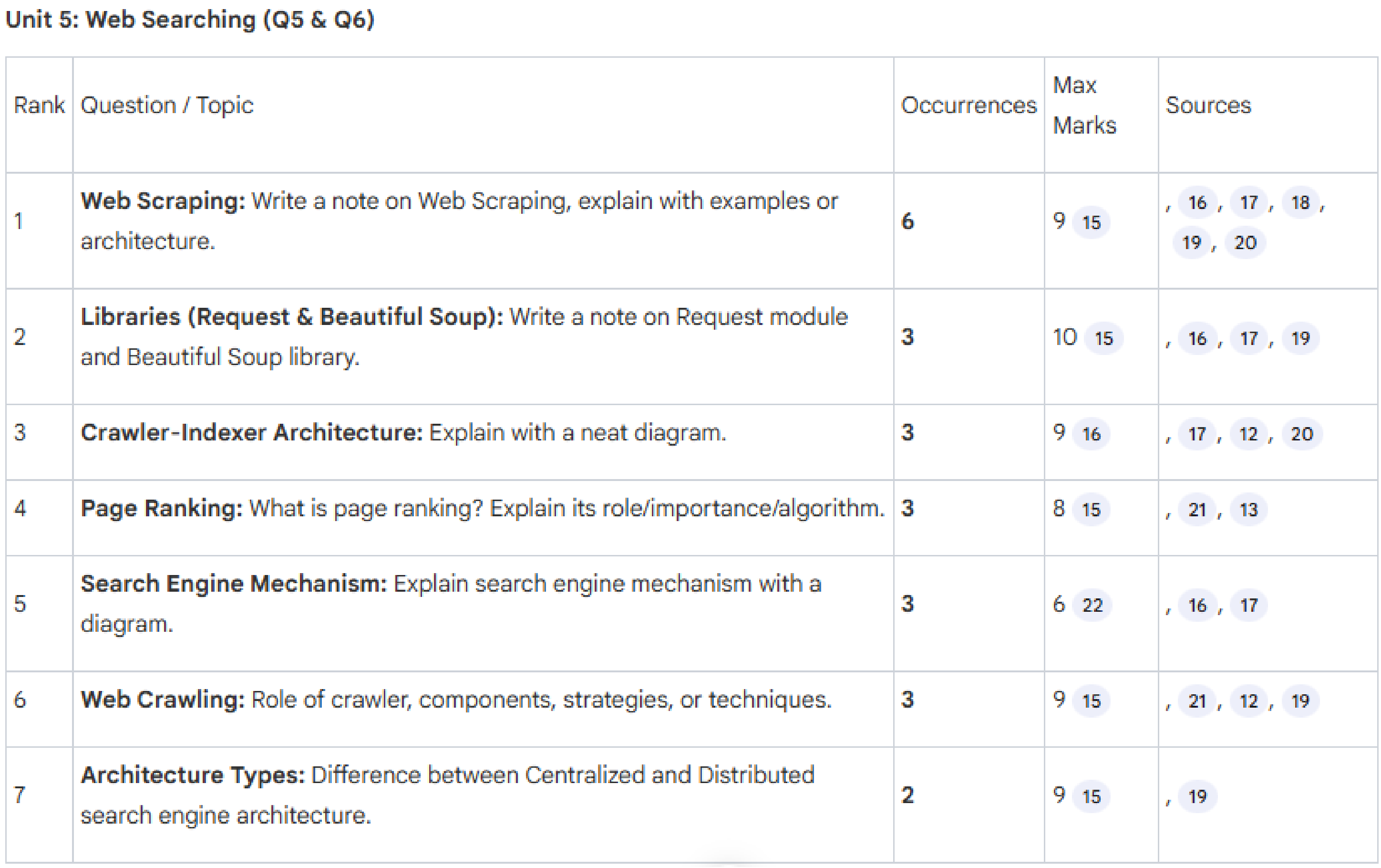


Unit 5: Web Searching (Q5 & Q6)

Rank	Question / Topic	Occurrences	Max Marks	Sources
1	**Web Scraping:** Write a note on Web Scraping, explain with examples or architecture.	**6**	9 15	, 16 , 17 , 18 , 19 , 20
2	**Libraries (Request & Beautiful Soup):** Write a note on Request module and Beautiful Soup library.	**3**	10 15	, 16 , 17 , 19
3	**Crawler-Indexer Architecture:** Explain with a neat diagram.	**3**	9 16	, 17 , 12 , 20
4	**Page Ranking:** What is page ranking? Explain its role/importance/algorithm.	**3**	8 15	, 21 , 13
5	**Search Engine Mechanism:** Explain search engine mechanism with a diagram.	**3**	6 22	, 16 , 17
6	**Web Crawling:** Role of crawler, components, strategies, or techniques.	**3**	9 15	, 21 , 12 , 19
7	**Architecture Types:** Difference between Centralized and Distributed search engine architecture.	**2**	9 15	, 19

Q1) Write a note on Web Scraping, explain with examples or architecture.

Web scraping (also known as screen scraping, data mining, web harvesting, or web data extraction) is the process of automatically fetching and extracting arbitrary data from websites.

- **The Concept:** While a web crawler downloads pages for search engine indexing, web scraping focuses on **extracting specific data** (e.g., product prices, stock trends) and transforming it into a structured format (like a spreadsheet, JSON, or database) for analysis.
- **How it works:** It involves an automated program that queries a web server, requests content (usually HTML), and then parses that content to extract specific information, ignoring the visual formatting.

2. Architecture of Web Scraping
The architecture of a web scraping system typically follows a three-step pipeline to convert unstructured web data into structured data.

Step 1: Data Collection (Fetching)

- **Action:** The system (or "scraper") connects to the target websites using protocols like HTTP.
- **Mechanism:** It sends a request to the URL (similar to how a browser does) and downloads the raw HTML code of the webpage.
- **Tools:** In Python, libraries like the requests module are commonly used to send these HTTP requests.

