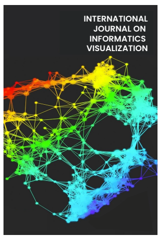
INTERNATIONAL JOURNAL
ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv



Evaluating Web Scraping Performance Using XPath, CSS Selector, Regular Expression, and HTML DOM With Multiprocessing Technical Applications

Irfan Darmawan [a], Muhamad Maulana [b], Rohmat Gunawan [b,*], Nur Widiyasono [b]

[a] *Department of Information System, Telkom University, Bandung, Indonesia*
[b] *Department of Informatics, Siliwangi University, Tasikmalaya, Indonesia*
Corresponding author: [*]rohmatgunawan@unsil.ac.id

***Abstract*— Data collection has become a necessity today, especially since many sources of data on the internet can be used for various needs. The main activity in data collection is collecting quality information that can be analyzed and used to support decisions or provide evidence. The process of retrieving data from the internet is also known as web scraping. There are various methods of web scraping that are commonly used. The amount of data scattered on the internet will be quite time-consuming if the web scraping is done on a large scale. By applying the parallel concept, the multi-processing approach can help complete a job. This study aimed to determine the performance of the web scraping method with the application of multi-processing. Testing is done by doing the process of scraping data from a predetermined target web. Four web scraping methods: CSS Selector, HTML DOM, Regex, and XPath, were selected to be used in the experiment measured based on the parameters of CPU usage, memory usage, execution time, and bandwidth usage. Based on experimental data, the Regex method has the least CPU and memory usage compared to other methods. While XPath requires the least time compared to other methods. The CSS Selector method is the smallest in terms of bandwidth usage compared to other methods. The application of multi-processing techniques to each web scraping method is proven to save memory usage, reduce execution time and reduce bandwidth usage compared to only using single processing.**

***Keywords*— Multiprocessing; scraping; website; HTML DOM.**

Manuscript received 10 Jan. 2022; revised 25 Mar. 2022; accepted 18 Apr. 2022. Date of publication 31 Dec. 2022.

I. INTRODUCTION

Web scraping is a technique for extracting data from a website and saving it to a file system or database for some purpose. The extracted web data needs to pass through the Hypertext Transfer Protocol (HTTP) protocol or a web browser [1] [2]. Web scraping is an effective and efficient technique for obtaining reliable, fast, and automatic information in extracting and storing data from a website [3]–[5]. The existence of this web scraping technique can be used for research purposes, data analysis, and information collection from various media on the internet automatically. Examples of the use of web scraping include the acquisition and categorization of web page information related to hydroponics [6], classification of job vacancies based on data search results on the internet [7], and analysis of natural disaster information [8], Twitter web scraping [9] [10], web scraping on GitHub [11]–[14], web scraping data on google scholar [15], web scraping on Instagram [16] [17].

Various web scraping methods have been commonly used, including traditional copy and paste, Regular Expression (Regex) [16], Hypertext Markup Language Document Object Model (HTML DOM), Xpath [18] , and CSS Selector [18]–[21]. Apart from that, several programming languages have also developed various libraries to support the web scraping process, including Beautifulsoup, lxml, and scrappy, which are commonly used to do web scraping in python programming [22].

In data-intensive computing applications, the current hardware allows for parallel program execution [23]–[25] [26]. The application of parallel and multiprocessor algorithms can break down significant numerical problems into smaller subtasks, reducing the total computation time on multiprocessor computers and resulting in better performance [23]. In dealing with this parallel computing problem, the concept of a processing "pool" is used: "tasks" (data) are forwarded in bulk to the pool, and the pool handles the distribution of tasks to a number of available worker processes [27]–[29].

Optimization of hardware so that it can work in parallel during program execution is one of the interesting things to research. For programs to be executed in parallel, a special configuration is required for better performance. This research aims to conduct a comparative study of web scraping performance by applying multi-processing techniques. The experiment compared web scraping performance by applying single-processing and multi-processing techniques. Python programming language was chosen to be used in this research because it is open source, multiplatform, lightweight, compact, and very suitable for rapid prototyping, although it is powerful enough to write significant applications [23].

II. Material and Method

A. *XPath*

XPath (XML Path Language) is a query language to select parts (nodes) of an XML document [30] In addition to selecting, XPath can also be used to calculate values such as strings, numbers, and Booleans in an XML and HTML file. The World Wide Web Consortium (W3C) has even set standards for the use of XPath.

B. *CSS Selector*

CSS Selector is a method for finding HTML elements on web pages and extracting data from them [20]. A CSS Selector is declared as part of a markup style that applies to match the tags and attributes in the markup.

C. *HTML DOM*

HTML DOM is a standard object model for getting, changing, adding, or removing HTML elements [31]. The DOM works by defining the objects and properties of all HTML elements, with methods to access them. A web browser does not require the use of the DOM to display HTML documents. But with the DOM, Javascript can access all the elements in the HTML document.

D. *Regex*

Regular Expression (Regex) is a language construction to match text based on certain patterns, especially for complex cases. Regex is also used to match certain character patterns in a set of strings[32]. Regex has two kinds of characters, namely regular characters and meta characters.

