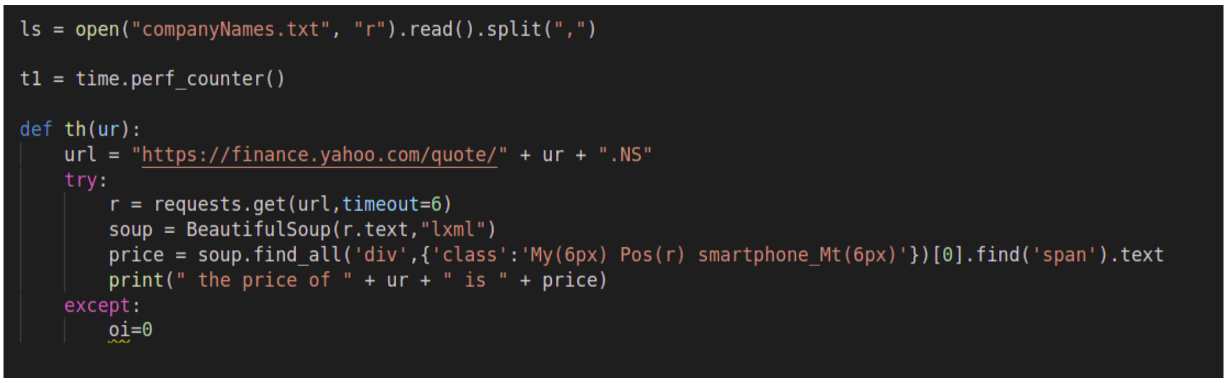


Fig 1.2 Multithreaded Web Crawler

6. IMPLEMENTATION

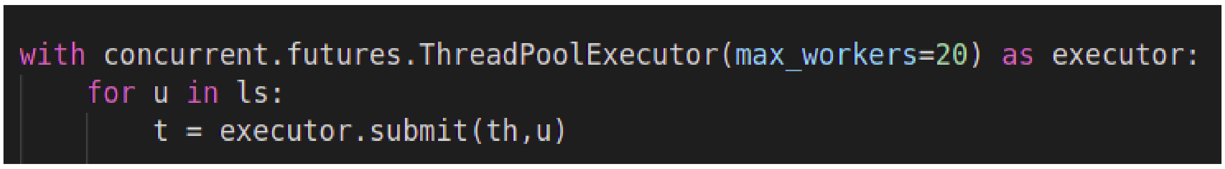
6.1 Description of Modules/Programs :

The symbols of the companies are given in a text file. Each symbol is added to URL for scraping stock price of that particular company. Function th is created to scrape prices from any url.



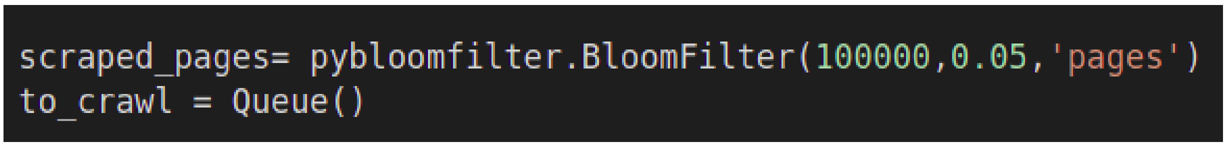
```
ls = open("companyNames.txt", "r").read().split(",")

t1 = time.perf_counter()

def th(ur):
    url = "https://finance.yahoo.com/quote/" + ur + ".NS"
    try:
        r = requests.get(url,timeout=6)
        soup = BeautifulSoup(r.text,"lxml")
        price = soup.find_all('div',{'class':'My(6px) Pos(r) smartphone_Mt(6px)'})[0].find('span').text
        print(" the price of " + ur + " is " + price)
    except:
        oi=0
```

Multiple threads call this function with different parameters ur for different companies.

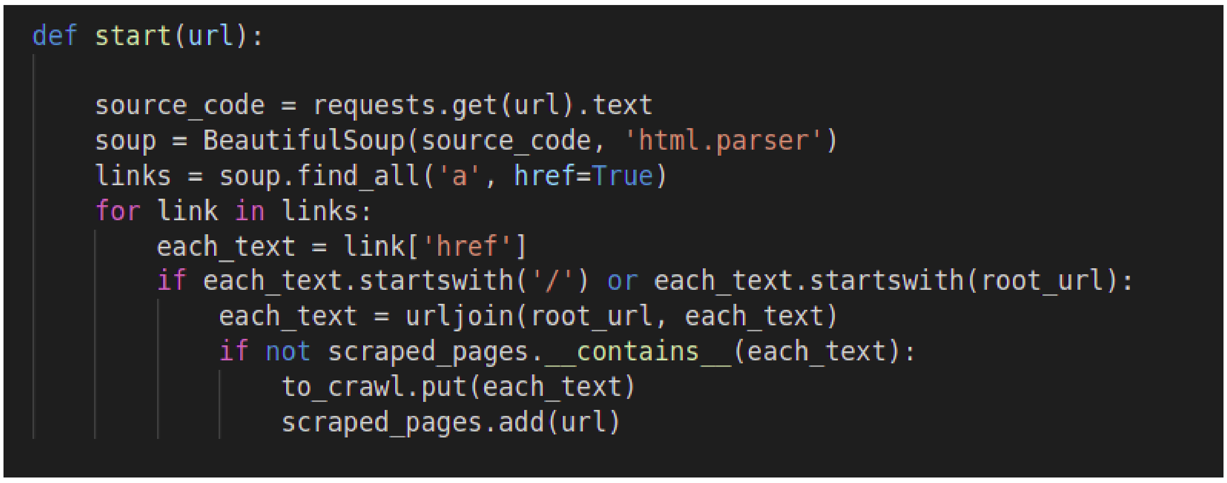


```
with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
    for u in ls:
        t = executor.submit(th,u)
```

Below is the function for crawling all the news urls from a seed url. Scraped pages are stored in a bloom filter for searching them repeatedly and detecting duplicates.

