Distributed Web Crawler Using Dask

A Parallel Computing Approach

Tools

Python, Dask, BeautifulSoup, Pandas, Matplotlib

Presented by: Sana Banaras(037)

Presented to: Mam Madiha Wahid

What is Web Crawling?

Automated Program

Visits websites and collects data automatically.

Reads Content

Reads website content: text, links, and HTML.

Key Applications

Used for data analysis, research, and building search tools.

Why We Need Web Crawling

Efficient Data Collection

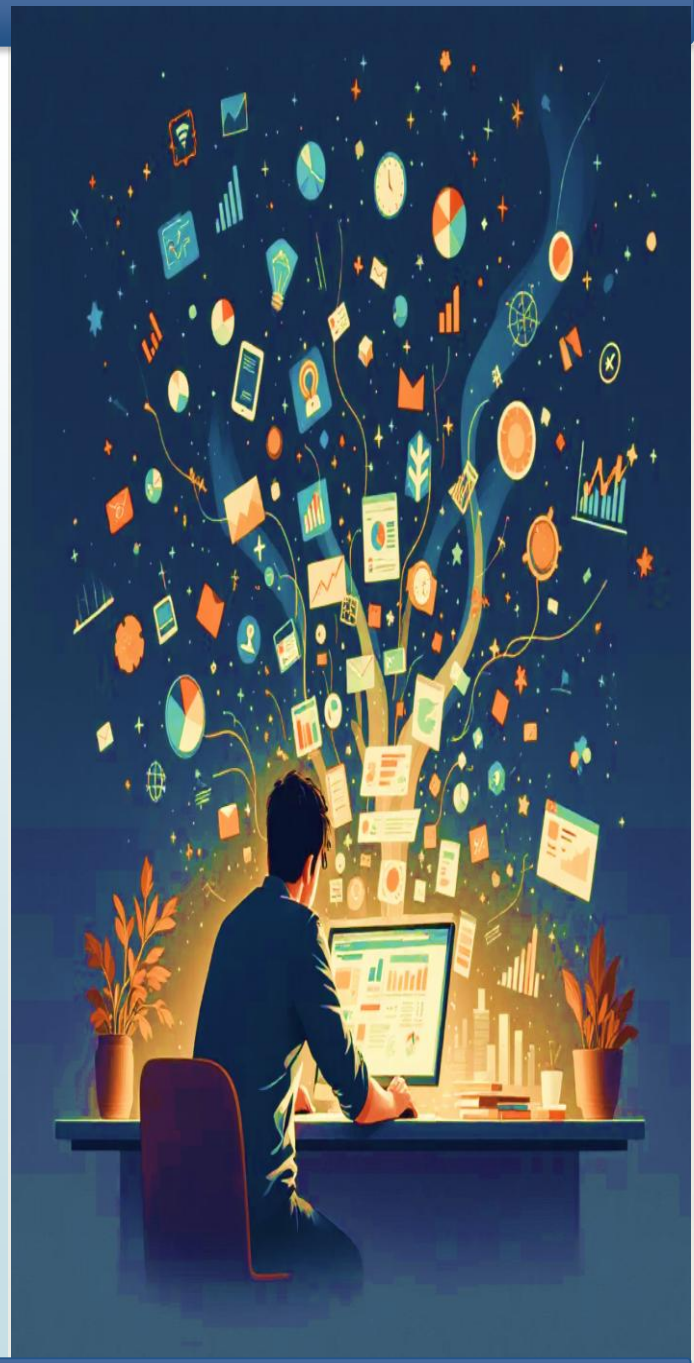
Collects vast amounts of data quickly and effectively.

Overcomes Manual Limitations

Manual data collection is slow, tedious, and prone to errors

Enabling Deep Analysis

Facilitates analysis of website content for trends and patterns.