E. *Multi-processing*

Multi-processing is the ability of a system to support more than one processor at the same time [33]. A program that uses multi-processing will be broken down into smaller routines that run independently. The operating system will allocate these threads to the processor, which increases system performance. The use of multi-processing by making the processing processes parallel is necessary to achieve the best performance. Each multi-processing task will run in its own process, and each program running on the computer is represented by one or more processes [26], [34].

There are six main stages carried out in this study, as shown in Fig 1.

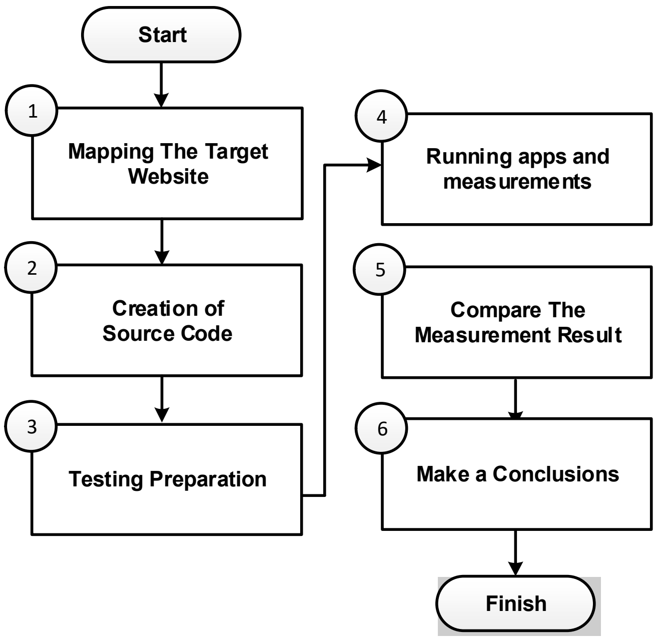
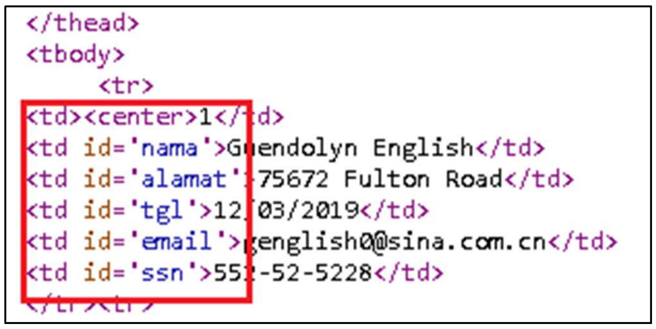

Start
1
Mapping The Target Website
2
Creation of Source Code
3
Testing Preparation
4
Running apps and measurements
5
Compare The Measurement Result
6
Make a Conclusions
Finish

Fig. 1 Research Stages

F. *Mapping* The Target Website

Website mapping is one way to identify meta attributes that contain data objects. This activity is carried out to obtain information on the data object to be retrieved, as shown in Fig 2.



```
</thead>
<tbody>
    <tr>
<td><center>1</td>
<td id='nama'>Gwendolyn English</td>
<td id='alamat'>75672 Fulton Road</td>
<td id='tgl'>12/03/2019</td>
<td id='email'>english0@sina.com.cn</td>
<td id='ssn'>55[illegible]-52-5228</td>
```

Fig. 2 The target site's web page marked on the id element

G. *Creation* of Source Code

The source code creation of the program is divided into two versions. For the first version, the program code is made without multi-processing, while the second version uses multi-processing. This is done to examine the difference between the two from the results of the test parameters obtained. Each version of the code consists of four different web scraping methods, so a total of eight program code sources are created.

Source code programs created using different libraries are adapted for compatibility with each method used. XPath uses the lxml library, Regex uses the re library, CSS selector uses BeautifullSoup, and Html DOM uses htmldom. The source code snippet for each web scraping method used in the experiment is shown in Figure 3-6.

```python
# Memanggil Library yang dibutuhkan CSS Selector
import requests
from bs4 import BeautifulSoup
from multiprocessing import Pool
# Membuat fungsi scraping
def ini_css(url):
    res = requests.get(url)
    soup = BeautifulSoup(res.text, 'html.parser')
    sampel = soup.find('table', {'id':'example'})
    tbody = sampel.find('tbody')

    for tr in tbody.find_all('tr'):
        nomor = tr.find_all('td')[0].text.strip()
        nama = tr.find_all('td')[1].text.strip()
        alamat = tr.find_all('td')[2].text.strip()
        tgl = tr.find_all('td')[3].text.strip()
        email = tr.find_all('td')[4].text.strip()
        ssn = tr.find_all('td')[5].text.strip()
        print('|',nomor,'|',nama,'|',alamat,'|',tgl,'|',email,'|',s
sn,'|')
# Proses Multiprocessing menggunakan Pool
p = Pool(10)
# Menjalankan fungsi scraping css selector dengan multiprocessing
p.map(ini_css, all_urls)
p.terminate()
p.join()
```