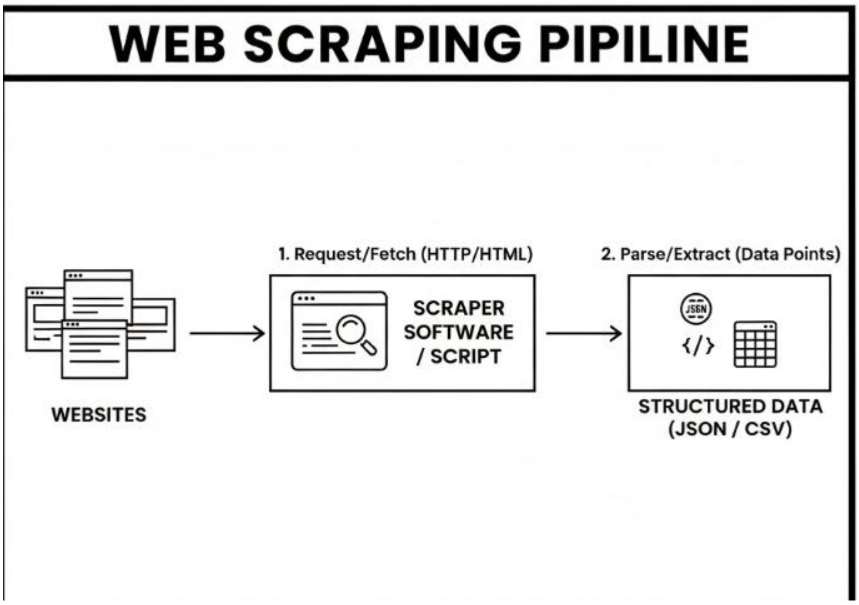
Step 2: Data Parsing and Transformation

- **Action:** This is the core processing step. The system analyzes the downloaded raw HTML.
- **Mechanism:** It navigates the HTML structure (DOM tree) to locate specific elements (like <div class="price"> or <table>).
- **Transformation:** It extracts the desired text or values and cleans them (e.g., removing currency symbols, correcting dates).

- **Tools:** Libraries like **Beautiful Soup** are used to parse HTML and navigate the tree structure to find data.

Step 3: Data Storage

- **Action:** The extracted and cleaned data is saved for future use.
- **Formats:** The data is usually exported into a structured format such as **CSV, XML, JSON**, or stored directly into a database (SQL/NoSQL).


WEB SCRAPING PIPILINE
1. Request/Fetch (HTTP/HTML)
2. Parse/Extract (Data Points)
SCRAPER SOFTWARE / SCRIPT
JSON
</>
WEBSITES
STRUCTURED DATA (JSON / CSV)

3. Workflow: Steps to Perform Web Scraping

To implement a scraping task, the process generally involves these practical steps:

1. **Find the URL:** Identify the specific webpage that contains the data you want.
2. **Inspect the Page:** Use browser "Developer Tools" to inspect the HTML structure and find the unique tags or IDs associated with the target data.
3. **Write the Code:** Create a script (e.g., using Python) to request the page and extract the information.
4. **Store the Data:** Save the output to a file or database.

4. Examples and Applications

Web scraping is widely used in various industries for data-driven decision-making. Common examples include:

- **Pricing Intelligence:** E-commerce companies scrape competitor websites to monitor product prices and adjust their own pricing strategies dynamically.
- **Financial Planning:** Extracting stock prices, currency exchange rates, or financial reports from news sites for analysis.
- **Sentiment Analysis:** Scraping reviews from social media or product pages to analyze consumer sentiment and brand reputation.
- **Lead Generation:** Collecting contact details (emails, phone numbers) from business directories.

5. Challenges in Web Scraping

While powerful, web scraping faces several technical hurdles:

Pattern Changes: If a website updates its design (changing HTML tags/IDs), the scraper script may break and fail to find the data.

Anti-Scraping Technologies: Websites use "Honeypot traps" (invisible links that only bots follow) or CAPTCHAs to detect and block scrapers.

Dynamic Content: Many modern sites load data using JavaScript *after* the initial page load, which simple HTTP requests cannot fetch (requiring tools like Selenium).

Q2) Write a note on Request module and Beautiful Soup library.

1. The Requests Module

Definition: The Requests library is described as the standard for making HTTP requests in Python. It is designed to be "simple and human-friendly," abstracting the complexities of making requests behind a simple API so that users can focus on interacting with services rather than configuring connection details.

Key Features:
Ease of Use: It allows users to send HTTP/1.1 requests extremely easily. There is no need to manually add query strings to URLs or form-encode POST data; the library handles this automatically.
HTTP Methods: It supports all major HTTP methods such as GET, POST, PUT, DELETE, HEAD, and OPTIONS.
Response Handling: When a request is made, the library returns a **Response Object** containing all the information returned by the server (content, encoding, status, headers).

Basic Usage (from Sources): To use it, one must first install it and import it. The most common method used is get() to fetch a webpage.

```
import requests


# Making a GET request
r = requests.get('https://www.google.com')


# Checking the status code
print(r.status_code)


# Accessing the content
print(r.content)
```

2. Beautiful Soup Library

Definition: Beautiful Soup is a Python library designed for pulling data out of **HTML and XML files**,. It functions by creating a parse tree from the page source code, which allows the programmer to extract data in a hierarchical and readable manner.

Key Features:
Navigation and Search: It provides "Pythonic idioms" for iterating, searching, and modifying the parse tree. This capability saves programmers hours of work compared to writing custom parsing logic.
Parser Integration: It sits on top of an HTML or XML parser (like lxml or Python's built-in html.parser) to do the heavy lifting of understanding the document structure.
Handling "Tag Soup": It is robust and designed to handle messy or poorly formatted web code (often called "tag soup"), cleaning it up into a valid tree structure.

