IR U5

1. Parallel Query Processing

Parallel Query Processing is a technique in Information Retrieval (IR) where **multiple processors or machines work simultaneously** to process user queries. This is essential because modern search engines must handle **billions of documents** and **millions of queries per second**.

Parallel processing improves **speed, scalability, throughput, and system performance**.

2. Need for Parallel Query Processing

- The web contains extremely large document collections.
- Single-processor indexing/search is too slow.
- Multiple queries arrive simultaneously.
- Real-time retrieval demands fast response times.
- Large-scale distributed search engines require parallel architecture.

3. Models of Parallel Query Processing

(a) Document Partitioning

- The document collection is split across machines.
- Each machine builds its own **local inverted index**.
- A user query is sent to all machines.
- Each machine returns its result, and the system merges them.
  Advantages: Simple, easy scaling
  Limitation: Merging results may be slow

(b) Term Partitioning

- The index is partitioned by **terms**, not documents.
- Machine 1 stores terms A–G
- Machine 2 stores terms H–P, etc.
- Query is split based on the terms it contains.
  Advantage: Compact index per machine
  Limitation: Query must communicate with many nodes

(c) Hybrid Partitioning

- Combination of document and term partitioning.
- Balances load between machines.
- Used in large-scale search engines.

4. Techniques in Parallel Query Processing

- **Load Balancing** → ensures equal workload distribution
- **Distributed Hashing** → maps documents/terms to specific nodes
- **Result Merging** → combines ranked lists from all machines
- **Caching** → stores common queries for faster reuse
- **Fault Tolerance** → continues processing even if a node fails

5. Advantages

- High-speed query processing
- Handles massive data efficiently
- Scales horizontally (add more machines)
- Improves throughput for multiple simultaneous users
- Reduces latency during heavy search loads

6. Challenges

- Synchronization between distributed machines
- Network overhead when merging results
- Fault tolerance complexity
- Data replication cost
- Uneven distribution in partitioning

2. MapReduce in Information Retrieval

MapReduce is a **distributed programming model** used for processing large-scale data across clusters. Introduced by Google, it simplifies tasks such as **index building, web mining, query analysis, and log processing** in IR.

It follows two major phases: **Map** and **Reduce**, which automatically manage parallel execution, fault tolerance, and scalability.

2. Why MapReduce for IR?

- IR requires processing **massive text datasets**.
- Building inverted indexes manually is slow.
- Parallel clusters can process large amounts of web data.
- MapReduce automatically handles data distribution.

3. Working of MapReduce

(a) Map Phase

- Input text documents are divided into blocks.
- Each mapper processes one block.
- Outputs **key-value pairs** such as: **(term, documentID)**
  Example:
  Document: "data mining techniques"
  Mapper output:

- (data, doc1)
- (mining, doc1)
- (techniques, doc1)

(b) Shuffle & Sort Phase

- Intermediate results are grouped by key (term).
- All entries with the same term are collected together.
- Automatically handled by the framework.

(c) Reduce Phase

- Reducer receives all values for a term.
- Builds **posting lists** or inverted index entries: Example:
  mining → [doc1, doc5, doc20]

4. Applications of MapReduce in IR

- **Inverted Index Construction**
  - Core component of a search engine
- **Query Log Analysis**
  - Understanding user behavior
- **Web Mining**
  - Extracting patterns, keywords
- **PageRank Computation**
  - Uses MapReduce for large-scale link analysis
- **Spam Detection**
  - Identifying low-quality or duplicate pages
- **Text Classification & Clustering**

5. Advantages of MapReduce

- **Scalable:** works on thousands of nodes
- **Fault Tolerant:** automatically restarts failed tasks
- **Parallel Execution:** tasks distributed to multiple machines
- **Simple Programming Model:** only Map() and Reduce() functions
- **Handles Big Data Easily:** ideal for large IR tasks

6. Limitations

- High I/O overhead due to disk operations
- Not suitable for real-time search
- Slow for small datasets
- Multiple MapReduce stages may be needed for complex tasks

7. Tools & Frameworks

- Hadoop MapReduce
- Apache Spark (MapReduce-like faster system)
- Google Cloud Dataflow
- Amazon EMR

Web Crawler

A **Web Crawler** (also called Spider/Robot) is an automated program that systematically browses the World Wide Web to download and collect web pages. Crawlers form the foundation of search engines, data mining systems, and many online applications. They fetch pages, extract links, and continuously explore the web.

Web Crawler Structure

A Web Crawler typically has several key components that work together:

(a) Seed URLs

- Crawling begins from a set of starting URLs.
- These links act as the entry point to the web.

(b) URL Frontier (Queue System)

- A data structure (queue or priority queue) storing URLs to visit next.
- Maintains crawl order such as breadth-first, depth-first, or priority-based crawling.

(c) Fetching Module

- Sends HTTP requests to fetch HTML pages from servers.
- Follows politeness policies and robots.txt rules.

(d) Parser

- Extracts text content, metadata, and links from HTML.
- Uses HTML parsing libraries to identify hyperlinks.

(e) Duplicate & Loop Detection

- Prevents revisiting the same page multiple times.
- Uses hashing or checksums to detect duplicate URLs/content.

(f) Storage Component

- Stores the fetched pages, parsed data, or extracted information.
- Data may be saved in databases or indexing systems.

Web Crawler Libraries

Web Crawler libraries provide pre-built tools and frameworks that simplify the development of web crawlers. Instead of manually writing code for fetching pages, parsing HTML, and handling links, these libraries offer ready-made modules for efficient and scalable crawling. They support tasks like HTML parsing, link extraction, scheduling, request handling, and working with dynamic content.