Fig. 3 CSS Selector Code Snippet

```python
# Memanggil library kebutuhan
import re
import urllib.request
from multiprocessing import Pool
# Membuat fungsi scraping regex
def ini_regex(url):
    response = urllib.request.urlopen(url).read()
    text = response.decode()
    cari_nama = re.compile("<td id='nama'>(.*)</td>")
    cari_alamat = re.compile("<td id='alamat'>(.*)</td>")
    cari_tgl = re.compile("<td id='tgl'>(.*)</td>")
    cari_email = re.compile("<td id='email'>(.*)</td>")
    cari_ssn = re.compile("<td id='ssn'>(.*)</td>")

    nama = re.findall(cari_nama,text)
    alamat = re.findall(cari_alamat,text)
    email = re.findall(cari_email,text)
    tgl = re.findall(cari_tgl,text)
    ssn = re.findall(cari_ssn,text)

    batch = []
    batch[:] = range(1)
    for i in batch:
        print(nama,tgl,email,alamat,ssn)
# Proses multiprocessing
p = Pool(10)
p.map(ini_regex, all_urls)
p.terminate()
p.join()
```

Fig. 4 Regex Code Snippet

```python
# Memanggil library kebutuhan
import requests
from htmldom import htmldom
from multiprocessing import Pool
# Membuat fungsi scraping htmldom
def ini_htmldom(url):
    dom = htmldom.HtmlDom(url).createDom()
    nama = dom.find("td[id=nama]")
    alamat = dom.find("td[id=alamat]")
    data = dom.find("td[id=data]")
    tgl = dom.find("td[id=tgl]")
    email = dom.find("td[id=email]")
    ssn = dom.find("td[id=ssn]")

    nm = [(n.text()) for n in nama]
    em = [(e.text()) for e in email]
    al = [(a.text()) for a in alamat]
    tg = [(t.text()) for t in tgl]
    sn = [(s.text()) for s in ssn]
    print(nm,al,tg,em,sn)
# Proses multiprocessing
p = Pool(10)
p.map(ini_htmldom, all_urls)
p.terminate()
p.join()
```

Fig. 5 HTML DOM Code Snippet

```python
# Memanggil library kebutuhan
import requests
from lxml import html
from multiprocessing import Pool
# Membuat fungsi scraping regex
def ini_xpath(url):
    site = requests.get(url)
    data = html.fromstring(site.content)
    nama = data.xpath("//td[@id='nama']/text()")
    alamat = data.xpath("//td[@id='alamat']/text()")
    tgl = data.xpath("//td[@id='tgl']/text()")
    email = data.xpath("//td[@id='email']/text()")
    ssn = data.xpath("//td[@id='ssn']/text()")

    print(nama,tgl,email,alamat,ssn)
# Proses multiprocessing
p = Pool(10)
p.map(ini_xpath, all_urls)
p.terminate()
p.join()
```

Fig. 6 Xpath Code Snippet

Apart from programming by applying four different scraping methods, other program codes were also used in the experiment, including program code to calculate processing time, program code to calculate memory usage, program code to calculate CPU usage, and program code to calculate Bandwidth usage. The snippet of the program code to calculate the scraping process time using the os library is shown in Fig 7.

```python
import os
# Menghitung Waktu Proses
start = time.time()
# Bagian Kode Program

# End Time Process
end = time.time()
# Hasil Waktu Proses
print("Waktu proses : {} second".format(end-
start))
```

Fig. 7 Code Snippet Time Process Calculate

Fig.7 shows the time.time() function contained in the os library, which is used to get time information. The time data obtained before the main program is run is stored in the start variable, while the time data obtained after the main program is stored in the end variable. The time used for the main program to run is calculated based on the difference between the end and start variable values. In calculating memory usage, the psutil library is used. The snippet of the memory usage calculation code is shown in Fig 8.

```python
import psutil
# Menghitung Penggunaan Memori Start
memori1 = psutil.Process(os.getpid()).memory_info().rs
s
# Bagian Kode Program

# Menghitung Penggunaan Memori End
memori2 = psutil.Process(os.getpid()).memory_info().rs
s
# Hasil Penggunaan Memori
print("Penggunaan memory : {} bytes".format(memori2-
memori1))
```

Fig. 8 Code Snippet Calculating Memory Usage

Fig. 8 shows that the memory_info().rss function is applied to the program code to get information about the memory used. Memory usage data obtained before the main program is run is stored in memory variable1, while memory usage data obtained after the main program is run is stored in the memory2 variable. Memory usage while the main program is running is obtained based on the difference between the values of the memory2 and memory1 variables.

```
import psutil
# Menghitung Penggunaan CPU Awal
cpu0 = psutil.cpu_percent()
# Bagian Kode Program

# Menghitung Penggunaan CPU Awal
cpu = psutil.cpu_percent()
# Hasil Penggunaan CPU
print("CPU yang digunakan : {} %".format(cpu))
```

Fig. 9 Code Snippet Calculating CPU Usage

Fig. 9 shows that the psutil.cpu_percent() function is applied to the program code to get information about the CPU being used. CPU usage data obtained before the main program is run is stored in variable memory0, while CPU usage data obtained after the main program is run is stored in variable cpu1. CPU usage while the main program is running is obtained based on the difference between the values of the cpu1 and cpu0 variables.

```
import psutil
# Menghitung Bandwith Upstream dan Downstream
up = psutil.net_io_counters().bytes_sent
down = psutil.net_io_counters().bytes_recv
# Bagian Kode Program

# Menghitung Bandwith Usage
up0 = psutil.net_io_counters().bytes_sent
down0 = psutil.net_io_counters().bytes_recv
# Hasil Bandwidth Upstream & Downstream
print("Penggunaan Bandwidth : Upstream = {} Do
wnstream = {}".format(up0-up,down0-down))
```

Fig. 10 Code Snippet Calculating Bandwidth Usage

Fig. 10 shows the program code snippet using the psutil library with the functions net_io_counter().byte_sent and net_io_counter().byte_recv applied to the program code to get bandwidth usage information.