Problems with Traditional Crawling

Sequential Visits

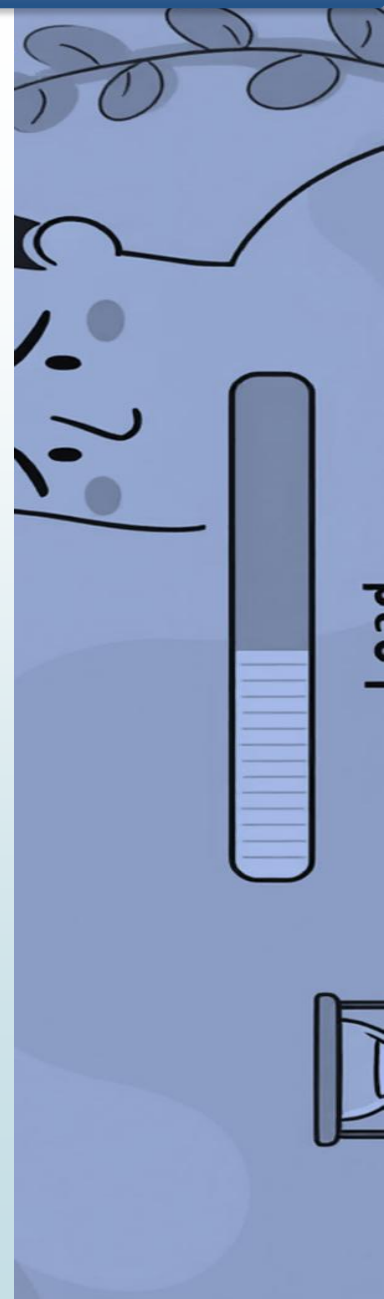
Visits one page at a time, leading to slow performance.

Inefficient CPU Usage

CPU remains idle while waiting for network responses.

Scalability Issues

Cannot efficiently handle hundreds of pages or multiple websites simultaneously.



Problems with Traditional Crawling

Dask for Parallelism

Enables simultaneous crawling of multiple web pages.

BeautifulSoup for Text

Extracts clean, readable text from complex web pages.

Word Frequency Analysis

Identifies common words and key themes within the collected data.

Visualisation

Presents results clearly using graphs and charts.

Project Goal

01

Distributed Crawler

Build a Dask-powered web crawler for parallel execution

02

Text Extraction

Extract readable text from diverse websites.

03

Frequency Analysis

Perform word-frequency analysis on extracted content.

04

Visualise Results

Present findings with charts and graphs (Matplotlib/WordCloud).

05

Performance Comparison

Compare parallel vs. sequential crawling efficiency.

Presented by: Rafia Islam

Tools And Technologies

Core

Python,
Jupyter Notebook

Parallel Computing

Dask (Dask Bag)

Fetching and Parsing

Requests,
BeautifulSoup (bs4)

Analysis and Visualization

Pandas,
Matplotlib,
WordCloud

Why Dask (Parallel Computing)

01

Parallel Execution

Utilizes all CPU cores simultaneously

02

Dask Bag

Optimized for unstructured data (like lists of URLs).

03

Overcomes GIL

Bypasses Python’s Global Interpreter Lock for true parallelism.

Data Acquisition & Processing Tools

01

Requests

Sends HTTP GET requests & handles timeouts.

02

BeautifulSoup (bs4)

Cleans messy HTML, removes tags/scripts, extracts pure text

03

Pandas

Organizes text data into DataFrames for sorting and analysis

Visualization Tools

Matplotlib

matplotlib

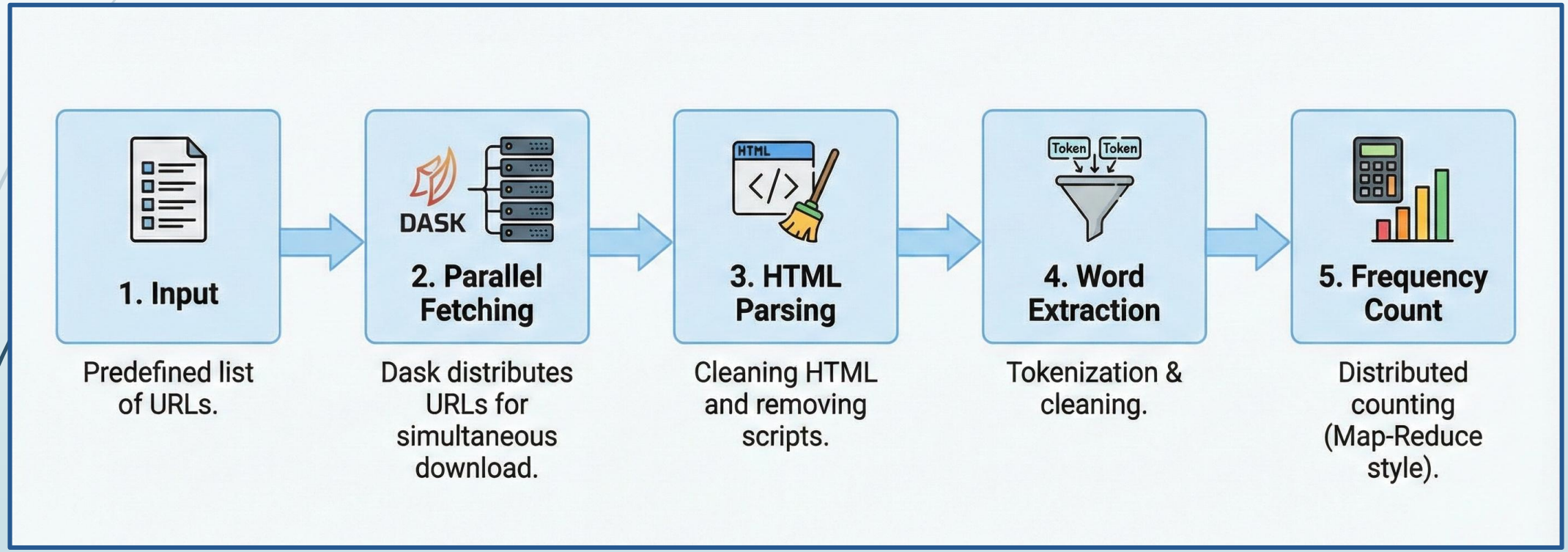
Creates Bar Charts for frequency comparison.

