Google search Design

Wednesday, 16 October 2024 6:31 PM

Designing a Search Engine like Google:

- We can first have a distributed data store of page content
- Algorithm approach to prioritize top search in list of items
    - Basic formula - Term Frequency / Document Frequency , Frequency or term of a document / Frequency or term of all the documents - This is wrong formula as we need to compute the frequency of all the document across internet.
    - Google's search ranking algorithm, primarily known as **PageRank**, ranks web pages based on multiple factors, including relevance and authority.
        - Key Factors
            - Relevance to the Query
                - **Keywords**: Pages containing keywords or phrases matching the search query.
                - **Semantics**: Understanding the meaning behind words, using Natural Language Processing (NLP) to match intent, not just keywords.
            - **PageRank**: A key concept where pages are ranked based on the number and quality of inbound links (backlinks). Pages linked from authoritative sources tend to rank higher.
                - Each link is treated as a "vote" for a page, and pages with more votes from reputable sites get higher ranks.
            - **Content Quality**: The algorithm evaluates how useful, original, and up-to-date the content is.
                - **Freshness**: Recent content might get a boost, especially for queries that require updated information.
                - **User Engagement**: Google uses signals like click-through rates, time on page, bounce rates, etc., to evaluate how users interact with content.
        - **User Experience**: Pages that load fast, are mobile-friendly, and offer secure connections (HTTPS) tend to perform better.
        - **RankBrain**: Google uses a machine learning algorithm (RankBrain) to understand the search intent better and provide the most relevant results. It looks at historical patterns and user behavior.
        - **Location & Language**: Google adjusts rankings based on the searcher's location, language preferences, and other personal factors (if logged in).
- Designing an Algorithm Similar to Google Search

    Crawling and Indexing:

    - **Crawlers**: The algorithm(Google bot crawl all the pages and stores metadata about each page) starts by crawling web pages. The crawler visits links from known pages and recursively indexes new ones. For efficiency, prioritize crawling pages that are updated frequently or have high authority. It stores the snapshot of all the pages metadata kind of a mini google and kept for indexing
    - **Indexing**: Once crawled, (An **inverted index** is a data structure used by search engines to make searches faster and more efficient. Instead of storing pages by their URL or title, an inverted index stores words (or terms) and points to all the documents (web pages) where those words appear) the page's content is stored in a database, where the text is tokenized, parsed, and indexed by keywords. Metadata, title tags, headers, and links are also indexed to enhance search relevance.

        How an Inverted Index Works:

        - Think of it as the opposite (inverted) of a traditional book index. In a book index, you have a list of **topics** and for each topic, you find **page numbers** where those topics are discussed.
        - In an **inverted index**, you have a list of **words (terms)** and for each word, you find the **URLs** or **documents** where those words appear.

        Example of an Inverted Index:

        Imagine you have 3 web pages:

        1. Page 1 contains the text: **"Cats are cute"**
        2. Page 2 contains the text: **"Dogs are friendly"**
        3. Page 3 contains the text: **"Cats and dogs are playful"**

        The inverted index for these pages would look like this:

Word	Documents (URLs/Pages)
cats	Page 1, Page 3
are	Page 1, Page 2, Page 3
cute	Page 1

dogs	Page 2, Page 3
friendly	Page 2
playful	Page 3

How Search Uses the Inverted Index:
When a user types a query like "cats are cute":

1. The search engine looks up the word "cats" in the inverted index and finds Page 1 and Page 3.
2. Then, it looks up the word "are" and sees it appears in all 3 pages.
3. Finally, it looks up "cute" and finds only Page 1.
4. The search engine will return Page 1 as the best result since it contains all the terms the user searched for.

Scoring Mechanism:

- Each indexed page is scored based on various factors:
  - **Content Relevance**: Match the page's keywords, headings, and metadata against the search query.
  - **Link Analysis** (PageRank): Use a link-based ranking algorithm like PageRank to measure the page's authority by evaluating incoming links.
  - **Content Quality**: Introduce scoring metrics based on content structure, media richness, originality, and freshness.
  - **User Engagement**: Measure user engagement based on how users interact with the content (time on site, click-through rates, bounce rates).

Search Query Handling:

- When a user submits a search query, the system identifies the **intent** (informational, navigational, transactional) using NLP models.
- The query is matched against the indexed data to retrieve the most relevant pages.

4. Relevance Ranking:

- **Relevance Score**: Each matching document is assigned a score based on:
  - **Keyword Matches**: Pages with the query's exact and semantic keywords rank higher.
  - **Authority Score**: Based on PageRank and backlink analysis.
  - **Freshness Score**: Newer content gets higher priority for time-sensitive queries.
  - **User Engagement Score**: Past interactions (clicks, bounce rates) provide additional ranking signals.
  - **Location**: Pages related to the user's geographical location may rank higher.

5. Final Ranking:

- Once all pages are scored, a **weighted ranking formula** combines these scores, presenting the top-ranked pages to the user. The formula would be dynamic and consider relevance, authority, and user experience.

6. Machine Learning:

- Implement a learning system similar to **RankBrain**, where historical data is used to improve future ranking. Machine learning models adjust weights for factors such as user preferences, behavior patterns, and content relevance over time.

7. Continuous Updates:

- Regularly update the index and refresh scoring factors based on algorithm improvements. Introduce personalized ranking, where user-specific data (past searches, history) influences the order of results.

How the Data is Stored:

1. **Crawl Database**: Pages are stored as text (stripped of HTML tags) with metadata (URLs, headers, timestamps).
2. **Inverted Index**: A large-scale search index is used where words point to all the documents (URLs) that contain them.
3. **Link Graph**: A graph-based structure is used for PageRank, representing links between websites.
4. **Engagement Data**: Click-through rates, bounce rates, etc., are stored in a behavior database for personalized ranking.
5. **Machine Learning Models**: RankBrain and other ML models are trained on historical search and click patterns.