A. Testing Preparation

The hardware and software used in the experiment were prepared at this stage. The hardware and software specifications used in the experiment are shown in Table I.

TABLE I
HARDWARE AND SOFTWARE SPECIFICATION

No.	Item	Specification
1.	CPU	Intel(R) Xeon(R) CPU @ 2.30GHz
2.	Memory	4 GB
3.	Operating System	Debian 4.9.228-1 x86_64
4.	Programing Language	Python 3.8.0

B. Running Apps and Measurement

The web scraping program code by applying four different methods is executed at this stage. The data obtained based on each parameter of the experiment are recorded in the table. The experiment was repeated 20 times.

C. Compare The Measurement Result

Measurement data from each experiment were collected, and each parameter's average value was taken. Then, the experimental data was compared between the four web scraping methods used.

D. Make a Conclusion

At this stage, conclusions are drawn from the measurement data that have been compared between the four methods used by applying multi-processing techniques and without applying multi-processing techniques.

III. RESULT AND DISCUSSION

This section presents the results of the experiments that have been carried out. Every web scraping method that has been implemented in the program code is executed. Experimental data based on each parameter are recorded and presented in a table.

A. The Measurement Result of CPU Usage

Table II displays CPU usage data at the time of web scraping execution for each method. From the experimental results, the data obtained are as follows: the CSS Selector method uses an average of 25.6% CPU and there is an increase to an average of 66.2% when multi-processing techniques are applied, HTML DOM uses an average of 41.3% CPU and there is an increase an average of 80% when multi-processing techniques are applied, Regex uses an average of 1% CPU and an increase to an average of 6% when multi-processing techniques are applied, while XPath uses an average of 9.4% CPU and there is an increase to an average of 38% average when multi-processing technique is applied.

TABLE II
CPU USAGE MEASUREMENT RESULTS

No	CSS SELECTOR		HTMLDOM		REGEX		XPATH	
	Single proces sing	Multi proces sing	Single proce ssing	Multi proce ssing	Single proce ssing	Multi proce ssing	Single proce ssing	Multi proce ssing
1	25,3	67,6	42,0	80,2	0,8	6,3	9,7	36,1
2	25,9	65,5	42,8	79,9	0,9	6,3	10,0	36,9
3	25,8	66,5	42,6	79,5	1,2	6,7	9,7	37,3
4	25,9	64,0	42,9	80,0	0,8	6,4	9,6	38,0
5	26,0	66,0	41,2	79,7	1,1	2,3	9,6	38,4
6	26,6	67,2	42,0	79,7	0,9	6,9	10,0	37,4
7	26,0	67,0	42,9	80,2	1,0	6,2	9,2	37,8
8	26,2	63,9	42,1	80,1	1,0	5,8	9,3	37,3
9	25,8	66,2	42,3	79,9	0,9	6,5	9,7	37,7
10	27,2	66,7	42,2	80,5	1,2	6,4	9,3	37,4
11	25,8	67,0	42,4	80,0	1,2	6,5	9,7	37,3
12	25,9	66,1	33,1	80,0	0,9	6,3	9,2	37,7
13	26,7	66,2	42,4	79,9	1,2	6,8	9,2	38,2
14	25,9	65,8	41,0	80,1	0,9	6,3	9,6	38,5
15	25,9	66,0	42,3	79,9	0,9	4,5	9,3	37,3
16	26,3	65,6	43,1	79,9	1,4	6,2	9,6	37,7
17	26,0	65,9	41,8	80,1	1,6	6,8	8,9	37,4
18	17,0	66,8	41,9	80,1	1,0	6,2	9,0	37,3
19	25,7	66,3	41,8	79,9	0,7	4,4	9,1	38,6
20	26,7	67,0	42,2	80,1	0,9	6,6	9,1	38,5
AVG	**25,6**	**66,2**	**41,8**	**80,0**	**1,0**	**6,4**	**9,4**	**38,0**

B. The Measurement Result of Memory Usage

Data on memory usage during web scraping execution for each method is shown in Table III. From the experimental results, the data obtained are as follows: the CSS Selector method uses an average of 29,070 KB of memory and there is an increase to an average of 334 KB when multi-processing techniques are applied, HTML DOM uses an average of 158.3 KB of memory and there is an increase to an average 340 KB when the multi-processing technique is applied, Regex uses an average of 160.9 KB of memory and there is an increase to an average of 359 KB when multi-processing techniques are applied, while Xpath uses an average of 114 KB of memory and there is an increase to an average of 351 KB when the multi-processing technique is applied. multi-processing technique applied.