WordCloud

WordCloud

Generates visual summaries of most repeated words.

Architecture Workflow

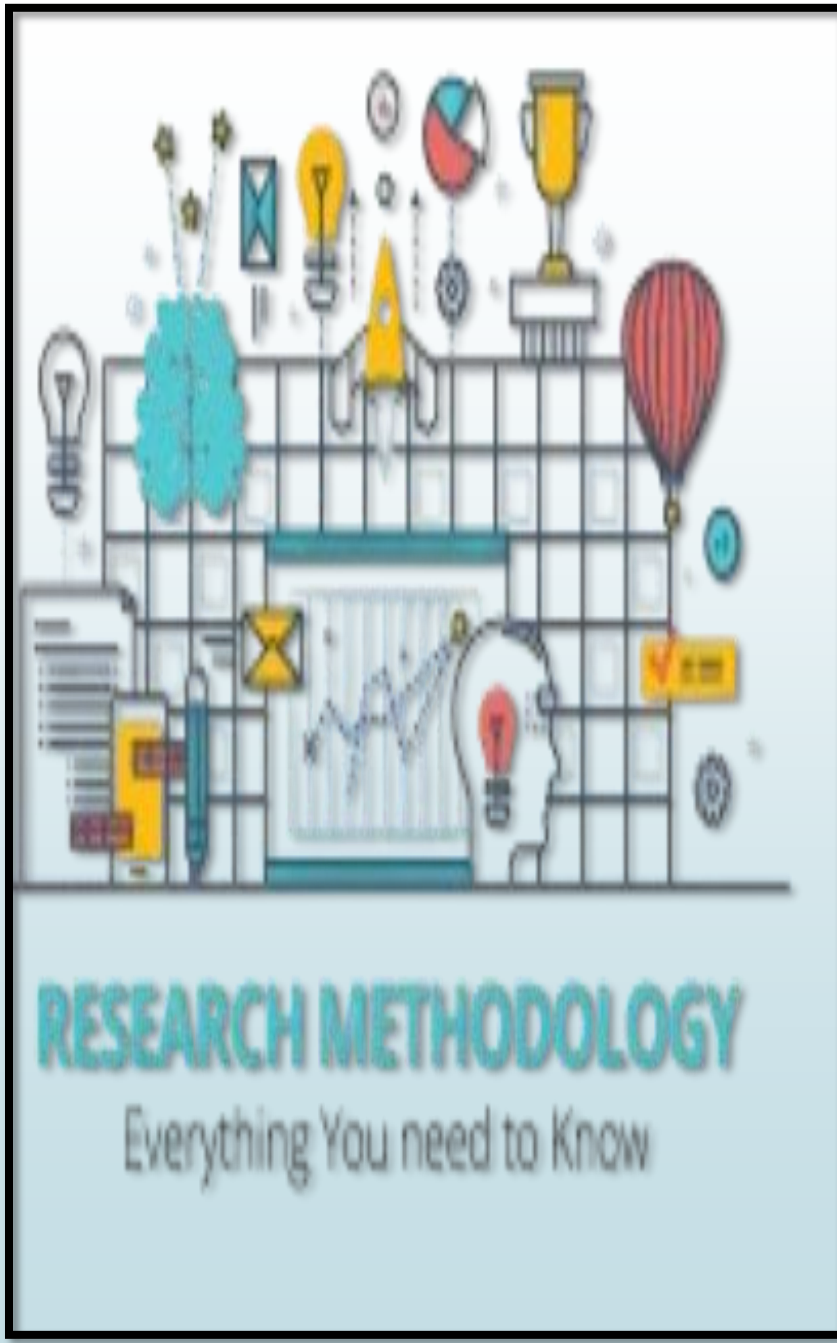


Methodology

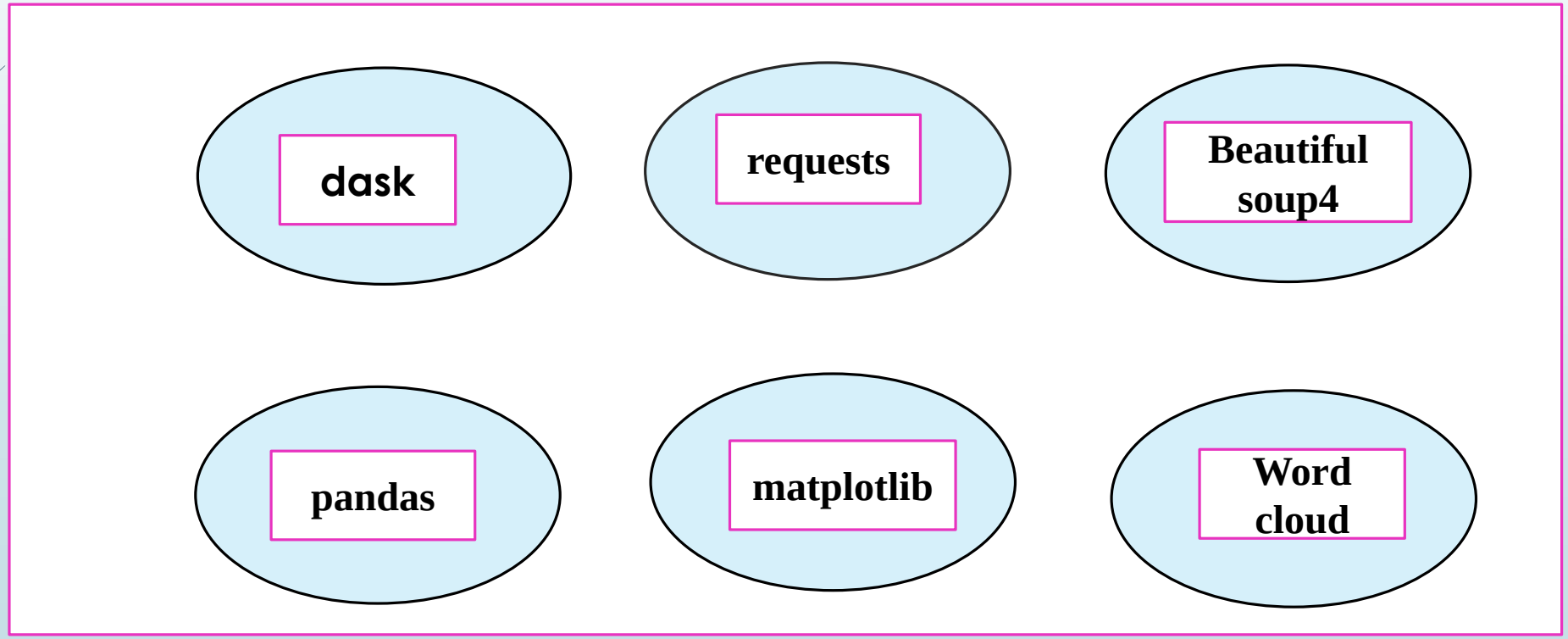
Methodology refers to the systematic process, techniques, and steps used to design, implement, and evaluate a research project or system. The complete step-by-step workflow of how the project was developed.

The methodology is divided into the following phases:

- Library Installation
- URL Dataset Preparation
- Fetch Function Development
- HTML Parsing & Text Extraction
- Sequential Crawler
- Distributed Crawler Implementation using Dask
- Data Storage & Preparation
- Visualization
- Complexity Analysis



1: Library Installation



2: URL Dataset Preparation

URL Dataset Preparation is the step where we decide which web pages will be crawled and analyzed by the distributed web crawler.

Input dataset for crawling

Each URL becomes one parallel task

3: Fetch Function Development

The **fetch()** function retrieves raw HTML content from each URL:

- Sends HTTP GET request
- Handles network delays using timeout
- Ensures pipeline continuity by returning empty text on failure

4: HTML Parsing & Text Extraction

The **parse()** function converts HTML into clean text:

Purpose:

- ❖ Removes HTML tags, scripts, CSS
- ❖ Extracts readable plain text
- ❖ Converts text into a list of lowercase words (tokens)
- ❖ This prepares the data for frequency computation.