Basic Usage (from Sources): To use it, the library (typically imported as bs4) is initialized with the HTML string fetched by the Requests module.

```
from bs4 import BeautifulSoup


# Create the Soup object (Parse the HTML)
# 'html.parser' is the parser used
soup = BeautifulSoup(html_doc, 'html.parser')


# Extract data (e.g., the title of the page)
print(soup.title)


# Find specific elements (e.g., all paragraph tags)
soup.find_all('p')
```

3. How They Work Together

In the architecture of a web scraping system, these two libraries perform distinct but complementary steps:

1. **Data Collection (Requests):** The **Requests** module is used to **fetch** the raw HTML code from a target URL, effectively acting like a web browser downloading a page.
2. **Data Parsing (Beautiful Soup):** The **Beautiful Soup** library is then used to **parse** that raw HTML code to navigate the document tree and **extract** specific data points (like prices, titles, or links) required by the user.

----

q3)**Crawler-Indexer Architecture**

The centralized architecture creates a central repository of the web. It is the standard "Crawler -Indexer" model used by early generations of search engines. In this model, the search engine operations (crawling, indexing, and query processing) are performed at a single logical site,.

The Crawler-Indexer architecture represents the fundamental structure of a centralized search engine. It is designed to handle the massive scale of the Web by dividing the system into two main functional parts:

1. **Data Acquisition (Backend):** Consisting of the **Crawler** and **Indexer**, which gathers and organizes information from the World Wide Web.
2. **Information Retrieval (Frontend):** Consisting of the **Query Engine** and **User Interface**, which handles user requests and displays results.

wler architecture.

Fig. Q12.1 shows crawler architecture.

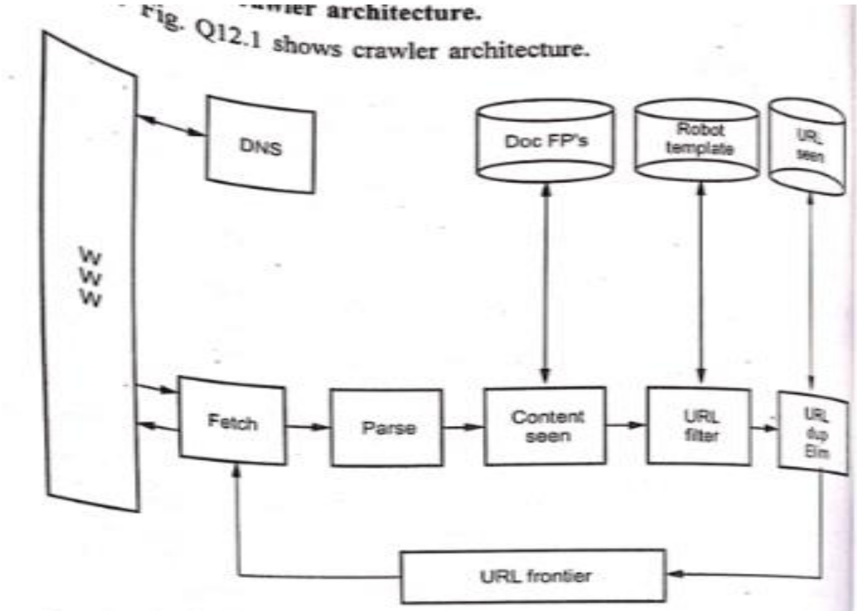


Fig. Q.12.1

2. Component Explanation

A. Crawler Module (Web Spider/Robot)

- **Function:** This is a software program that traverses the World Wide Web in a methodical, automated manner.
- **Process:** It starts with a list of seed URLs, downloads the web pages, identifies hyperlinks in the text, and adds them to the list of URLs to visit next.
- **Output:** It extracts the URL appearing in the retrieved pages and gives this information to the control module. It feeds the downloaded content to the **Indexer**.

B. Indexer Module

- **Function:** This module processes the raw data collected by the crawler to make it searchable.
- **Process:**
  - It separates the content from the HTML structure.
  - It performs text analysis, such as removing **stop words** (common words like "the", "is") and stemming (reducing words to their root).
  - It creates the **Inverted Index**, a data structure that maps keywords to the documents (URLs) where they appear.
- **Text Structure Utility:** The indexer often relies on utility functions to understand the structure of the text (e.g., identifying titles vs. body text) to assign proper weights to terms.

C. The Index (Repository)

- **Function:** This is the central storage of the search engine.
- **Content:** It stores the compressed inverted files, document metadata, and sometimes a compressed version of the document itself.
- **Role:** The query engine searches this index rather than the live web, ensuring fast response times.

D. Query Engine (Query Processor & Ranking)

- **Function:** This component bridges the user and the database.
- **Query Processing:** It parses the user's query (checking for spelling, expanding query terms).
- **Ranking:** The ranking module sorts the retrieved results based on relevance (e.g., using **PageRank** or term frequency). It ensures that the most relevant ("top") results are presented first to the user.

E. Interface Module (User Interface)

- **Function:** Allows users to interact with the system.
- **Operation:** It accepts the user's query string and displays the ranked search results returned by the Query Engine.

Summary of Workflow

1. **Crawlers** browse the **WWW** and download pages.
2. **Indexer** processes these pages and saves them into the **Index**.
3. **Users** submit queries via the **Interface**.
4. **Query Engine** searches the **Index**, ranks the results, and returns them to the user.

Q4) What is page ranking? Explain its role/importance/algorithm in web searching with algorithm

PageRank is a link analysis algorithm used by Google Search to rank web pages in their search engine results. It was named after Larry Page, one of Google's founders.

Concept: It evaluates the **"importance"** or **"authority"** of a webpage based on the number and quality of links pointing to it. The underlying assumption is that more important websites are likely to receive more links from other websites.