TABLE III
MEMORY USAGE MEASUREMENT RESULTS

No	CSS SELECTOR		HTMLDOM		REGEX		XPATH	
	Single proces sing	Multi proces sing	Single proce ssing	Multi proce ssing	Single proce ssing	Multi proce ssing	Single proce ssing	Multi proce ssing
1	28155	360	158707	380	1564	307	114606	319
2	29442	299	157741	368	1556	430	118128	385
3	29282	352	158699	303	1630	323	115261	372
4	29212	303	158429	307	1634	364	113295	327
5	28954	307	158617	327	1503	327	112230	360
6	29175	364	157708	372	1626	311	111616	327
7	28954	372	158760	307	1695	307	115220	339
8	29081	299	157790	368	1622	327	115015	380
9	29204	319	158638	311	1568	331	111493	372
10	29188	360	157741	385	1691	446	118087	393
11	29392	364	158605	368	1548	307	115056	303
12	28368	299	158617	303	1695	442	114524	323
13	29327	356	158703	372	1626	425	115261	380
14	29310	364	158650	311	1560	331	109199	335
15	28770	368	158638	323	1695	331	114606	385
16	29196	307	157528	368	1626	446	115589	376
17	29237	307	158646	319	1560	303	111534	323
18	29085	311	158580	385	1630	372	117432	376
19	28573	368	156971	319	1576	450	112230	335
20	29503	303	158638	311	1572	307	110182	311
AVG	**29070**	**334**	**158300**	**340**	**1609**	**359**	**114028**	**351**

C. The Measurement Result of Execution Time

Table IV displays execution time usage data at the time of web scraping execution for each method. From the experimental results, the data obtained are as follows: the CSS Selector method requires an average execution time of 10.87 seconds and the execution time becomes smaller on average 4.43 seconds when multi-processing techniques are applied, HTML DOM requires an average execution time of 22.65 seconds and the execution time becomes smaller on average 12.68 seconds when the multi-processing technique is applied, Regex requires an average execution time of 12.89 seconds and the execution time becomes smaller on average 3.1 seconds when the multi-processing technique is applied, while Xpath the average execution time is 8.33 seconds and the execution time becomes smaller on average 2.32 seconds when multi-processing techniques are applied.

TABLE IV
EXECUTION TIME MEASUREMENT RESULT

No	CSS SELECTOR		HTMLDOM		REGEX		XPATH	
	Single proces sing	Multi proces sing	Single proce ssing	Multi proce ssing	Single proce ssing	Multi proce ssing	Single proce ssing	Multi proce ssing
1	10.58	5.02	21.84	14.45	13.03	2.81	8.56	2.51
2	10.58	4.52	22.01	12.54	12.87	2.81	8.25	2.31
3	10.71	4.52	22.06	12.64	12.89	2.91	8.46	2.31
4	10.68	4.52	22.06	12.74	13.31	2.61	8.33	2.31
5	10.54	4.42	22.08	12.64	12.82	7.92	8.32	2.31
6	10.62	4.32	22.11	12.64	12.71	2.71	8.32	2.31
7	10.64	4.32	22.14	12.54	12.79	2.71	8.27	2.31
8	10.49	4.52	22.17	12.65	12.83	2.71	8.30	2.31
9	10.64	4.42	22.18	12.64	12.98	2.71	8.23	2.31
10	10.50	4.32	22.19	12.74	13.10	2.61	8.24	2.31
11	10.57	4.32	22.26	12.44	12.74	2.71	8.27	2.31
12	10.69	4.42	22.37	12.54	12.82	2.81	8.33	2.31
13	10.42	4.32	22.37	12.44	12.83	2.71	8.32	2.31
14	10.56	4.42	22.58	12.64	13.15	2.81	8.25	2.31
15	10.64	4.32	22.60	12.54	12.72	3.31	8.33	2.31
16	10.44	4.42	22.74	12.64	13.03	2.71	8.36	2.31
17	10.54	4.42	22.74	12.55	12.77	2.71	8.43	2.31
18	16.26	4.32	22.81	12.44	12.76	2.71	8.33	2.31
19	10.67	4.42	22.89	12.65	12.65	4.21	8.34	2.31
20	10.71	4.42	28.74	12.44	12.96	2.81	8.31	2.31
AVG	**10.87**	**4.43**	**22.65**	**12.68**	**12.89**	**3.10**	**8.33**	**2.32**

D. The Measurement Result of Bandwidth Usage

Bandwidth usage involves uploading and downloading data. Bandwidth usage is calculated based on the difference between these activities. Data on bandwidth usage results when conducting web scraping experiments without applying multi-processing techniques are shown in Table V. While data on bandwidth usage results when conducting web scraping experiments using multi-processing techniques are shown in Table VI. From the experimental results before applying the multi-processing technique, the following data are obtained: the CSS Selector method uses an average bandwidth of 39,944 Kbps upstream and 325,416 Kbps downstream, the average HTML DOM uses a bandwidth of 126,948 Kbps upstream and 1,780,577 Kbps downstream, Regex averages using a bandwidth of 82,669 Kbps upstream and 1,756,610 Kbps downstream, while the average Xpath uses a bandwidth of 159,780 Kbps upstream and 386,299 Kbps downstream.