5: Sequential Crawling

A sequential crawler was developed to measure performance before applying parallelism:

- Establishes baseline performance
- Demonstrates inefficiency of one-by-one crawling

6: Distributed Crawling Using Dask Bag

Dask Bag was used to execute crawling tasks in parallel:

- **from_sequence(urls)**
- **map(fetch)**
- **map(parse)**
- **flatten()**
- **frequencies()**
- **compute()**

7: Visualization

Matplotlib

matplotlib

Creates Bar Charts for frequency comparison.

WordCloud

WordCloud

Generates visual summaries of most repeated words.

8: Performance Evaluation

Two types of measurements were taken:

Execution Time

- Sequential crawling time
- Dask crawling time
- Speedup achieved (Parallel / Sequential)

Frequency Comparison

- Top words extracted in each mode
- Consistency of parsing results

8: Performance Evaluation

N = number of URLs
C = number of CPU cores

Sequential Time Complexity

$T = O(N \times fetch\_time \times parse\_time)$

Parallel Time Complexity

$T = O((N / C) \times fetch\_time)$

This demonstrates theoretical speedup under parallel execution.

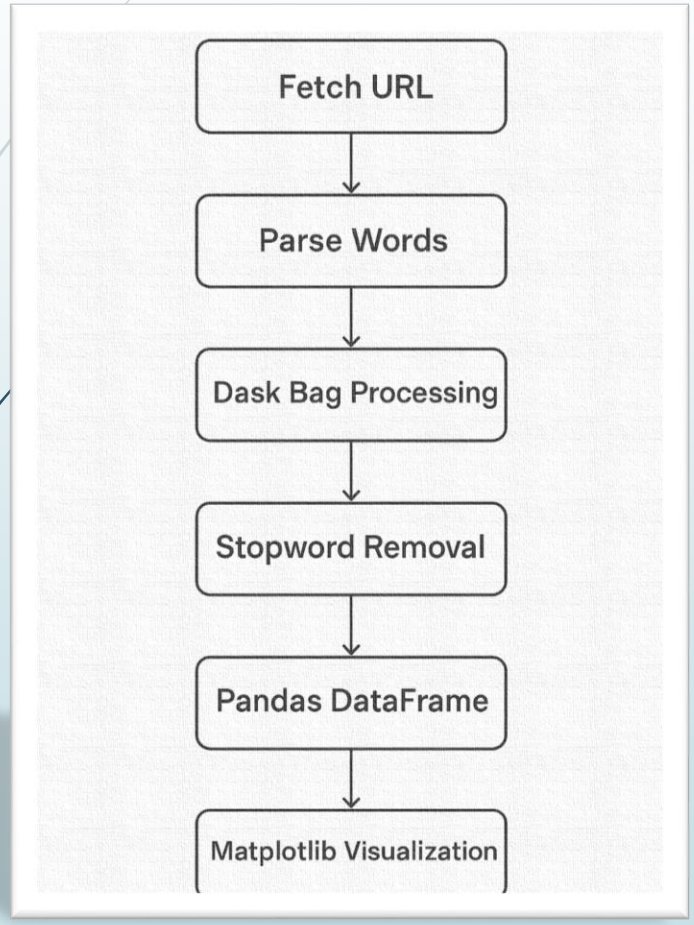
Presented by: Ommamah Latif (032)

Error	Meaning
403	site blocked your request
429	too many requests
requests.exceptions.ConnectionError	internet / DNS issue