- **The "Vote" Analogy:** The algorithm treats a link from Page A to Page B as a "vote" for Page B. However, not all votes are equal. A vote (link) from a page that is itself highly ranked (important) counts more than a vote from a low-ranked page.

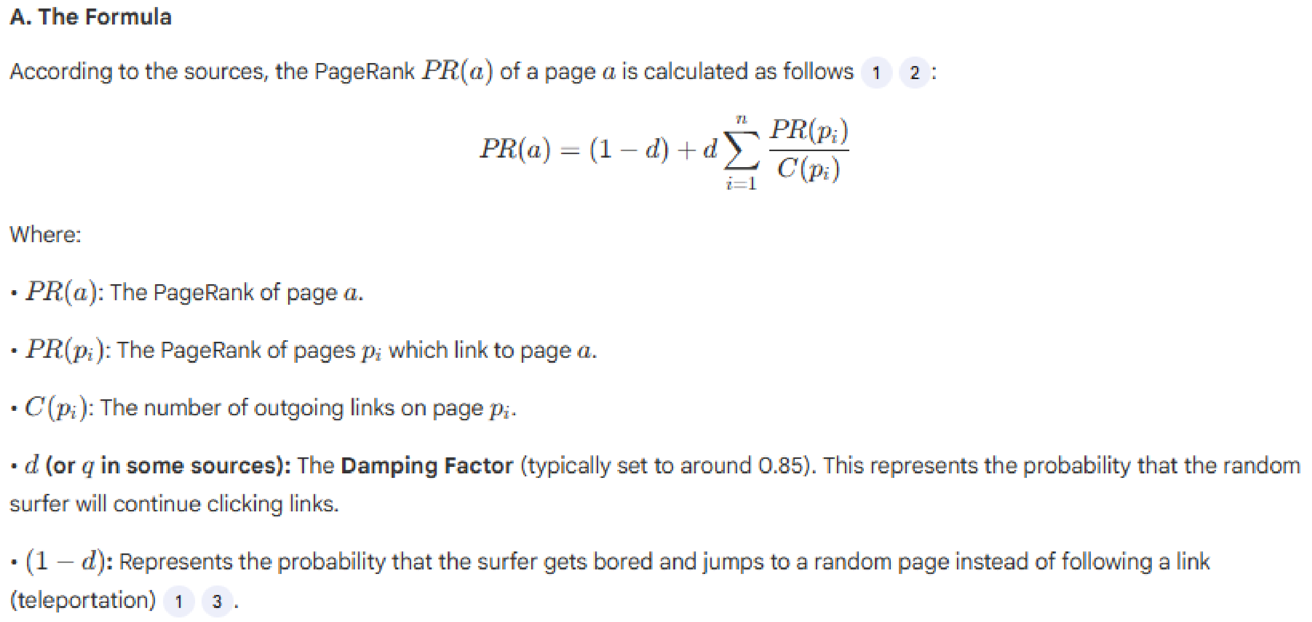
2. Role and Importance in Web Searching

PageRank plays a critical role in filtering the massive amount of information on the Web to provide high-quality results:

1. **relevance vs. Authority:** While traditional text analysis determines which pages are *relevant* to a query (contain the keywords), PageRank determines which of those pages are *authoritative* (trusted by others). It pushes the most authoritative pages to the top of the Search Engine Results Pages (SERPs).
2. **Quality Control:** It helps filter out spam or low-quality pages. A page might be stuffed with keywords to appear relevant, but if no reputable sites link to it, it will have a low PageRank.
3. **The Random Surfer Model:** PageRank is based on the behavior of a hypothetical "random surfer" who clicks on links at random. The PageRank of a page represents the probability (between 0 and 1) that this random surfer will land on that page. A probability of 0.5 implies a 50% chance of landing on that document.

3. The PageRank Algorithm

The PageRank algorithm is **recursive** because the rank of a page depends on the ranks of the pages linking to it.



A. The Formula

According to the sources, the PageRank $PR(a)$ of a page a is calculated as follows [1] [2]:

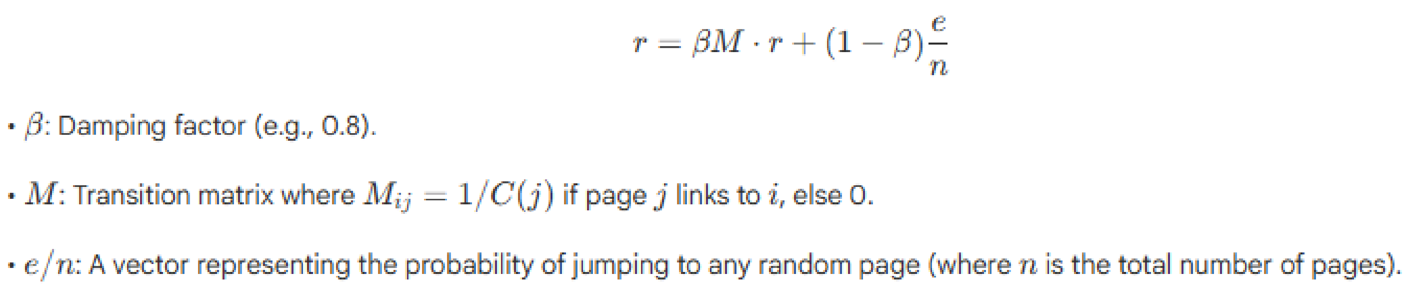
$$PR(a) = (1-d) + d\sum_{i=1}^{n} \frac{PR(p_i)}{C(p_i)}$$

Where:

- $PR(a)$: The PageRank of page a.
- $PR(p_i)$: The PageRank of pages p_i which link to page a.
- $C(p_i)$: The number of outgoing links on page p_i.
- d **(or q in some sources):** The **Damping Factor** (typically set to around 0.85). This represents the probability that the random surfer will continue clicking links.
- $(1-d)$**:** Represents the probability that the surfer gets bored and jumps to a random page instead of following a link (teleportation) [1] [3].