TABLE V
BANDWIDTH USAGE MEASUREMENT RESULT (WITHOUT MULTIPROCESSING)

NO	CSS SELECTOR		HTMLDOM		REGEX		XPATH	
	UP	DOWN	UP	DOWN	UP	DOWN	UP	DOWN
1	39528	324975	70859	1750847	57767	1729035	1600025	1136765
2	39884	324466	1180752	2478296	58208	1729299	1057087	890044
3	39181	324148	84111	1757479	83567	1755653	64366	357740
4	41397	323549	80008	1753818	86273	1757579	38273	319972
5	42861	327356	62145	1741082	84922	1754761	50799	340526
6	43087	327738	72628	1742739	83930	1752537	91349	387684
7	43501	327526	61205	1731294	87480	1758816	18750	302349
8	24257	311653	67099	1743519	94012	1767327	20352	309390
9	40363	324978	67865	1739940	87574	1766236	32373	323477
10	39729	329412	65176	1738541	83538	1754655	17782	302570
11	39181	323321	66897	1737130	83750	1755638	18413	301744
12	41620	325883	67487	1741891	81159	1755332	23522	309481
13	40450	326134	66118	1745114	80510	1749175	18913	302326
14	41158	324212	83648	1763233	86645	1762348	20251	306447
15	37909	321797	58463	1729829	79775	1751486	23208	314142
16	40670	328083	76376	1754378	88643	1763215	20175	305940
17	39913	329382	64179	1734396	147157	1851636	18421	301624
18	42150	325986	65869	1734825	71332	1747634	22019	307113
19	41606	331942	63149	1735524	56808	1726009	20376	304244
20	40442	325775	58838	1727932	70326	1743824	19141	302392
AVG	**39944**	**325416**	**126948**	**1780577**	**82669**	**1756610**	**159780**	**386299**

Data were obtained when multi-processing techniques were applied and experiments were carried out, as shown in Table VI.

TABLE VI
BANDWIDTH USAGE MEASUREMENT RESULT (MULTI-PROCESSING)

NO	CSS SELECTOR		HTMLDOM		REGEX		XPATH	
	UP	DOWN	UP	DOWN	UP	DOWN	UP	DOWN
1	26598	308506	61210	1732078	60407	1726977	29081	311957
2	22598	310498	68739	1738624	59377	1732961	24876	309800
3	21515	303362	1047223	2249842	61684	1736757	24283	308663
4	22830	308630	1042069	2244203	66164	1734415	23907	304975
5	22518	305981	68569	1742571	62109	1735275	26413	308459
6	23418	306521	69395	1745932	67076	1736083	23948	305448
7	23150	307607	62205	1729499	59738	1729392	22095	302522
8	24179	309527	70961	1744613	60548	1730105	22205	306759
9	23195	309171	72074	1746492	62816	1737185	23552	307838
10	23472	306564	206117	1986352	60464	1727275	25553	309231
11	21642	303506	65112	1734439	68833	1741652	25196	310629
12	21922	301861	65267	1735010	64604	1737768	23352	310549
13	23997	315248	62521	1732808	60552	1729934	22564	304269
14	26790	309801	63427	1730984	68618	1742880	23074	303341
15	24987	305983	64420	1732829	66382	1735971	24196	306202
16	22648	309160	64366	1731636	69331	1739131	21008	301381
17	24893	309963	61508	1734375	62638	1734983	24042	310877
18	25446	308896	65959	1740999	68993	1741047	22048	304948
19	26278	312826	65528	1738273	67373	1740444	23618	307820
20	22944	307501	67946	1739206	72835	1751675	21910	304136
AVG	**23751**	**308056**	**170731**	**1800538**	**64527**	**1736096**	**23846**	**306990**

The CSS Selector method uses an average bandwidth of 23,751 Kbps upstream and 308.056 Kbps downstream, HTML DOM uses an average bandwidth of 170,731Kbps upstream and 1,800,538 Kbps downstream, Regex uses an average bandwidth of 64,527 Kbps upstream and 1,736,096 Kbps downstream. Xpath uses an average bandwidth of 2,384 Kbps upstream and 306,990 Kbps downstream.

After conducting experiments for each selected method and calculating the average value for each parameter, then it is compared to determine the performance based on the four selected parameters, as shown in Table VII and Table VIII.

TABLE VII
COMPARISON OF THE AVERAGE VALUE OF CPU USAGE, MEMORY USAGE AND EXECUTION TIME

No	Method		CPU Usage	Memory Usage	Execution Time
1	CSS Selector	Without Multiprocessing	25,60 %	29.070 KB	10.87 s
		With Multi-processing	66,20 %	334 KB	4.43 s
2	HTML DOM	Without Multiprocessing	41,80 %	158.300 KB	22.65 s
		With Multi-processing	80,00 %	340 KB	12.68 s
3	REGEX	Without Multiprocessing	**1,00 %**	**1.609 KB**	11.402 KB
		With Multi-processing	**6,40 %**	**359 KB**	351 KB
4	XPATH	Without Multiprocessing	9,40 %	11.402 KB	**8.33 s**
		With Multi-processing	38,00 %	351 KB	**2.32 s**

From the data in Table VII, it can be seen that the Regex method has the least CPU and memory usage compared to the CSS Selector, HTML DOM, and XPath methods. Whereas XPath takes the least amount of time to run web scraping compared to other methods.

