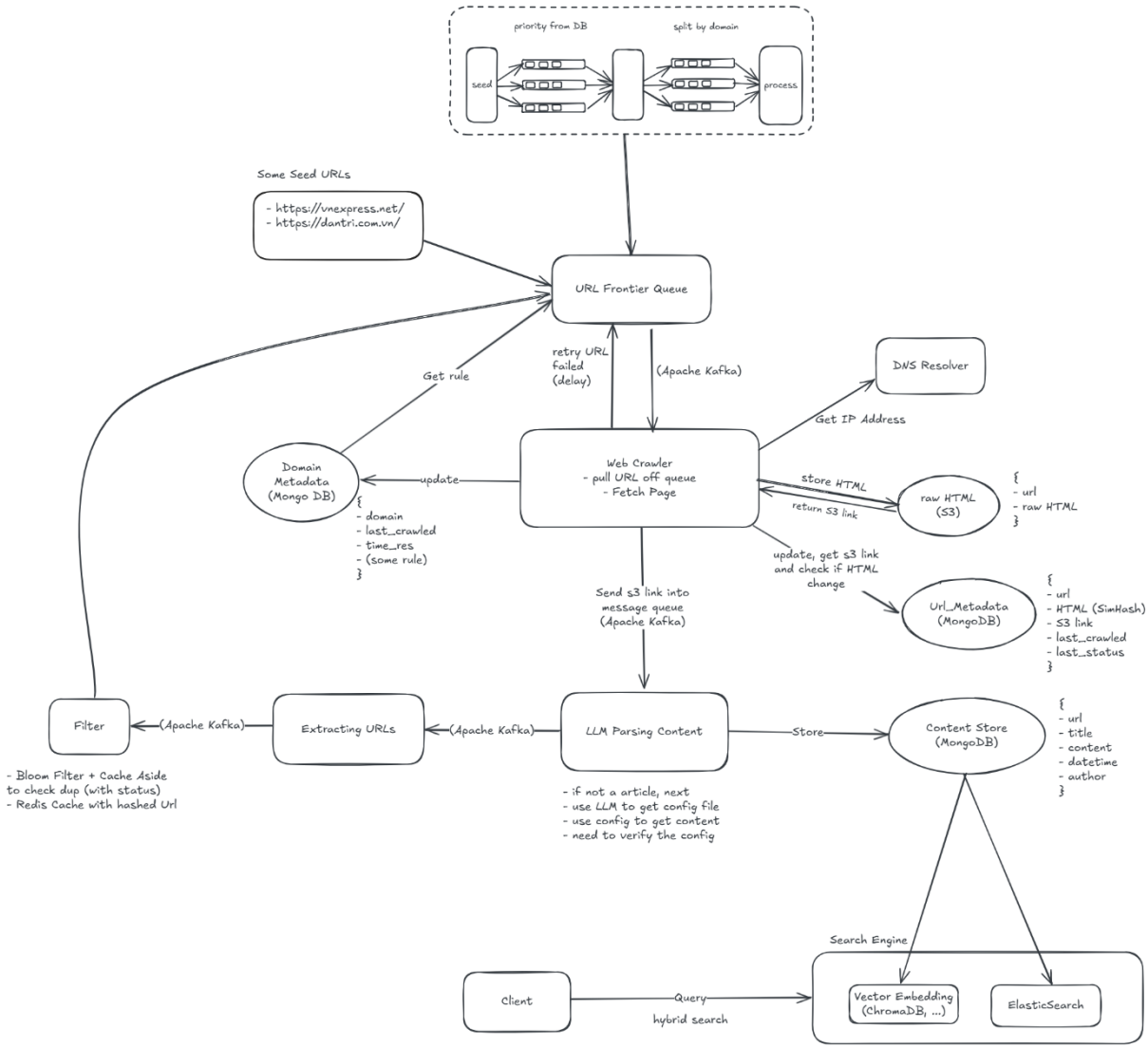
Thiết kế hệ thống cho Crawl System (báo chí):



Data Flow:

1. Nhập Seed URLs (do người quản trị nhập)
2. Chuyển Seed sang Frontier Queue, tại đây sẽ được phân chia lại queue theo độ ưu tiên và chia theo domain
3. Web Crawler thực hiện lấy các url từ queue và thực hiện fetch (tại đây trải qua đầy đủ các bước: lấy IP thông qua DNS resolver; Kiểm tra đã crawl url này chưa, nếu rồi thì lấy raw HTML đã lưu trên S3 để kiểm tra thay đổi, nếu thay đổi thì tiếp tục. Chưa crawl thì thực hiện tạo mới; Update lại metadata cho domain đó)
4. Gửi link s3 chứa HTML ở trên vào message queue
5. Thực hiện parsing nếu đó là một trang bài báo.
6. Lưu nội dung parse được vào DB, cũng như có trigger để cập nhật search engine.
7. Thực hiện extract url mới
8. Thông qua bộ lọc và gửi về Frontier Queue

9. Lặp lại các bước trên

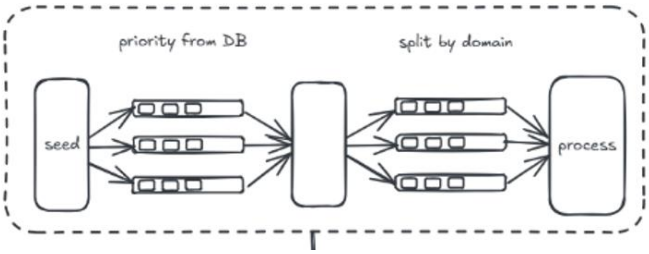
Chi tiết các module chính như sau:

1. **Seed URLs**

- Mục đích: Tạo và quản lý seed URLs (người quản trị sẽ nhập các trang home để có thể lấy các bài báo mới nhanh nhất). Ngoài ra có thể set 1 schedule cho phần này để tự động lập lịch chạy lại sau khoảng 15-20p để đảm bảo yêu cầu có bài báo mới nhất muộn nhất 30p.
- Có thể sử dụng ScheduledExecutorService ở phần này.