By providing an answer based on these principles, you demonstrate a strong understanding of search engine mechanics and can address specific components like ranking, crawling, and scoring efficiently.

We can also use forward index..
Forward Index: The forward index is the opposite of an inverted index. Instead of mapping words to documents, a forward index maps **documents (pages)** to **words**. It answers the question: **"Which words are found in this**

document?"
- Example: Page 1 contains the words: **"cats, are, cute"**.

How They Work Together:
- **Forward Index**: First stores which words are present in each document, along with metadata (such as the location of the word: title, body, etc.).
- **Inverted Index**: Then built from the forward index by reversing the relationship (mapping words to documents).

Word Frequency:
- A single word can appear in millions of pages (e.g., "the" or "and"). To handle this, search engines can implement **frequency-based pruning** where common words are deprioritized in the search index.

Handling Titles, Headers, and Content Weighting:
In search engines, words that appear in **titles, headers, or meta tags** are more important than words in the body of the text. This is where **ranking algorithms** (like PageRank or others) come into play.
- In the **forward index**, each word is not just stored with its position but also with **metadata** about where it appears:
    - Is the word in the **title**?
    - Is the word in the **header**?
    - Is it bolded or italicized?
    - Is it part of the **URL**?

How Search Engines Handle Huge Volumes of Data:
To deal with billions of words and billions of pages, search engines break the problem into smaller pieces:
- **Sharding**: The forward and inverted index are broken into smaller pieces called **shards**. Each shard contains a portion of the index (for example, a subset of words). These shards are distributed across many servers, allowing the search engine to process huge amounts of data in parallel.
- **Replication**: Data is also replicated across different servers to ensure reliability and speed. This allows for **fault tolerance** (in case a server fails) and faster **retrieval** of data.
- **MapReduce**: Technologies like **MapReduce** are used to break the processing of large datasets into smaller tasks that can be handled in parallel by different machines. Google originally developed MapReduce to handle its indexing process.

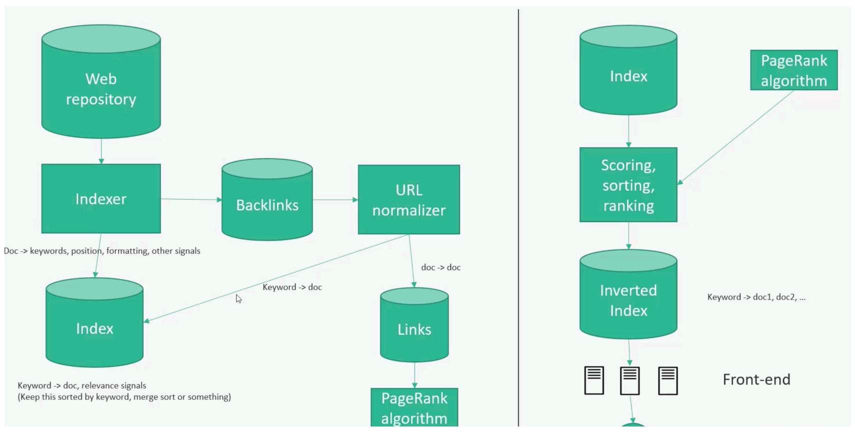
Sharding is a database partitioning technique where large datasets are divided into smaller, more manageable pieces called **shards**. Each shard contains a subset of the entire dataset, which can be stored on different servers or machines. Sharding is used to improve performance, scalability, and fault tolerance.

Example:
Imagine a search engine has to index a huge set of web pages. Instead of storing the entire index in one large database (which could be slow and difficult to manage), the data can be split into smaller shards:
- **Shard 1**: Contains documents (or web pages) with URLs starting from "A" to "F".
- **Shard 2**: Contains documents with URLs starting from "G" to "L".

MapReduce is a programming model used for processing and generating large datasets by dividing the work into smaller tasks that can be processed in parallel. It consists of two main phases: **Map** and **Reduce**.
- **Map phase**: Processes input data and converts it into intermediate key-value pairs.
- **Reduce phase**: Aggregates and processes the intermediate key-value pairs to produce the final result.

Example:
Let’s say Google wants to count the number of occurrences of a specific word (e.g., "cloud") across billions of web pages. Here’s how MapReduce would work:
- **Map phase**: The entire dataset (web pages) is split into smaller chunks (e.g., individual web pages). Each chunk is processed in parallel on different servers, and for every occurrence of the word "cloud" in a page, the mapper emits a key-value pair ("cloud", 1).
    - Page 1: Emits ("cloud", 1), ("cloud", 1)
    - Page 2: Emits ("cloud", 1)
    - Page 3: Emits ("cloud", 1), ("cloud", 1), ("cloud", 1)
- **Shuffle and Sort**: The intermediate key-value pairs are shuffled and sorted by the key ("cloud").
- **Reduce phase**: The reducer takes all the key-value pairs associated with the word "cloud" and sums them up to get the total count of occurrences of the word "cloud" across all pages.
    - ("cloud", 1), ("cloud", 1), ("cloud", 1), ("cloud", 1), ("cloud", 1) → **Total count: 5**

Designing the system:


Web repository
Indexer
Backlinks
URL normalizer
Doc -> keywords, position, formatting, other signals
Keyword -> doc
doc -> doc
Index
Links
Keyword -> doc, relevance signals
(Keep this sorted by keyword, merge sort or something)
PageRank algorithm
Index
PageRank algorithm
Scoring, sorting, ranking
Inverted Index
Keyword -> doc1, doc2, ...
Front-end