TABLE VIII
COMPARISON OF THE AVERAGE VALUE OF BANDWIDTH USAGE

No	Method		Bandwidth Usage (Kbps)	
1	CSS Selector	Without Multiprocessing	Upstream	**39.944**
			Downstream	**325.416**
		With Multi-processing	Upstream	**23.751**
			Downstream	**308.056**
2	HTML DOM	Without Multiprocessing	Upstream	126.948
			Downstream	1.780.577
		With Multi-processing	Upstream	170.731
			Downstream	1.800.538
3	REGEX	Without Multiprocessing	Upstream	82.669
			Downstream	1.756.610
		With Multi-processing	Upstream	64.527
			Downstream	1.736.096
4	XPATH	Without Multiprocessing	Upstream	159.780
			Downstream	386.299
		With Multi-processing	Upstream	23.846
			Downstream	306.990

The data in Table VIII shows that the CSS Selector method uses the smallest bandwidth compared to the HTML DOM, Regex, and XPath methods.

IV. CONCLUSION

Based on experimental data, the Regex method has the least CPU and memory usage compared to the CSS Selector, HTML DOM, and XPath methods. Whereas XPath takes the least amount of time compared to other methods. The CSS Selector method is the smallest in terms of bandwidth usage compared to other methods. The application of multi-processing techniques can save memory usage, reduce execution time and reduce bandwidth usage. However, this will increase the CPU workload due to the optimization of the cores contained in it.

REFERENCES

[1] B. Zhao, “Encyclopedia of Big Data,” *Encycl. Big Data*, pp. 3–5, 2019, doi: 10.1007/978-3-319-32001-4.

[2] M. El Asikri[1], S. Krit, and H. Chaib, “Using Web Scraping In A Knowledge Environment To Build Ontologies Using Python And Scrapy Article in,” *Eur. J. Transl. Clin. Med.*, vol. 07, no. 03, pp. 433–442, 2020.

[3] S. E. Chasins, M. Mueller, and R. Bodik, “Rousillon: Scraping distributed hierarchical web data,” *UIST 2018 - Proc. 31st Annu. ACM Symp. User Interface Softw. Technol.*, pp. 963–975, 2018, doi: 10.1145/3242587.3242661.

[4] O. ten Bosch, D. Windmeijer, A. Van Delden, and G. Van den Heuvel, “Web scraping meets survey design: Combining forces,” *Bigsurv18 Conf.*, pp. 1–13, 2018.

[5] A. V Saurkar and S. A. Gode, “An Overview On Web Scraping Techniques And Tools,” *Int. J. Futur. Revolut. Comput. Sci. Commun. Eng.*, pp. 363–367, 2018.

[6] A. Priyanto and M. R. Ma’arif, “Implementasi Web Scrapping dan Text Mining untuk Akuisisi dan Kategorisasi Informasi dari Internet (Studi Kasus: Tutorial Hidroponik),” *Indones. J. Inf. Syst.*, vol. 1, no. 1, pp. 25–33, 2018, doi: 10.24002/ijis.v1i1.1664.

[7] C. Slamet, R. Andrian, D. S. Maylawati, Suhendar, W. Darmalaksana, and M. A. Ramdhani, “Web Scraping and Naïve Bayes Classification for Job Search Engine,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 288, no. 1, pp. 0–7, 2018, doi: 10.1088/1757-899X/288/1/012038.

[8] I. P. Sonya, “Analisis Web Scraping untuk Data Bencana Alam dengan Menggunakan Teknik Breadth-First Search Terhadap 3 Media Online,” *J. Ilm. Inform. Komput. Univ. Gunadarma*, vol. 21, no. 3, pp. 69–77, 2016.

[9] I. Dongo, Y. Cadinale, A. Aguilera, F. Martínez, Y. Quintero, and S. Barrios, “Web Scraping versus Twitter API,” pp. 263–273, 2020, doi: 10.1145/3428757.3429104.

[10] J. You, J. Lee, and H. Y. Kwon, “A complete and fast scraping method for collecting tweets,” *Proc. - 2021 IEEE Int. Conf. Big Data Smart Comput. BigComp 2021*, pp. 24–27, 2021, doi: 10.1109/BigComp51126.2021.00014.

[11] R. M. Awangga, S. F. Pane, and R. D. Astuti, “Implementation of web scraping on GitHub task monitoring system,” *Telkomnika*

(Telecommunication Comput. Electron. Control., vol. 17, no. 1, pp. 275–281, 2019, doi: 10.12928/TELKOMNIKA.v17i1.11613.

[12] D. Maldeniya, C. Budak, L. P. Robert, and D. M. Romero, "Herding a Deluge of Good Samaritans: How GitHub Projects Respond to Increased Attention," *Web Conf. 2020 - Proc. World Wide Web Conf. WWW 2020*, pp. 2055–2065, 2020, doi: 10.1145/3366423.3380272.