2. Major Web Crawler Libraries

(a) Scrapy

Description

Scrapy is a **powerful, fast, and scalable Python framework** designed specifically for web crawling and scraping. It is used by professionals to build large-scale crawlers.

Key Features

- **Asynchronous architecture** → faster crawling
- **Spiders** → custom classes to define crawl behavior
- **Item Pipelines** → clean, validate, and store extracted data
- **Built-in link extractor** for discovering URLs
- **Auto-throttling** and **politeness policies**

Benefits

- Extremely fast
- Modular and easy to extend
- Suitable for crawling large websites or millions of pages

Use Cases

- E-commerce data extraction
- Price tracking
- News or blog crawling

(b) Beautiful Soup

Description

Beautiful Soup is a Python library designed for **HTML and XML parsing**. It is commonly used for small to medium crawling tasks.

Key Features

- Parses poorly formed HTML gracefully
- Provides tree-based navigation (find, find_all)
- Works well with **Requests** to fetch pages
- Extracts specific elements like tables, headings, links

Benefits

- Very easy to use
- Great for scraping simple static websites
- Supports multiple parsers such as html.parser, lxml

Limitations

- Not suitable for large-scale crawling
- Cannot execute JavaScript

(c) Requests

Description

Requests is a simple and elegant Python library used for **sending HTTP requests**.

Key Features

- Supports GET, POST, PUT, DELETE
- Handles cookies, headers, authentication
- Integrates easily with Beautiful Soup for parsing
- Easy error handling and redirection support

Benefits

- Human-readable API
- Lightweight and highly reliable
- Ideal for downloading web pages or interacting with APIs

Use Case

- Used as the base module for many small web crawlers

(d) Selenium

Description

Selenium is a browser automation tool that can control a real browser (Chrome, Firefox, Edge). It is used for **JavaScript-heavy or dynamic websites**.

Key Features

- Executes JavaScript
- Interacts with web elements (click, fill forms, scroll)
- Supports full browser rendering
- Handles login-based or infinite scrolling pages

Benefits

- Extracts content that normal crawlers cannot see
- Ideal for testing and scraping dynamic web apps

Limitations

- Slower than Scrapy or Requests
- Requires more resources (browser automation)

1. Python Scrapy

Scrapy is a **powerful, open-source Python framework** designed for **web crawling and web scraping**. Unlike simple libraries, Scrapy provides a full architecture for large-scale crawlers. It is widely used to extract structured data from websites, especially when high speed and scalability are required.

2. Key Components of Scrapy

(a) Spiders

- Core component that defines *how* a website will be scraped.
- Contains start URLs and parsing logic.

(b) Engine

- Controls the entire data flow between different components.

(c) Scheduler

- Maintains the queue of pending URLs.
- Decides which request to process next.

(d) Downloader

- Responsible for sending HTTP requests and receiving responses.

(e) Item Pipeline

- Cleans, processes, and stores extracted data.

3. How Scrapy Works (Flow)

1. Spider sends initial request to start URLs.
2. Downloader fetches page and sends HTML back to Spider.
3. Spider parses content and extracts required data.
4. Extracted links are added to Scheduler for further crawling.
5. Cleaned items are passed through Item Pipelines and stored.

4. Features of Scrapy

- **Asynchronous Crawling:** High-speed and efficient.
- **Auto-Throttle:** Prevents overloading web servers.
- **Built-in Link Extractors:** Automatically follows hyperlinks.
- **Middleware Support:** Allows custom request/response handling.
- **Exporters:** Saves data as JSON, CSV, XML, DB, etc.

5. Advantages of Scrapy

- Very **fast** due to asynchronous architecture.
- Highly **scalable** for million-page crawls.
- Easy to extend and customize.
- Supports logging, caching, and error handling.
- Suitable for real-world industrial crawling.

6. Applications of Scrapy

- E-commerce price tracking
- News aggregation
- Blog crawling
- Market research & competitor analysis
- Scraping large datasets for machine learning

2. Beautiful Soup

1. Introduction

Beautiful Soup is a **Python library for parsing HTML and XML documents**. It is not a crawler by itself, but is mainly used for **extracting information from downloaded web pages**. It works well with Requests library for small-scale scraping tasks.

2. Features of Beautiful Soup

(a) Easy HTML Parsing

- Handles badly formatted HTML gracefully.
- Finds elements using tags, attributes, CSS selectors.

(b) Supports Multiple Parsers

- html.parser
- lxml
- html5lib

(c) Tree-Based Navigation

- Methods like .find(), .find_all(), .select()
- Easy to navigate parent, child, and sibling elements.

3. How Beautiful Soup Works

1. Fetch the web page using **Requests**.
2. Load HTML content into BeautifulSoup object.

3. Parse required elements (titles, links, paragraphs, tables).
4. Clean and store extracted data.

4. Advantages of Beautiful Soup

- Very simple and beginner-friendly.
- Best for **small-scale or single-page scraping**.
- Handles incorrect HTML structures.
- Works well for text extraction and data cleaning.

5. Limitations

- Not suitable for large crawls (slow).
- Cannot execute JavaScript (unlike Selenium).
- Needs to be combined with Requests for full crawling functionality.

6. Applications of Beautiful Soup

- Extracting article content from blogs
- Scraping static product pages
- Extracting tables & HTML forms
- Data cleaning for research
- Academic and project-based scraping tasks