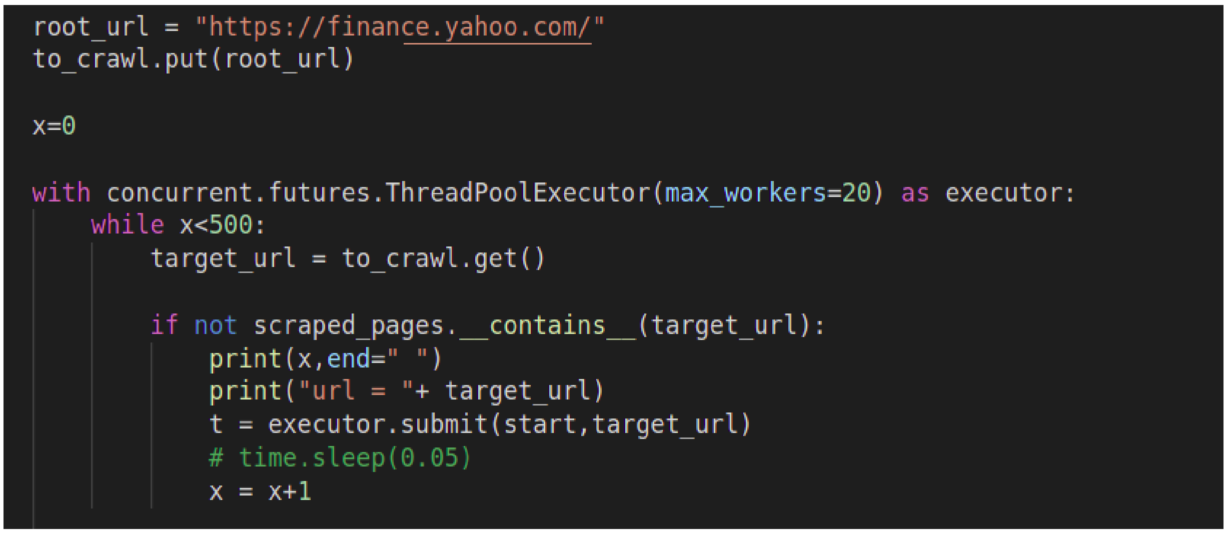


```
scraped_pages= pybloomfilter.BloomFilter(100000,0.05,'pages')
to_crawl = Queue()
```



```
def start(url):

    source_code = requests.get(url).text
    soup = BeautifulSoup(source_code, 'html.parser')
    links = soup.find_all('a', href=True)
    for link in links:
        each_text = link['href']
        if each_text.startswith('/') or each_text.startswith(root_url):
            each_text = urljoin(root_url, each_text)
            if not scraped_pages.__contains__(each_text):
                to_crawl.put(each_text)
                scraped_pages.add(url)
```

Below, 20 workers are used to scrape urls in root_url page by calling start function simultaneously.



```python
root_url = "https://finance.yahoo.com/"
to_crawl.put(root_url)

x=0

with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
    while x<500:
        target_url = to_crawl.get()

        if not scraped_pages.__contains__(target_url):
            print(x,end=" ")
            print("url = "+ target_url)
            t = executor.submit(start,target_url)
            # time.sleep(0.05)
            x = x+1
```

Modules Used –

1. **Concurrent futures -** Concurrent.futures module provides a high-level interface for asynchronously executing callables.
2. **ThreadPoolExecutor** – It is an Executor subclass that uses a pool of threads to execute calls asynchronously.
3. **Time** - Return the value (in fractional seconds) of a performance counter.
4. **Beautiful Soup -** Beautiful Soup 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
5. **Urllib -** urllib is a package that collects several modules for working with URLs: urllib. request for opening and reading URLs
6. **Queue -** A Queue is a linear structure which follows a particular order in which the operations are performed.
7. **Bloom Filter -** Bloom filter is a space-efficient probabilistic data structure. We used it to store already parsed URLs so that we will not parse that URL again.
8. **Request** – It used is to make HTTP requests simpler and more human-friendly.