[13] H. Hata, N. Novielli, S. Baltes, R. G. Kula, and C. Treude, "GitHub Discussions: An exploratory study of early adoption," *Empir. Softw. Eng.*, vol. 27, no. 1, pp. 1–32, 2022, doi: 10.1007/s10664-021-10058-6.

[14] M. AlMarzouq, A. AlZaidan, and J. AlDallal, "Mining GitHub for research and education: challenges and opportunities," *Int. J. Web Inf. Syst.*, vol. 16, no. 4, pp. 451–473, 2020, doi: 10.1108/IJWIS-03-2020-0016.

[15] A. Rahmatulloh and R. Gunawan, "Web Scraping with HTML DOM Method for Data Collection of Scientific Articles from Google Scholar," *Indones. J. Inf. Syst.*, vol. 2, no. 2, pp. 95–104, 2020, doi: 10.24002/ijis.v2i2.3029.

[16] L. C. Dewi, Meiliana, and A. Chandra, "Social media web scraping using social media developers API and regex," *Procedia Comput. Sci.*, vol. 157, pp. 444–449, 2019, doi: 10.1016/j.procs.2019.08.237.

[17] A. Himawan, A. Priadana, and A. Murdiyanto, "Implementation of Web Scraping to Build a Web-Based Instagram Account Data Downloader Application," *IJID (International J. Informatics Dev.*, vol. 9, no. 2, pp. 59–65, 2020, doi: 10.14421/ijid.2020.09201.

[18] T. Rizaldi and H. A. Putranto, "Perbandingan Metode Web Scraping Menggunakan CSS Selector dan Xpath Selector," *Teknika*, vol. 6, no. 1, pp. 43–46, 2017, doi: 10.34148/teknika.v6i1.56.

[19] R. Gunawan, A. Rahmatulloh, I. Darmawan, and F. Firdaus, "Comparison of Web Scraping Techniques : Regular Expression, HTML DOM and Xpath," pp. 1–8, 2019, doi: 10.2991/icoiese-18.2019.50.

[20] T. H. E. World, S. L. Web, and D. Site, "CSS Selector Reference," *w3schools.com*, 2018.

[21] M. Ahmed and I. Diab, "Prevent XPath and CSS Based Scrapers by Using Markup Randomizer," *Int. Arab J. e-Technology*, vol. 5, no. 2, pp. 78–87, 2018.

[22] O. Uzun, Erdinc; Yerlikaya, Tarik; Kirat, "Comparison of Python Libraries Used for Web Data Extraction," *Tech. Univ. - Sofia, Plovdiv branch, Bulg.*, vol. 24, 2018.

[23] Z. A. Aziz, D. Naseradeen Abdulqader, A. B. Sallow, and H. Khalid Omer, "Python Parallel Processing and Multiprocessing: A Rivew," *Acad. J. Nawroz Univ.*, vol. 10, no. 3, pp. 345–354, 2021, doi: 10.25007/ajnu.v10n3a1145.

[24] J. Kready, S. A. Shimray, M. N. Hussain, and N. Agarwal, "YouTube data collection using parallel processing," *Proc. - 2020 IEEE 34th Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2020*, pp. 1119–1122, 2020, doi: 10.1109/IPDPSW50202.2020.00185.

[25] E. Tejedor, Y. Becerra, G. Alomar, and A. Queralt, "PyCOMPSs : Parallel computational workflows in Python," 2016, doi: 10.1177/1094342015594678.

[26] A. Sherman and P. Den Hartog, "DECO : Polishing Python Parallel Programming," no. May, 2016.

[27] A. M. Price-Whelan and D. Foreman-Mackey, "schwimmbad: A uniform interface to parallel processing pools in Python," *J. Open Source Softw.*, vol. 2, no. 17, pp. 10–11, Sep. 2017, doi: 10.21105/joss.00357.

[28] A. Malakhov, "Composable Multi-Threading for Python Libraries," *Proc. 15th Python Sci. Conf.*, no. Scipy, pp. 15–19, 2016, doi: 10.25080/majora-629e541a-002.

[29] A. Malakhov, D. Liu, A. Gorshkov, and T. Wilmarth, "Composable Multi-Threading and Multi-Processing for Numeric Libraries," *Proc. 17th Python Sci. Conf.*, no. Scipy, pp. 18–24, 2018, doi: 10.25080/majora-4af1f417-003.

[30] T. Schlitt, "XML and XPath with PHP," *w3schools.com*, 2018.

[31] W3C, "What is the Document Object Model?," *w3.org*, 2016.

[32] A. Backurs and P. Indyk, "Which Regular Expression Patterns Are Hard to Match?," *Proc. - Annu. IEEE Symp. Found. Comput. Sci. FOCS*, vol. 2016-Decem, pp. 457–466, 2016, doi: 10.1109/FOCS.2016.56.

[33] M. Arif Sazali, M. Syahir Sarkawi, and N. Syazwani Mohd Ali, "Multi-processing implementation for MCNP using Python," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 1231, no. 1, p. 012003, 2022, doi: 10.1088/1757-899x/1231/1/012003.

[34] S. K. Abeykoon, M. Lin, and K. K. Van Dam, "Parallelizing X-ray Photon Correlation Spectroscopy Software Tools using Python Multiprocessing," 2017.