B. Matrix Representation (Iterative Calculation)

Since the web is a graph, PageRank can also be computed using matrices. If r is the PageRank vector and M is the transition matrix of the web graph, the calculation is updated iteratively:



$$r = \beta M \cdot r + (1-\beta)\frac{e}{n}$$

- β: Damping factor (e.g., 0.8).
- M: Transition matrix where $M_{ij} = 1/C(j)$ if page j links to i, else 0.
- e/n: A vector representing the probability of jumping to any random page (where n is the total number of pages).

C. Step-by-Step Execution

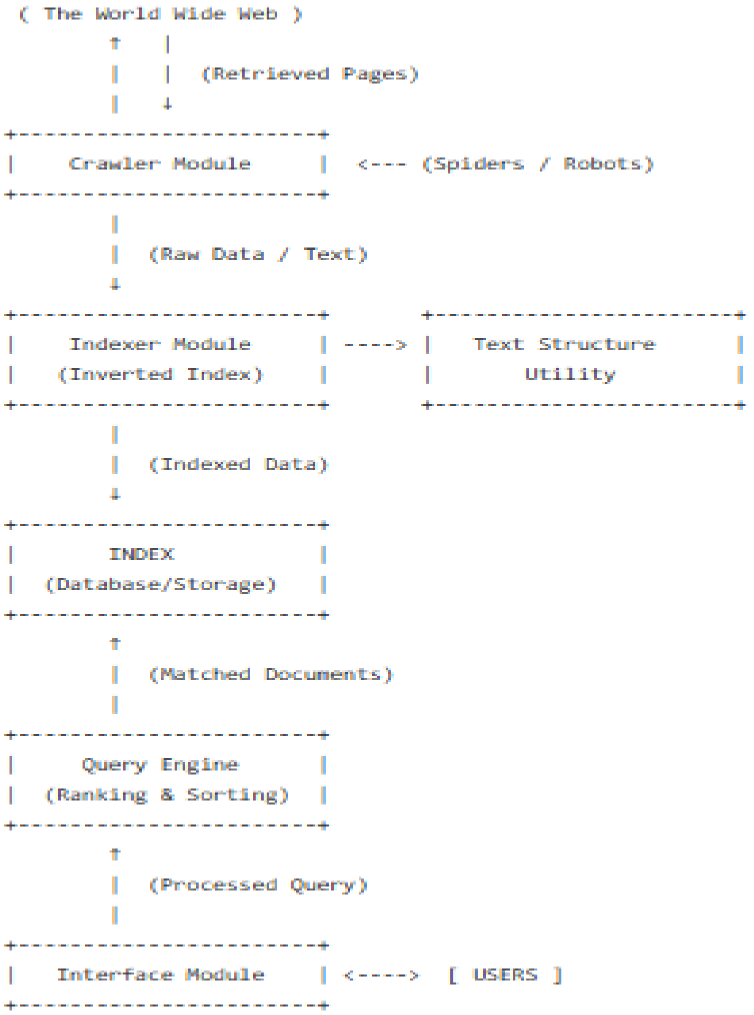
1. **Initialization:** Assign an initial rank (e.g., $1/n$) to every page in the network.
2. **Iteration:** For each page, calculate its new rank by summing the rank contributions from all pages that link to it. The contribution from a page P is its current rank divided by its number of outgoing links.
3. **Damping:** Apply the damping factor to model the random surfer stopping or jumping.
4. **Convergence:** Repeat the process until the PageRank values stabilize (change very little between iterations).

Summary Example

If Page A links to Page B and Page C, Page A passes half of its rank to B and half to C. If Page B has a high PageRank, its v ote gives a significant boost to the pages it links to. This cycle repeats until the system determines a steady "importance" scor e for every page.

---

q5) Explain search engine mechanism with a diagram.



```
  ( The World Wide Web )
          ↑    |
          |    |  (Retrieved Pages)
          |    ↓
+------------------------+
|     Crawler Module     |  <--- (Spiders / Robots)
+------------------------+
          |
          |  (Raw Data / Text)
          ↓
+------------------------+        +------------------------+
|     Indexer Module     | ----> |    Text Structure      |
|    (Inverted Index)    |        |        Utility         |
+------------------------+        +------------------------+
          |
          |  (Indexed Data)
          ↓
+------------------------+
|         INDEX          |
|   (Database/Storage)   |
+------------------------+
          ↑
          |  (Matched Documents)
          |
+------------------------+
|      Query Engine      |
|   (Ranking & Sorting)  |
+------------------------+
          ↑
          |  (Processed Query)
          |
+------------------------+
|    Interface Module    | <----> [ USERS ]
+------------------------+
```

The search engine mechanism operates through the interaction of the following key modules:

1. Crawler Module (Data Acquisition)

• **Function:** This module acts as the "hands" of the search engine. It consists of software programs known as **Spiders**, **Robots**, or **Crawlers**.

• **Mechanism:** It traverses the World Wide Web by starting with a list of "seed" URLs. It downloads web pages, parses them to extract hyperlinks, and adds those new links to its list to visit next.

• **Output:** It feeds the downloaded documents (raw HTML and text) to the Indexer Module.

2. Indexer Module (Data Organization)

• **Function:** This module acts as the "brain" that organizes the chaotic web data into a structured format that allows for fast searching.

• **Mechanism:**

◦ **Text Extraction:** It separates the actual text content from the HTML tags and formatting.

◦ **Text Analysis:** It uses a **Text Structure Utility** to analyze the text (e.g., removing stop words like "the" or "and", and performing stemming).

