Scalable Web Crawler Design Documentation

1. Overview, Requirements, and Assumptions

A. Functional Requirements:

- Web page crawling: Fetch pages and follow links.
- Politeness and robots.txt compliance: Respect crawl delays and disallowed paths.
- URL prioritization: Schedule URLs based on freshness, importance, and domain policies.
- Content parsing: Extract links and page metadata; detect duplicates with fingerprints.
- Storage and indexing: Persist raw pages and metadata for later indexing.
- Monitoring and error handling: Log successes, failures, and adjust scheduling dynamically.

B. Nonfunctional Requirements:

- High throughput: Fetch millions of pages per day using thousands of concurrent workers.
- Low latency: Quick processing and scheduling of pages.
- Fault tolerance: Distributed, replicated systems to avoid single points of failure.
- Scalability: Horizontal scaling of fetchers, frontiers, and storage systems.
- Efficient resource utilization and secure operations.

C. Assumptions:

- Billions of URLs exist on the Web; deduplication is essential.
- The crawler must be distributed across multiple regions to reduce latency.
- A distributed message broker and in-memory caches (e.g., Redis) will support stateful components.

2. High-Level Architecture and Component Responsibilities

A. URL Frontier and Scheduler:

- Maintains a queue of URLs to crawl, with deduplication to filter out redundant URLs.
- Uses distributed storage (e.g., a NoSQL database or Kafka) for high throughput.

B. Fetchers and Politeness Manager:

- Distributed fetcher nodes make HTTP requests to download pages.
- A Politeness Manager reads robots.txt and enforces per-domain crawl delays.

C. Parsing and Content Processing:

- Parser workers extract HTML content, links, and metadata.
- Content fingerprinting detects duplicate pages.

D. Data Storage and Indexing:

- Raw content is stored in a distributed object store (e.g., S3/HDFS).

- Metadata is saved in a NoSQL database for fast retrieval and indexing.
- An indexing pipeline processes stored data for search engine input.

E. Messaging and Queuing:

- A message broker (e.g., Kafka) decouples fetching, parsing, and storage.

F. Monitoring and Analytics:

- Real-time dashboards track crawl rates, errors, and latency metrics.

3. Detailed Workflow

A. URL Discovery and Scheduling:

- URLs are submitted as seed URLs or extracted from crawled pages.
- Deduplication is performed using distributed hash tables or Bloom filters.
- The URL frontier schedules URLs based on priority rules.

B. Fetching Process:

- Fetcher nodes pull URLs from the frontier and check the politeness policy.
- Pages are downloaded with robust error handling and retries.

C. Parsing and Extraction:

- Downloaded pages are parsed to extract links and content.
- Extracted links are normalized and reinserted into the frontier.
- Fingerprints are computed for duplicate detection.

D. Storage and Indexing:

- Raw HTML and assets are stored persistently.
- Metadata is recorded for indexing and search engine input.

E. Monitoring:

- Systems log performance, errors, and throughput for real-time adjustments.

4. Scalability, Fault Tolerance, and Global Distribution

A. Horizontal Scalability:

- Thousands of fetcher and parser nodes work concurrently.
- The URL frontier is sharded across multiple nodes using consistent hashing.

B. Fault Tolerance:

- Data replication in object stores and metadata databases ensures durability.
- The message broker supports partitioning and redundancy to handle failures.

C. Global Distribution:

- The crawler is deployed in multiple data centers; regional frontiers reduce latency.
- Global DNS and routing systems direct traffic to the nearest available resources.

5. Protocols and External Integrations

A. Communication Protocols:

- Client-to-Server: HTTPS for API requests and secure data transfer.
- Interservice Communication: gRPC or REST over secured TCP with mutual TLS.

B. External Integrations:

- DNS Caching: Reduces lookup latency.
- Robots.txt Fetching: Ensures compliance with website crawl policies.
- CDN: May be used for caching static assets if needed.

C. Load Balancing and Autoscaling:

- Global load balancers and regional autoscaling groups ensure that the system scales with increasing crawl demand.

6. Final Thoughts

This design outlines a robust architecture for a huge web crawler capable of indexing billions of pages. Key features include:

- A distributed URL frontier and deduplication mechanism to manage billions of URLs.
- A massive fleet of fetchers and parser nodes that work concurrently while respecting politeness policies.
- A decoupled, high-throughput pipeline using message brokers and sharded storage systems.
- Horizontal scalability and fault tolerance through replication, autoscaling, and global distribution.

While industry leaders like Google incorporate many proprietary optimizations, this framework provides a comprehensive foundation upon which such a system can be built.