6.2 Source Code :

```
import concurrent.futures

import time

import requests
```

```
from bs4 import BeautifulSoup
from urllib.parse import urljoin
from queue import Queue
import pybloomfilter

ls = open("companyNames.txt", "r").read().split(",")

t1 = time.perf_counter()

def th(ur):
    url = "https://finance.yahoo.com/quote/" + ur + ".NS"
    try:
        r = requests.get(url,timeout=6)
        soup = BeautifulSoup(r.text,"lxml")
        price = soup.find_all('div',{'class':'My(6px) Pos(r)
smartphone_Mt(6px)'})[0].find('span').text
        print(" the price of " + ur + " is " + price)
    except:
        oi=0



with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
    for u in ls:
        t = executor.submit(th,u)


        time.sleep(0.05)

t2 = time.perf_counter()
print("*****-----******")
print(f'Finished in {t2-t1} seconds')
print("*****-----******")
```

```
t1 = time.perf_counter()

scraped_pages= pybloomfilter.BloomFilter(100000,0.05,'pages')
to_crawl = Queue()

def start(url):

    source_code = requests.get(url).text
    soup = BeautifulSoup(source_code, 'html.parser')
    links = soup.find_all('a', href=True)
    for link in links:
        each_text = link['href']
        if each_text.startswith('/') or each_text.startswith(root_url):
            each_text = urljoin(root_url, each_text)
            if not scraped_pages.__contains__(each_text):
                to_crawl.put(each_text)
                scraped_pages.add(url)



root_url = "https://finance.yahoo.com/"
to_crawl.put(root_url)

x=0

with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
    while x<500:
        target_url = to_crawl.get()

        if not scraped_pages.__contains__(target_url):
            print(x,end=" ")
```

```
          print("url = "+ target_url)
          t = executor.submit(start,target_url)
          # time.sleep(0.05)
          x = x+1

     t2 = time.perf_counter()
     print("*****-----******")
     print(f'Finished in {t2-t1} seconds')
```

6.3 Test cases :

URL to scrape stock prices from.

URL to crawl for news headlines.

E.g.:-

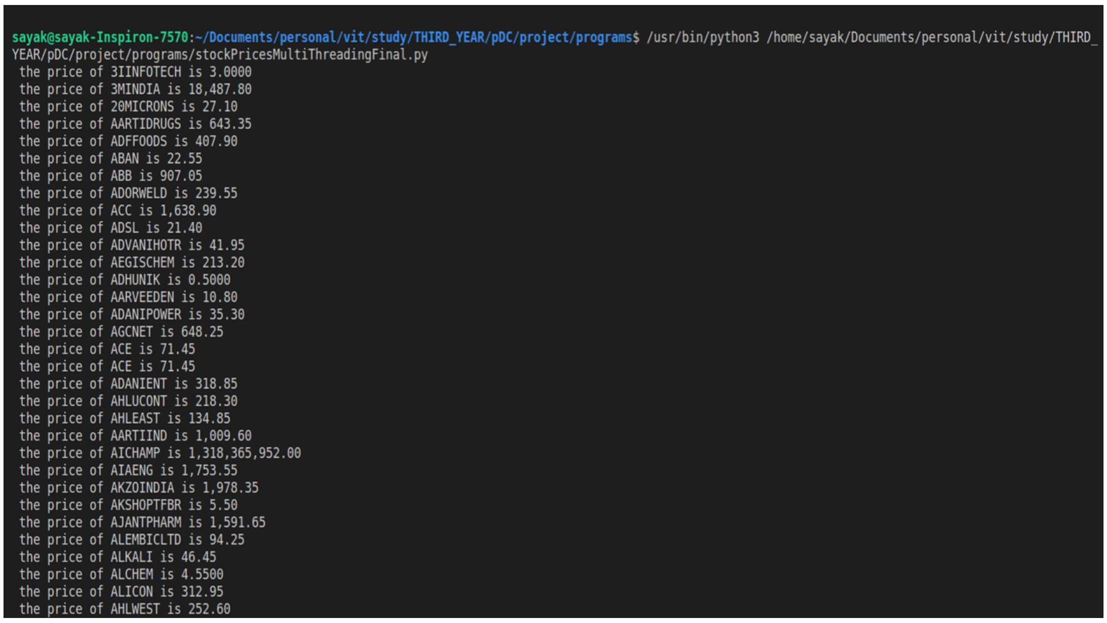
https://finance.yahoo.com/quote/ for scraping stock prices.

https://finance.yahoo.com/ for crawling news articles.

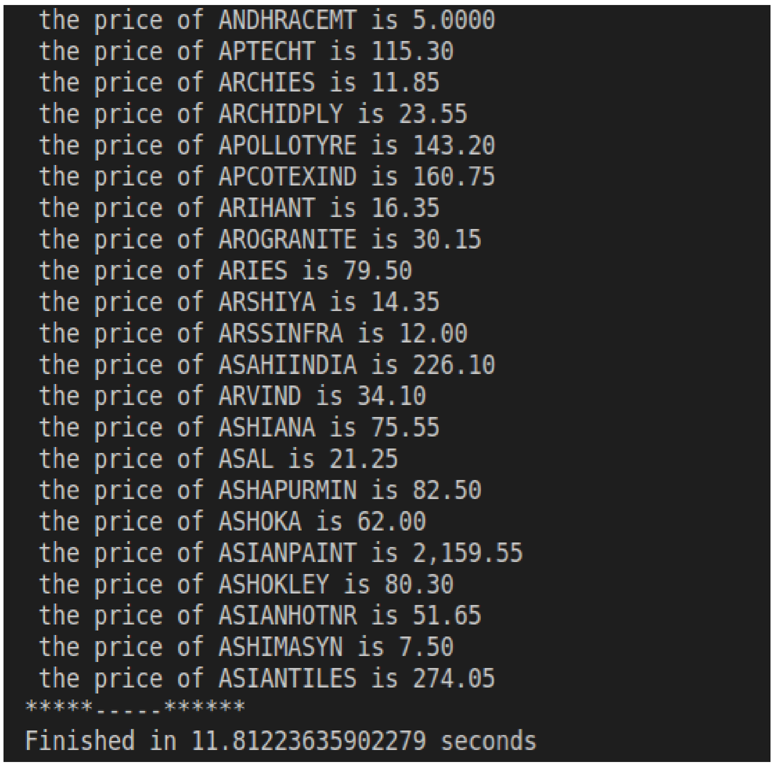
7. Output and Performance Analysis

7.1 Execution snapshots :

Scraping stock prices:

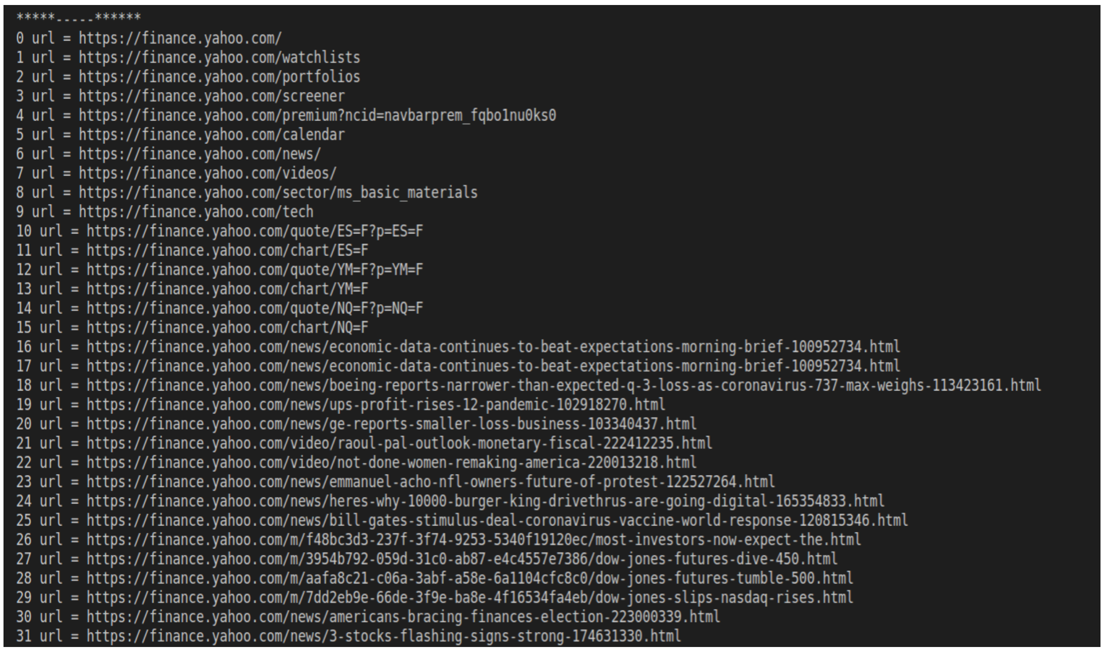


```
sayak@sayak-Inspiron-7570:~/Documents/personal/vit/study/THIRD_YEAR/pDC/project/programs$ /usr/bin/python3 /home/sayak/Documents/personal/vit/study/THIRD_
YEAR/pDC/project/programs/stockPricesMultiThreadingFinal.py
 the price of 3IINFOTECH is 3.0000
 the price of 3MINDIA is 18,487.80
 the price of 20MICRONS is 27.10
 the price of AARTIDRUGS is 643.35
 the price of ADFFOODS is 407.90
 the price of ABAN is 22.55
 the price of ABB is 907.05
 the price of ADORWELD is 239.55
 the price of ACC is 1,638.90
 the price of ADSL is 21.40
 the price of ADVANIHOTR is 41.95
 the price of AEGISCHEM is 213.20
 the price of ADHUNIK is 0.5000
 the price of AARVEEDEN is 10.80
 the price of ADANIPOWER is 35.30
 the price of AGCNET is 648.25
 the price of ACE is 71.45
 the price of ACE is 71.45
 the price of ADANIENT is 318.85
 the price of AHLUCONT is 218.30
 the price of AHLEAST is 134.85
 the price of AARTIIND is 1,009.60
 the price of AICHAMP is 1,318,365,952.00
 the price of AIAENG is 1,753.55
 the price of AKZOINDIA is 1,978.35
 the price of AKSHOPTFBR is 5.50
 the price of AJANTPHARM is 1,591.65
 the price of ALEMBICLTD is 94.25
 the price of ALKALI is 46.45
 the price of ALCHEM is 4.5500
 the price of ALICON is 312.95
 the price of AHLWEST is 252.60
```