◦ **Inverted Index:** It builds a massive **Inverted Index** (similar to the index at the back of a book) which maps specific keywords to the documents (URLs) in which they appear.

3. The Index (Storage)

• **Function:** This is the central repository of the search engine.

• **Content:** It stores the compressed inverted files, document metadata, and often a compressed version of the documents themselves. The Query Engine searches this repository, not the live Web, ensuring rapid response times.

4. Query Engine (Information Retrieval)

• **Function:** This component bridges the gap between the user's request and the stored data.

• **Query Processing:** It receives the user's search terms, parses them (checking spelling, expanding synonyms), and converts them into a format the system can understand.

• **Ranking:** The most critical part of this module is the **Ranking Algorithm**. It sorts the retrieved documents based on **relevance** (e.g., how often the keyword appears) and **importance** (e.g., **PageRank**, which evaluates the quality of links pointing to the page). This ensures the most useful results appear at the top.

5. Interface Module (User Interaction)

• **Function:** This is the "face" of the search engine.

• **Mechanism:** It provides an input box for users to type queries and displays the **Search Engine Results Page (SERP)** containing the ranked list of links, titles, and snippets returned by the Query Engine.

Q6) What is role of crawler in web searching? Explain the strategies used by the web crawler.
A **Web Crawler** (also known as a **Web Spider** or **Web Robot**) is a software program that browses the World Wide Web in a methodical, automated manner. Its primary function is to traverse the web to collect documents (such as HTML pages, PDFs, images) to build the database for a search engine.

1. Components of a Web Crawler

A web crawler (or spider) is a program that automatically traverses the web to download documents. According to the architecture described in **"ISR Decode"**, a crawler consists of the following key components that work together to manage the fetching process,:

1. **The URL Frontier:** This is a list (often a queue) containing all the URLs that are yet to be fetched in the current crawl. It manages the schedule of which pages to visit next.

2. **DNS Resolution Module:** Before a page can be fetched, its domain name (e.g., www.google.com) must be translated into an IP address. This module interacts with the Domain Name System (DNS) to determine the web server from which to fetch the page.

3. **Fetch Module:** This module uses the HTTP protocol to send a request to the web server and retrieve the actual web page content (HTML, PDF, etc.).

4. **Parsing Module:** Once a page is downloaded, this module analyzes the text. It extracts the content for indexing and extracts a set of **new links** (hyperlinks) found on that page to be added to the queue.

5. **Duplicate Elimination Module:** To avoid infinite loops or processing the same page twice, this module checks whether an extracted link is already in the URL Frontier or has recently been fetched.

Role of the Crawler in Web Searching

The crawler acts as the data acquisition component of a search engine. Its specific roles include:

- **Discovery and Retrieval:** The crawler starts with a list of URLs to visit (seeds). It visits these URLs, downloads the page content, and identifies all the hyperlinks contained within that page. These new links are added to the list of URLs to be visited next.
- **Feed the Indexer:** The main purpose of the crawler is to create a local copy of the visited pages. These downloaded pages are then passed to the **Indexer Module**, which processes the text to create the Inverted Index used for searching.
- **Maintenance (Freshness):** The web is dynamic; pages change frequently. The crawler must revisit pages periodically to update the search engine's database with the latest content or remove dead links.
- **Metadata Extraction:** Crawlers read specific tags, such as keyword Meta tags, to help the search engine determine what a site is about and how to index it.

Strategies Used by the Web Crawler

Since the web is vast and resources (bandwidth, storage, time) are limited, a crawler cannot visit every page instantly. It must use specific strategies (ordering policies) to decide which URL to visit next. According to the sources, the common strategie s include:

A. Breadth-First Search (BFS)

- **Strategy:** In this approach, the crawler visits all the direct neighbors (links) of the current page before moving deeper into the site structure.
- **Mechanism:** It visits pages level by level. If it starts at the Home Page (Level 1), it visits all links on the Home Page (Level 2) before clicking any links found on those Level 2 pages.
- **Usage:** This is a standard technique used to cover a wide variety of pages rather than getting stuck deep inside one specific website.

B. Depth-First Search (DFS)

- **Strategy:** The crawler follows a chain of links as far as possible before backtracking.
- **Mechanism:** It clicks the first link on a page, then the first link on *that* new page, and continues until it hits a dead end, then backtracks to the previous branch.
- **Usage:** This is less common for general web crawling because the crawler might get trapped in infinite loops (spider traps) or spend too much time on a single, unimportant path.

C. Best-First Search (Heuristic / Importance-Based)

- **Strategy:** The crawler prioritizes URLs based on their perceived "importance" or "quality" rather than just their position in the structure.
- **Mechanism:** It assigns a score to unvisited URLs based on heuristics.
  - **PageRank:** It may prioritize pages with a high PageRank (pages that many other pages link to).
  - **Hubs and Authorities:** It may look for "Hub" pages (which link to many good pages) or "Authority" pages (which contain valuable content).
- **Goal:** To ensure that the most valuable and authoritative content is indexed first.

