Web Crawler

Design Document

Submitted by
Hasan Mahmud

This document outlines the design and architecture of a web crawler, which is a program that systematically browses the internet, gathers information, and indexes web pages for various purposes such as search engine indexing, data mining, or content aggregation.

Goals and Objectives

The primary goals of the web crawler project are as follows:

- Retrieve and analyze web pages from the internet.
- Extract relevant data from web pages according to predefined rules.
- Store crawled data efficiently for further processing or analysis.
- Ensure scalability, reliability, and performance under varying workloads.

Architecture Overview

The web crawler architecture consists of the following components:

Crawler Module

- Responsible for fetching web pages from the internet.
- Implements logic for crawling rules, such as respecting robots.txt directives and avoiding crawling duplicate URLs.
- Utilizes multi-threading or asynchronous processing for concurrent fetching of multiple web pages.

Parsing Module

- Parses the HTML content of fetched web pages to extract relevant data.
- Utilizes parsing libraries or frameworks such as BeautifulSoup or Scrapy for efficient extraction of structured data.
- Supports customization for different types of content extraction (e.g., text, images, metadata).

Storage Module

- Stores crawled data in a structured format for further processing or analysis.
- Utilizes databases for storing metadata about crawled URLs, including URL, timestamp, HTTP status code, and other relevant information.
- Utilizes object storage for storing the actual content of web pages, such as HTML, images, or other multimedia files.

Scheduler Module

- Manages the scheduling of URLs to be crawled.
- Implements prioritization logic based on factors such as domain authority, freshness, or relevance.
- Supports distributed scheduling to distribute workload across multiple crawler instances.

Monitoring and Logging

- Monitors the health and performance of the web crawler system.
- Logs important events and errors for troubleshooting and auditing purposes.
- Integrates with monitoring and logging services such as AWS CloudWatch or Google Stackdriver.

System Flow

The typical flow of the web crawler system is as follows:

1. The Scheduler Module generates a list of URLs to be crawled based on predefined criteria or seed URLs.
2. The Crawler Module fetches web pages asynchronously, respecting crawling rules and directives.
3. The Parsing Module extracts relevant data from fetched web pages and stores it in the database and object storage.
4. The process repeats iteratively, with the Scheduler Module continuously updating the list of URLs to be crawled based on new discoveries or changes.

Scalability and Performance Considerations

- The system architecture is designed to be horizontally scalable, allowing for the addition of more crawler instances to handle increased workload.
- Load balancing and auto-scaling mechanisms ensure efficient resource utilization and high availability under varying traffic conditions.
- Caching mechanisms may be employed to reduce latency and improve performance for frequently accessed resources.

Security Considerations

- Implements security measures such as rate limiting, authentication, and access control to prevent abuse or unauthorized access.
- Utilizes encryption for data at rest and in transit to protect sensitive information.
- Regularly audits and updates dependencies to address security vulnerabilities.

Future Enhancements

- Implement support for distributed crawling across multiple geographically distributed locations.
- Enhance parsing capabilities to support more advanced content extraction techniques such as natural language processing or machine learning.
- Integrate with external APIs or services for additional data enrichment or analysis.

The web crawler design outlined in this document provides a scalable and efficient solution for retrieving, parsing, and storing web content from the internet. By following best practices in architecture, scalability, security, and performance, the system is well-equipped to handle a wide range of crawling tasks and adapt to changing requirements over time.