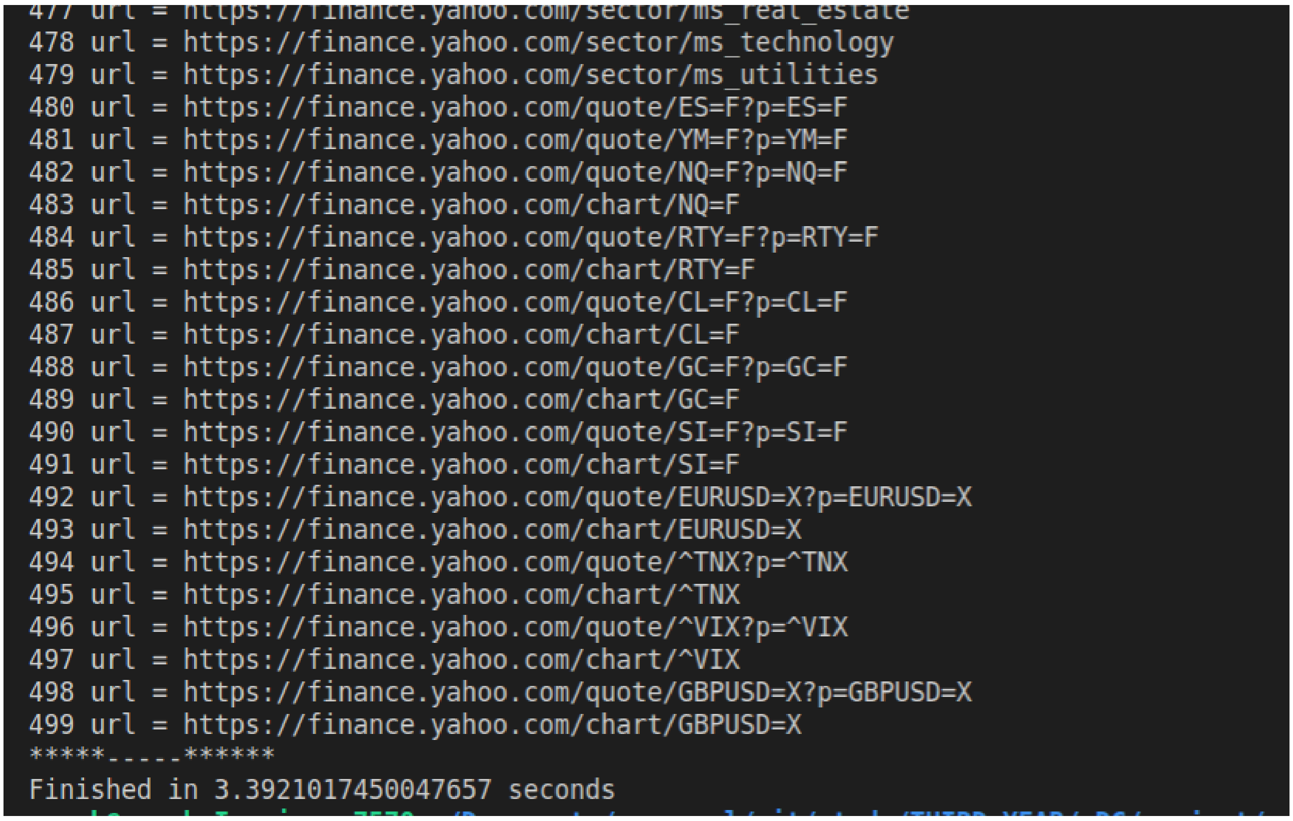


```
  the price of ANDHRACEMT is 5.0000
  the price of APTECHT is 115.30
  the price of ARCHIES is 11.85
  the price of ARCHIDPLY is 23.55
  the price of APOLLOTYRE is 143.20
  the price of APCOTEXIND is 160.75
  the price of ARIHANT is 16.35
  the price of AROGRANITE is 30.15
  the price of ARIES is 79.50
  the price of ARSHIYA is 14.35
  the price of ARSSINFRA is 12.00
  the price of ASAHIINDIA is 226.10
  the price of ARVIND is 34.10
  the price of ASHIANA is 75.55
  the price of ASAL is 21.25
  the price of ASHAPURMIN is 82.50
  the price of ASHOKA is 62.00
  the price of ASIANPAINT is 2,159.55
  the price of ASHOKLEY is 80.30
  the price of ASIANHOTNR is 51.65
  the price of ASHIMASYN is 7.50
  the price of ASIANTILES is 274.05
*****-----******
Finished in 11.81223635902279 seconds
```

Crawling yahoo finance for URLs:



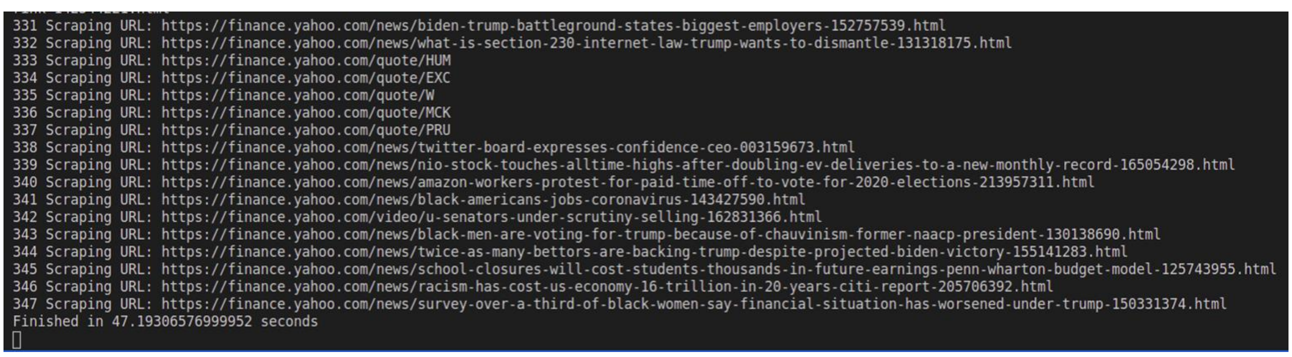
```
*****-----******
0 url = https://finance.yahoo.com/
1 url = https://finance.yahoo.com/watchlists
2 url = https://finance.yahoo.com/portfolios
3 url = https://finance.yahoo.com/screener
4 url = https://finance.yahoo.com/premium?ncid=navbarprem_fqbo1nu0ks0
5 url = https://finance.yahoo.com/calendar
6 url = https://finance.yahoo.com/news/
7 url = https://finance.yahoo.com/videos/
8 url = https://finance.yahoo.com/sector/ms_basic_materials
9 url = https://finance.yahoo.com/tech
10 url = https://finance.yahoo.com/quote/ES=F?p=ES=F
11 url = https://finance.yahoo.com/chart/ES=F
12 url = https://finance.yahoo.com/quote/YM=F?p=YM=F
13 url = https://finance.yahoo.com/chart/YM=F
14 url = https://finance.yahoo.com/quote/NQ=F?p=NQ=F
15 url = https://finance.yahoo.com/chart/NQ=F
16 url = https://finance.yahoo.com/news/economic-data-continues-to-beat-expectations-morning-brief-100952734.html
17 url = https://finance.yahoo.com/news/economic-data-continues-to-beat-expectations-morning-brief-100952734.html
18 url = https://finance.yahoo.com/news/boeing-reports-narrower-than-expected-q-3-loss-as-coronavirus-737-max-weighs-113423161.html
19 url = https://finance.yahoo.com/news/ups-profit-rises-12-pandemic-102918270.html
20 url = https://finance.yahoo.com/news/ge-reports-smaller-loss-business-103340437.html
21 url = https://finance.yahoo.com/video/raoul-pal-outlook-monetary-fiscal-222412235.html
22 url = https://finance.yahoo.com/video/not-done-women-remaking-america-220013218.html
23 url = https://finance.yahoo.com/news/emmanuel-acho-nfl-owners-future-of-protest-122527264.html
24 url = https://finance.yahoo.com/news/heres-why-10000-burger-king-drivethrus-are-going-digital-165354833.html
25 url = https://finance.yahoo.com/news/bill-gates-stimulus-deal-coronavirus-vaccine-world-response-120815346.html
26 url = https://finance.yahoo.com/m/f48bc3d3-237f-3f74-9253-5340f19120ec/most-investors-now-expect-the.html
27 url = https://finance.yahoo.com/m/3954b792-059d-31c0-ab87-e4c4557e7386/dow-jones-futures-dive-450.html
28 url = https://finance.yahoo.com/m/aafa8c21-c06a-3abf-a58e-6a1104cfc8c0/dow-jones-futures-tumble-500.html
29 url = https://finance.yahoo.com/m/7dd2eb9e-66de-3f9e-ba8e-4f16534fa4eb/dow-jones-slips-nasdaq-rises.html
30 url = https://finance.yahoo.com/news/americans-bracing-finances-election-223000339.html
31 url = https://finance.yahoo.com/news/3-stocks-flashing-signs-strong-174631330.html
```



```
477 url = https://finance.yahoo.com/sector/ms_real_estate
478 url = https://finance.yahoo.com/sector/ms_technology
479 url = https://finance.yahoo.com/sector/ms_utilities
480 url = https://finance.yahoo.com/quote/ES=F?p=ES=F
481 url = https://finance.yahoo.com/quote/YM=F?p=YM=F
482 url = https://finance.yahoo.com/quote/NQ=F?p=NQ=F
483 url = https://finance.yahoo.com/chart/NQ=F
484 url = https://finance.yahoo.com/quote/RTY=F?p=RTY=F
485 url = https://finance.yahoo.com/chart/RTY=F
486 url = https://finance.yahoo.com/quote/CL=F?p=CL=F
487 url = https://finance.yahoo.com/chart/CL=F
488 url = https://finance.yahoo.com/quote/GC=F?p=GC=F
489 url = https://finance.yahoo.com/chart/GC=F
490 url = https://finance.yahoo.com/quote/SI=F?p=SI=F
491 url = https://finance.yahoo.com/chart/SI=F
492 url = https://finance.yahoo.com/quote/EURUSD=X?p=EURUSD=X
493 url = https://finance.yahoo.com/chart/EURUSD=X
494 url = https://finance.yahoo.com/quote/^TNX?p=^TNX
495 url = https://finance.yahoo.com/chart/^TNX
496 url = https://finance.yahoo.com/quote/^VIX?p=^VIX
497 url = https://finance.yahoo.com/chart/^VIX
498 url = https://finance.yahoo.com/quote/GBPUSD=X?p=GBPUSD=X
499 url = https://finance.yahoo.com/chart/GBPUSD=X
*****-----******
Finished in 3.3921017450047657 seconds
```