D. Focused Crawling (Topic-Specific)

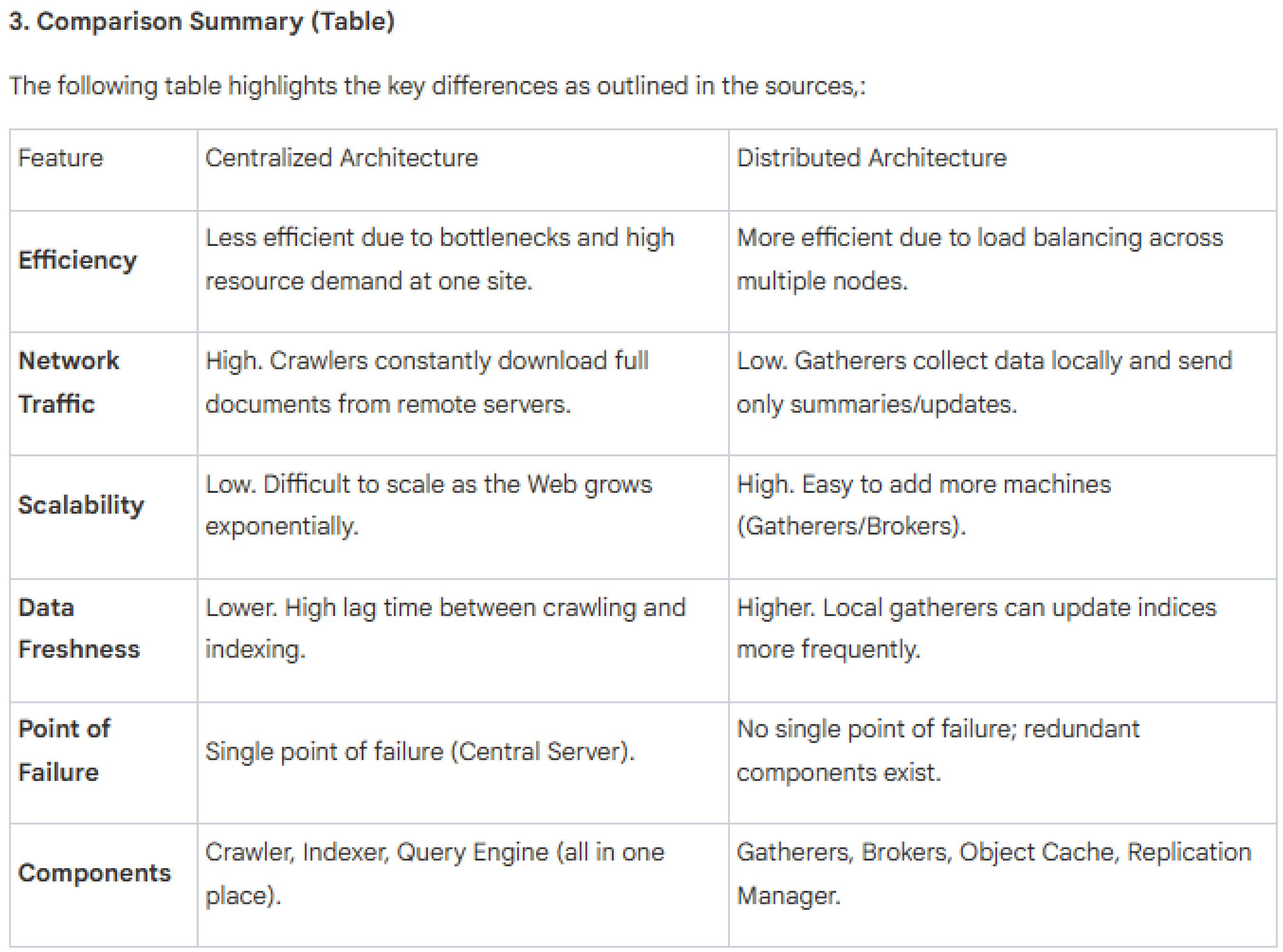
- While a general crawler gathers everything, a **Focused Crawler** attempts to download only pages that are relevant to a specific topic (e.g., "Health" or "Sports").
- It predicts the probability that a link will lead to a relevant page before visiting it, often by analyzing the anchor text o r the context of the link.

Summary of Workflow

1. **Select** a URL from the "Frontier" (list of unvisited URLs) based on the **Strategy** (e.g., Best-First).
2. **Fetch** (Download) the page via HTTP.
3. **Parse** the page to extract text and new links.
4. **Check** if the new links have already been visited.
5. **Add** new, unique links to the Frontier.

---

q7) Difference between Centralized and Distributed search engine architecture.



3. Comparison Summary (Table)

The following table highlights the key differences as outlined in the sources,:

Feature	Centralized Architecture	Distributed Architecture
Efficiency	Less efficient due to bottlenecks and high resource demand at one site.	More efficient due to load balancing across multiple nodes.
Network Traffic	High. Crawlers constantly download full documents from remote servers.	Low. Gatherers collect data locally and send only summaries/updates.
Scalability	Low. Difficult to scale as the Web grows exponentially.	High. Easy to add more machines (Gatherers/Brokers).
Data Freshness	Lower. High lag time between crawling and indexing.	Higher. Local gatherers can update indices more frequently.
Point of Failure	Single point of failure (Central Server).	No single point of failure; redundant components exist.
Components	Crawler, Indexer, Query Engine (all in one place).	Gatherers, Brokers, Object Cache, Replication Manager.

---

q8) Harvest Architecture

Harvest is a **distributed** crawler-indexer architecture designed to address the scalability and efficiency limitations of traditional centralized search engines (like the standard Crawler-Indexer model). It provides a framework for gathering, indexing, caching, and replicating information from the Web in a way that minimizes network traffic and server load.

It was specifically designed to solve the following problems found in centralized architectures:

1. High load on web servers caused by constant crawling.

2. High network traffic generated by transferring full documents.

3. Lack of coordination among different crawlers.

Architecture Diagram

1. **Web Site (Provider):** The source of the data.

2. **Gatherer:** Connected directly to the Web Site.

3. **Broker:** Receives data from Gatherers.

4. **Replication Manager:** Oversees the system.

5. **Object Cache:** Stores frequently accessed data.

6. **User:** Interacts with the Broker.