Fetching a webpage

Apply a header to resolve 403 error

Steps

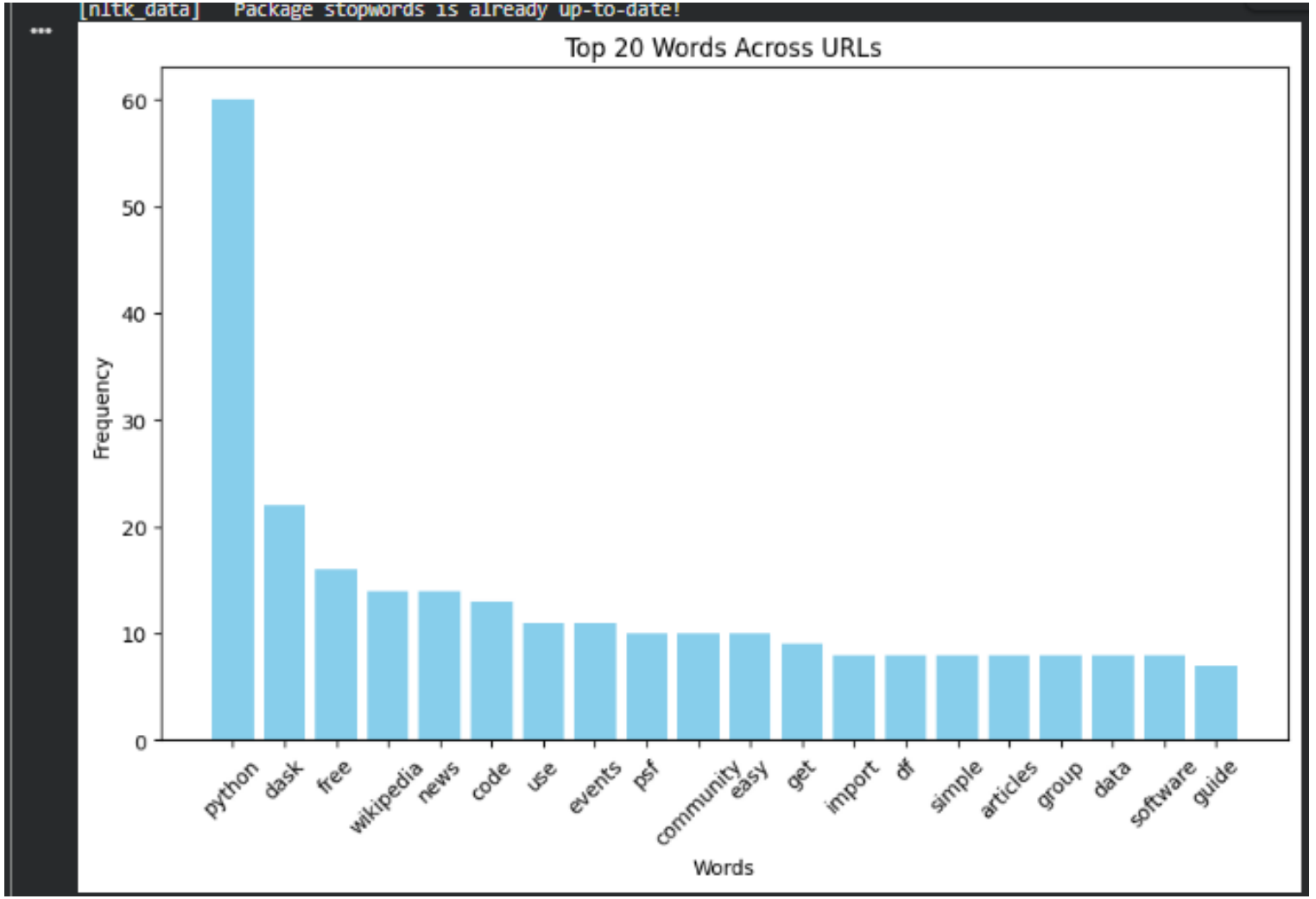


fetch_page(url)

- Sends HTTP GET request with a browser-like User-Agent.
- Uses `timeout=5` to prevent freezing.
- Returns HTML only if status code = 200, otherwise returns empty string.

Dask Bag Processing

- `from_sequence(urls, npartitions=10)` creates parallel URL partitions.
- `map(fetch_page)` fetches pages concurrently.
- `map(parse_words).flatten()` extracts & merges words from all pages.
- `frequencies().compute()` counts all word occurrences in parallel.


[nltk_data] Package stopwords is already up-to-date!
Top 20 Words Across URLs
Frequency
0
10
20
30
40
50
60
python
dask
free
wikipedia
news
code
use
events
psf
community
easy
get
import
df
simple
articles
group
data
software
guide
Words

Sequential crawling Vs Dask Parallel

Feature	Sequential Crawling	Dask Parallel Crawling
Execution Mode	One URL at a time	Multiple URLs in parallel
Speed	Slower	Faster
Resource Utilization	Low	High (CPU + network)
Scalability	Limited	High (multi-core / multi-node)
Fault Tolerance	Simple	Automatic retries, distributed safe
Complexity	Simple	More setup required (Dask client, workers)
Best Use	Small-scale or testing	Large-scale crawling / scraping

Sequential fetching

```
[41]
✓ 0s
# Sequential fetching and parsing
start_seq = time.time()

all_words_seq = []

for url in urls:
    html = fetch_page(url)
    words = parse_words(html)
    all_words_seq.extend(words)

# Count word frequencies
from collections import Counter
word_counts_seq = Counter([w for w in all_words_seq if w not in stop_words])

end_seq = time.time()
print(f"Sequential crawling time: {end_seq - start_seq:.2f} seconds")

...  Sequential crawling time: 0.82 seconds
```