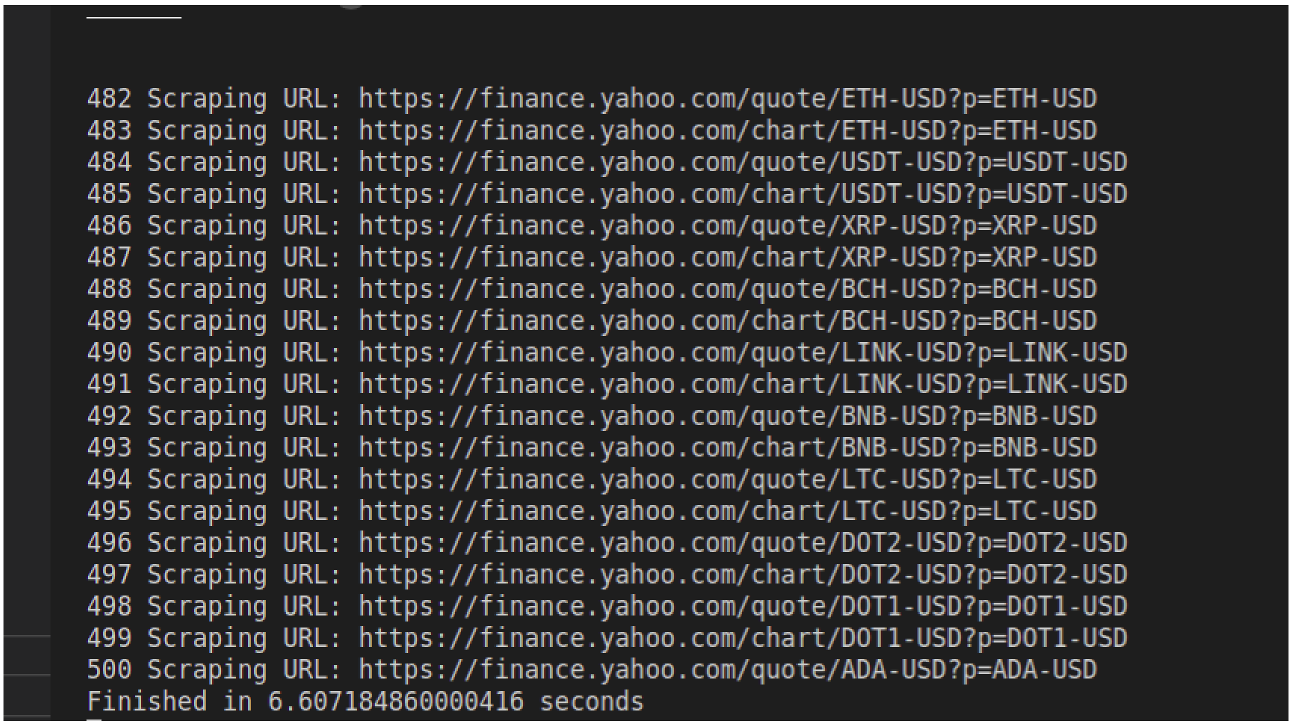
7.2 Output – in terms of performance metrics :

Serial crawler:-



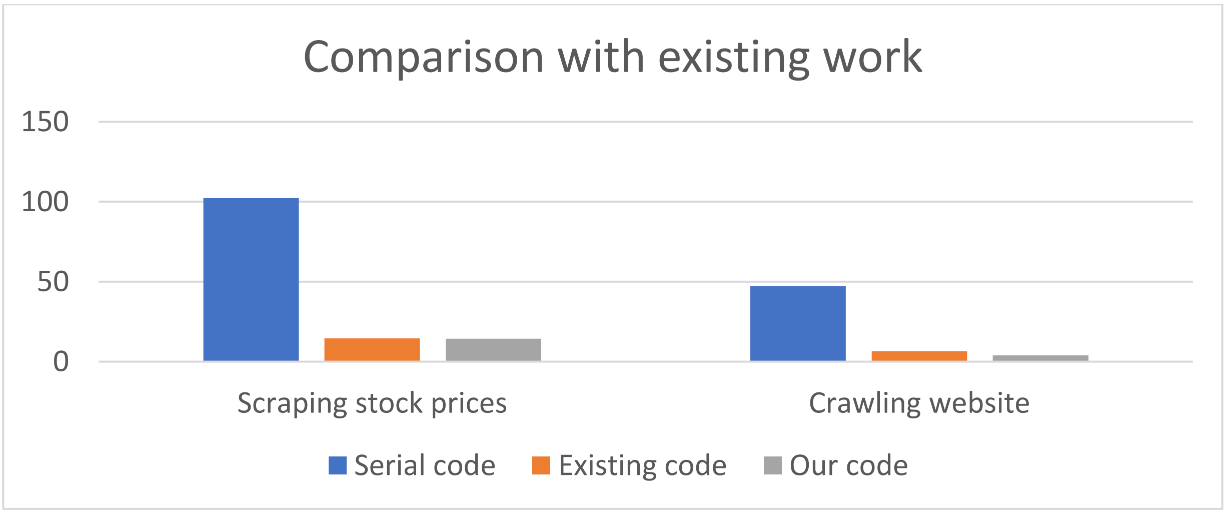
```
331 Scraping URL: https://finance.yahoo.com/news/biden-trump-battleground-states-biggest-employers-152757539.html
332 Scraping URL: https://finance.yahoo.com/news/what-is-section-230-internet-law-trump-wants-to-dismantle-131318175.html
333 Scraping URL: https://finance.yahoo.com/quote/HUM
334 Scraping URL: https://finance.yahoo.com/quote/EXC
335 Scraping URL: https://finance.yahoo.com/quote/W
336 Scraping URL: https://finance.yahoo.com/quote/MCK
337 Scraping URL: https://finance.yahoo.com/quote/PRU
338 Scraping URL: https://finance.yahoo.com/news/twitter-board-expresses-confidence-ceo-003159673.html
339 Scraping URL: https://finance.yahoo.com/news/nio-stock-touches-alltime-highs-after-doubling-ev-deliveries-to-a-new-monthly-record-165054298.html
340 Scraping URL: https://finance.yahoo.com/news/amazon-workers-protest-for-paid-time-off-to-vote-for-2020-elections-213957311.html
341 Scraping URL: https://finance.yahoo.com/news/black-americans-jobs-coronavirus-143427590.html
342 Scraping URL: https://finance.yahoo.com/video/u-senators-under-scrutiny-selling-162831366.html
343 Scraping URL: https://finance.yahoo.com/news/black-men-are-voting-for-trump-because-of-chauvinism-former-naacp-president-130138690.html
344 Scraping URL: https://finance.yahoo.com/news/twice-as-many-bettors-are-backing-trump-despite-projected-biden-victory-155141283.html
345 Scraping URL: https://finance.yahoo.com/news/school-closures-will-cost-students-thousands-in-future-earnings-penn-wharton-budget-model-125743955.html
346 Scraping URL: https://finance.yahoo.com/news/racism-has-cost-us-economy-16-trillion-in-20-years-citi-report-205706392.html
347 Scraping URL: https://finance.yahoo.com/news/survey-over-a-third-of-black-women-say-financial-situation-has-worsened-under-trump-150331374.html
Finished in 47.19306576999952 seconds
```