(Conceptual Flow: Web Site → Gatherer → Broker → User)

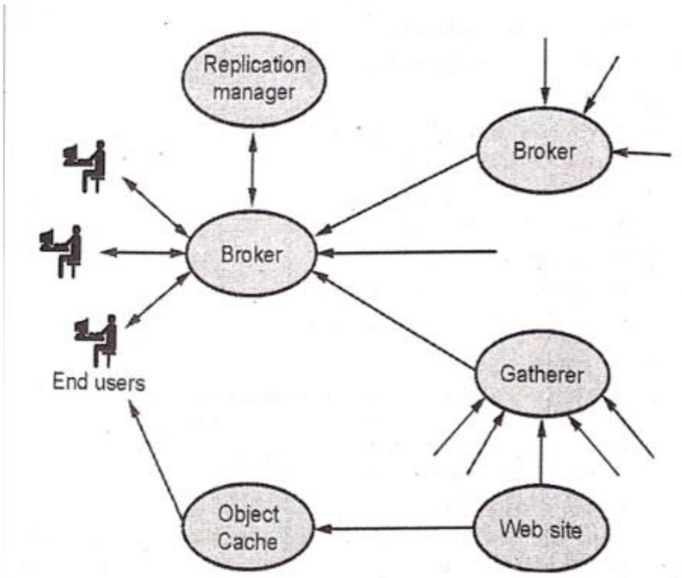


Fig. Q.13.1 Harvest architecture

Key Components of Harvest

The Harvest system is modular and consists of the following four primary components:

1. The Gatherer
• **Location:** Deployed close to the data source (the Web Server).
• **Function:** It is a subsystem that collects indexing information. Instead of downloading full documents to a central location, the Gatherer extracts information (keywords, summaries, bibliographic data) locally.
• **Efficiency:** It runs periodically to check for updates, minimizing the load on the web server compared to random hits from external crawlers. It sends only **summarized data** to the Broker, not the full document content.

2. The Broker
• **Function:** This component acts as the **Indexer** and **Query Interface**.
• **Mechanism:** It retrieves the summarized information from one or more Gatherers (or other Brokers). It creates an index of this data.
• **Flexibility:** Brokers can be specialized (e.g., a "Sports Broker" gathering data from sports-related Gatherers). It allows users to query the system and provides a retrieval interface.

3. The Replication Manager
• **Function:** It manages the replication of Brokers and Gatherers.
• **Goal:** It enhances **scalability** and **fault tolerance**. If one server is overloaded, the replication manager can distribute the load or direct users to a replica server.

4. The Object Cache
• **Function:** It creates a local storage of frequently accessed documents.
• **Goal:** It reduces network traffic and server load. When a user requests a document, the system checks the cache first. If the document is available, it is served immediately, reducing response latency.

Advantages of Harvest Architecture

1. **Reduced Network Traffic:** By sending only **summaries/indexing info** (which are much smaller than the original data) from the Gatherer to the Broker, network usage is significantly reduced (by up to 30-40% compared to centralized models).

2. **Reduced Server Load:** Gatherers are coordinated and run periodically, preventing the "battering" of web servers by multiple uncoordinated crawlers.

Q9) What is hyperlink? Explain structure of hyperlink and also explain searching using hyperlinks
A hyperlink is the fundamental connector that links hypertext documents within the World Wide Web.

- **Definition:** The World Wide Web is a system of interlinked hypertext documents accessed via the Internet. Web pages use hyperlinks to allow users to navigate between different web pages.
- **Conceptual Model:** In the case of hypertext, the conceptual model consists of the **data** and the **hyperlink**.
- **Role:** The power of the Web resides in the capability of hyperlinks to redirect information flow. This structure allows the system to evaluate the information content of a Web object effectively.

2. Structure of Hyperlink (in Web Retrieval)

HTML syntax (e.g., <a href>), they describe the **structural properties** of hyperlinks used in information retrieval and web searching:

- **Graph Structure:** The Web is viewed as a directed graph where pages are nodes and hyperlinks are edges. This structure is analyzed to find patterns such as "Hubs" and "Authorities".
- **Redirection of Flow:** The structure of a hyperlink is designed to redirect the flow of information. It creates a natural pathway that can be analyzed to understand the relationship between two pages.
- **Linkage Types:**
    - **In-links (Incoming edges):** Links pointing *to* a specific page.
    - **Out-links (Outgoing edges):** Links pointing *from* a specific page to others.

3. Searching Using Hyperlinks

Searching using hyperlinks involves exploiting the link structure of the Web to improve the accuracy and relevance of search results. Unlike traditional text retrieval which looks only at page content, this approach analyzes how pages are connected.

Key Applications of Hyperlink Searching: According to the sources, experiments confirm that hyperlinks are valuable for the following purposes:

1. **Improving Initial Ranking:** Enhancing the list of documents returned by a standard keyword search.
2. **Computing Popularity:** Estimating a Web page's popularity based on how many other pages link to it.
3. **Finding Hubs and Authorities:** Identifying key resources for a given topic.

Methods and Algorithms:

- **PageRank (Link Analysis):**
    - This technique interprets a hyperlink as a "citation" or a "vote." The core idea is: *"If a document is cited by a popular document, then it is possibly popular"*.
    - PageRank evaluates the links on a page as part of the ranking factors to push the most relevant and authoritative pages to the top of the search results.
    - It calculates a probability (value between 0 and 1) that a person clicking on a random link will arrive at a specific document.
- **HITS Algorithm (Hubs and Authorities):**
    - This algorithm uses hyperlink structure to distinguish between two types of pages:
        - **Hubs:** Pages that have many outgoing links to relevant content (e.g., a directory or list of resources). Better hub pages come from outgoing edges to good authorities.

        - **Authorities:** Pages that are pointed to by many hubs (e.g., the actual resource or high-quality content page). Better authority pages come from incoming edges from good hubs.
    - The system iteratively calculates scores for hubs and authorities to avoid the "explosion" of the link matrix.
- **Web Query Languages:**
    - Specific query languages have been developed that require knowledge of the Web site's structure. These languages utilize the underlying idea of knowledge induced by the presence of a hyperlink between two pages to refine searches.