Web Crawler Design

Functional Requirements

1. What is the use case: search
2. Storage: how long does the information persist? 5 years
3. Just HTML crawling; no PDF crawling.
4. How many pages do we crawl per month? 1 billion

Scale and Storage Calculations

1. 1 billion pages per month: 100 KB per page: * 60 = 6 PB for 5 years storage
2. QPS ~ 300 per second

High-level abstract design

Application Layer Design

1. Have a seed URL
2. Parse the contents of the seed URL
    a. Store the contents (plain-text)
    b. While parsing, every hyperlink, goes into “URL Processing Queue”
3. (URL Processing Queue).pop() goes into Step 1.

Data Layer Design

1. NoSQL database: Document type NoSQL:
    a. Primary key: hash(normalized(url))
    b. URL
    c. Contents

More Detailed Design

Seed URL — URL Processing Queue — Content Parser — Extract

Scale Design

API Layer

1. Caching caching caching

Data Layer

1. We know we’re going to have a lot of data right from the get go: For the DB, we should shard right from the start: hash(normalized(url)) % 6000
2. To feed into the search indexing service:
    a. Asynchronous queues that feed that data to the service
    b. Add a Read replica to the DB, the sole purpose is to feed into the service

Deeper-Dive

1. URL Processing Queue: Politeness. The queue itself is fed via a multi-level queue. Implement a rate-limiting algorithm based off of the URL domain.
    a. Every domain gets its own queue

b. While the processing on those queues happens with a FIFO scheme, the queues themselves get selected round-robin.
    i. URL -> hash(URL) - let's decide on having 50 such queues: hash(URL)%5000.
    ii. Also implement a URL domain-centric rate limiter; rather than have a rate per domain, maybe have a rate per hash(domain)%5000.