Existing parallel crawler:-



```
482 Scraping URL: https://finance.yahoo.com/quote/ETH-USD?p=ETH-USD
483 Scraping URL: https://finance.yahoo.com/chart/ETH-USD?p=ETH-USD
484 Scraping URL: https://finance.yahoo.com/quote/USDT-USD?p=USDT-USD
485 Scraping URL: https://finance.yahoo.com/chart/USDT-USD?p=USDT-USD
486 Scraping URL: https://finance.yahoo.com/quote/XRP-USD?p=XRP-USD
487 Scraping URL: https://finance.yahoo.com/chart/XRP-USD?p=XRP-USD
488 Scraping URL: https://finance.yahoo.com/quote/BCH-USD?p=BCH-USD
489 Scraping URL: https://finance.yahoo.com/chart/BCH-USD?p=BCH-USD
490 Scraping URL: https://finance.yahoo.com/quote/LINK-USD?p=LINK-USD
491 Scraping URL: https://finance.yahoo.com/chart/LINK-USD?p=LINK-USD
492 Scraping URL: https://finance.yahoo.com/quote/BNB-USD?p=BNB-USD
493 Scraping URL: https://finance.yahoo.com/chart/BNB-USD?p=BNB-USD
494 Scraping URL: https://finance.yahoo.com/quote/LTC-USD?p=LTC-USD
495 Scraping URL: https://finance.yahoo.com/chart/LTC-USD?p=LTC-USD
496 Scraping URL: https://finance.yahoo.com/quote/DOT2-USD?p=DOT2-USD
497 Scraping URL: https://finance.yahoo.com/chart/DOT2-USD?p=DOT2-USD
498 Scraping URL: https://finance.yahoo.com/quote/DOT1-USD?p=DOT1-USD
499 Scraping URL: https://finance.yahoo.com/chart/DOT1-USD?p=DOT1-USD
500 Scraping URL: https://finance.yahoo.com/quote/ADA-USD?p=ADA-USD
Finished in 6.607184860000416 seconds
```

Comparison between our code and existing work -



7.3 Performance comparison with existing works :

TASK	Serial code	Existing code	Our code
Scraping stock prices	102.23 sec	14.45 sec	14.43 sec
Crawling website	47.19 sec	6.6 sec	3.39 sec

8. CONCLUSION AND FUTURE DIRECTIONS

Our crawling using multithreading is almost ten times faster than serial crawler crawling the same data. Here we have used around 20 threads for crawling process. In ideal case our crawler should use lesser number of threads to avoid overloading the website or getting blocked. Also we added random sleep periods to show robot politeness which will not cause the site to block our crawler. Here our goal was to maximize the download rate while minimizing the overhead from parallelization and to avoid repeated downloads of the same page to crawl multiple links using the concept of multithreading.

9. REFERNCES

[1] Duen Horng Chau, Shashank Pandit, Samuel Wang, Christos Faloutsos Carnegie Mellon University Pittsburgh, PA 15213, US “AParallel Crawling for Online Social Networks”

[2] Mike Thelwall “A web crawler design for data mining”

[3] Vladislav Shkapenyuk Torsten Suel “Design and Implementation of a High-Performance Distributed Web Crawler”

[4] Harry Tursulistyono, Yani Achsan, Wahyu Catur Wibowo, Faculty of Computer Science, Kampus Depok Universitas, Indonesia Wahyuningdiah Trisari Harsanti Putri M. Muhtar Baswara Achsan Quintin Kurnia Dikara Barcah Faculty of Science & Engineering Universitas Paramadina Jakarta, Indonesia

“ Harvesting Bibliography Multi-thread, Safe and Ethical Web Crawling”

[5] Ming Ke, PengZhou Zhang, GuoWei Chen “The Crawler of Specific Resources Recognition Based on Multi-thread”

[6] Seonyoung Park, Youngseok Lee Chungnam National University Daejeon, Republic of Korea

“ Implementation of A Distributed Web Community Crawler ”

[7] Feng Ye, Zongfei Jing, Qian Huang, College of Computer and Information Hohai University, Yong Chen, Postdoctoral centre Nanjing Longyuan Micro-Electronic Company

“The Research of a Lightweight Distributed Crawling System”

[8] Sawroop Kaur Bal, G.Geetha “ Smart distributed web crawler ”

[9] Aveksha Kapoor, Vinay Arora "Application of bloom filter for duplicate URL detection in a web crawler"

[10] Weipeng Zhou, Pan Wang, Xuejiao Chen, Feng Ye "An Improved Bloom Filter In Distributed Crawler"