Parallel fetching

```
[42]
✓ 0s
import dask.bag as db

# Start timer
import time
start_dask = time.time()

# Create Dask Bag
bag = db.from_sequence(urls)
html_pages = bag.map(fetch_page)
words_bag = html_pages.map(parse_words).flatten()

# Count word frequencies
word_counts_dask = dict(words_bag.frequencies().compute())

# Remove stopwords (if used)
word_counts_dask = {w: c for w, c in word_counts_dask.items() if w not in stop_words}

end_dask = time.time()
print(f"Dask parallel crawling time: {end_dask - start_dask:.2f} seconds")

...  Dask parallel crawling time: 0.63 seconds
```

Word Analysis

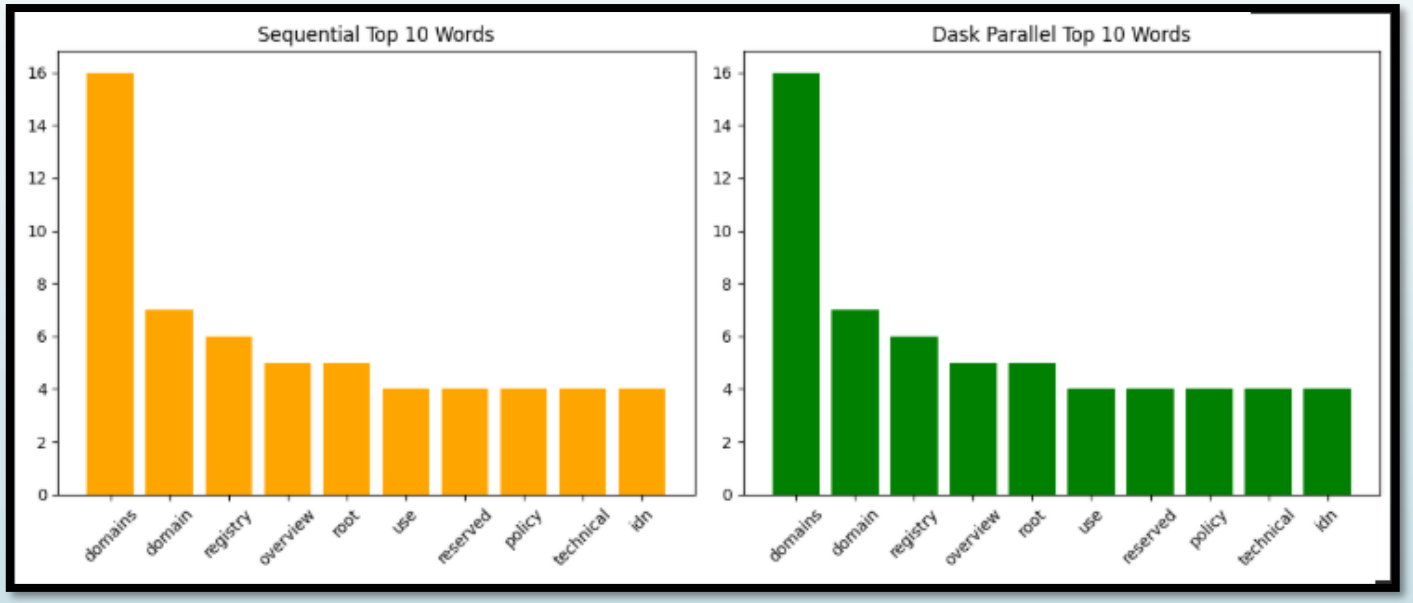
Data Preparation
- Extracted top-frequency words into DataFrames (df_top20, df_seq_top10, df_dask_top10)
- Sorted word counts for clear visualization

Top 20 Words Bar Chart (ax1)
- Plots vertical bar chart of top 20 Dask-crawled words
- Adds title, axis labels, rotated x-labels for readability

Sequential vs Dask Comparison (ax2, a
- ax2: Top 10 sequential words (orange)
- ax3: Top 10 Dask words (green)

Word Cloud (ax4)
- Generates word cloud using Dask word frequencies
- Displays with imshow and hides axes for a clean look

Layout & Presentation
- Uses subplot2grid for structured 3x2 layout
- coispan=2 for wide top + bottom



Challenges Faced

Presented by: Sania Arif (038)

Parallel URL Handling

Managing multiple URLs concurrently during parallel execution presented a significant challenge.

HTML Text Extraction

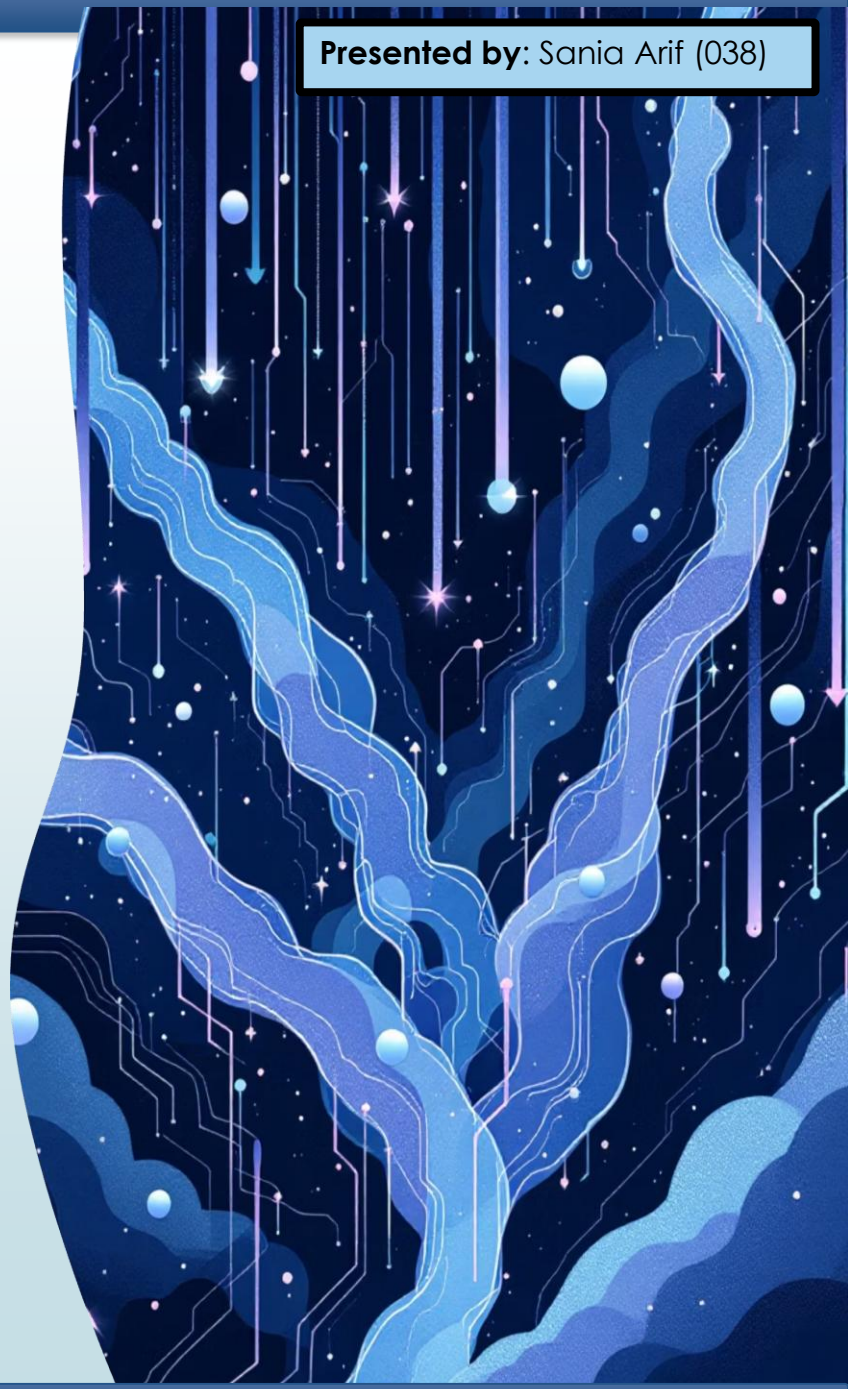
Extracting useful text from complex HTML pages required robust cleaning techniques.

Broken Link Management

Effectively handling broken or unreachable links was crucial for crawler stability.

Dask Bag & Lazy Execution

Grasping Dask Bag and its lazy execution model (compute) was key to optimising performance.



Project Conclusion

01

Distributed Crawler Implemented

Successfully built a distributed web crawler utilising Dask for enhanced performance.

02

Reduced Execution Time

Parallel crawling significantly decreased overall execution time, boosting efficiency.

03

Efficient Text Extraction

Achieved effective text extraction and precise word-frequency analysis from web content.

04

Practical Parallel Computing

Demonstrated the practical application and benefits of parallel computing in real-world scenarios.

05

Dask's Effectiveness Proven

Confirmed Dask's effectiveness for complex web data processing tasks.

Future Work

- Run crawler on **multi-node distributed clusters**
- Add **link-following (deep crawling)**
- Apply **NLP-based sentiment analysis**
- Store results in **MongoDB database**
- Develop a **real-time crawling dashboard**


Thank
You