2. **URL Frontier Queue**

- Mục đích: Tạo một hàng đợi cho các url từ Module trước đẩy sang hoặc các url lỗi cần retry. Để tối ưu thì đây nên là một hàng đợi ưu tiên có thể chia 1 đến 2 lần.



- Cách hoạt động:
    - Nhận được danh sách các url
    - Lớp đầu tiên sẽ lấy ra thông tin từ Domain Metadata để có thể xét độ ưu tiên cho url đó (có thể dựa trên time response, last crawl, hoặc cơ chế đánh điểm riêng). Ví dụ như vnexpress/trangchu > vnexpress/thoisu.
    - Các queue với độ ưu tiên lớn hơn sẽ được chọn trước.
    - Lớp 2 sẽ lọc các url trong cùng 1 domain ra thành các queue khác nhau để đảm bảo tránh bị block bởi website.
    - Sau đó sẽ có cơ chế phân chia các url vào các worker với delay sao cho phù hợp với từng queue.
- Công nghệ:
    - Ở đây việc gửi url đến các worker hay module sau sẽ dùng Apache Kafka để gửi vào các partition thích hợp. Có các group để xử lý riêng.
    - Dùng kafka vì sẽ đảm bảo được durability do message sẽ được lưu vào disk, và kafka cũng có cơ chế tự retry nếu như xử lý lỗi. Mỗi bước dùng kafka làm message queue ở sau sẽ dùng các topic khác nhau để quản lý. Topic giống như loại công việc nó sẽ đảm bảo các module nhận đúng công việc cần làm.

3. **Web Crawler**

- Mục đích: nhận url từ Frontier queue và thực hiện fetch page.
- Cách hoạt động:
    - Lấy URL từ queue

    - Sử dụng DNS lấy IP, gửi request để lấy HTML
    - Kiểm tra từ DB (URL metadata) xem đã crawl url này chưa
    - Nếu đã có thì chỉ cần lấy s3 link cũng như hashed HTML để kiểm tra
    - Thực hiện kiểm tra HTML vừa request và HTML đã lưu trước đó.
    - Nếu không có gì khác biết thì sẽ bỏ qua url này.
    - Nếu Chưa có thông tin metadata thì sẽ lưu mới trên cả DB và S3.
    - Tại đây cũng cập nhật Domain metadata để phục vụ một số chức năng khác.
    - Có cơ chế retry và lại url về frontier
    - Sau cùng thì sẽ gửi message đi với chỉ link S3 chứa file raw HTML (do Gửi message với chỉ 1 link sẽ nhanh hơn so với gửi cả 1đoạn HTML)
- Công nghệ:
    - Sử dụng S3 làm nơi lưu trữ raw HTML kích thước tệp khá lớn, về lâu dài sẽ gây áp lực lên DB (ví dụ như Mongo). S3 thì hỗ trợ lưu được phi cấu trúc, các file lớn, tài nguyên lớn. chi phí rẻ. Tuy nhiên sẽ mất thời gian để thao táo tải cũng như ghi. Nhưng chắc là đánh đổi được.
    - Về URL metadata thì do đây là dữ liệu cấu trúc, và khá nhẹ do HTML đã được hash nên là có thể dùng Mongo để lưu luôn, truy vấn cũng nhanh hơn.
    - Thuật toán hash được dùng là simHash, nó sẽ có ngưỡng chấp nhận sai lệch của dữ liệu, nên là trong khoảng chấp nhận thì coi như HTML không khác nhau.

4. LLM Parsing Content

- Mục đích: Trích xuất nội dung (tiêu đề, nội dung, ngày đăng, tác giả) từ HTML của article URLs.
- Cách hoạt động:
    - Lấy HTML từ S3, áp dụng selector từ config (lấy từ DB hoặc cache)
    - Nếu trong DB không có config thì sẽ làm các bước sau:
        - Cắt bớt HTML nếu vượt context window của LLM, giữ các thẻ quan trọng (`<h1>`, `<div>`), gửi đến LLM (Mistral, Gemini, LLama) với prompt yêu cầu tạo XPath selectors
        - LLM trả về JSON config
        - Lưu config vào MongoDB, cache trong Redis.
    - Nếu không có lỗi hay không được thì gửi đến LLM để lấy nội dung
    - Làm sạch dữ liệu: Loại bỏ quảng cáo, Chuẩn hóa datetime
    - Lưu kết quả JSON

5. Content Store

- Mục đích: Lưu trữ bài viết để tìm kiếm và phân tích
- MongoDB: Phù hợp với cấu trúc JSON không cố định
- Có thể thêm cơ chế trigger khi nội dung mới được thêm hoặc cập nhật trong Content Store

6. Search Engine

- Mục đích: Cung cấp API tìm kiếm (hybrid: Dựa vào loại truy vấn để quyết định ưu tiên kết quả từ cái nào)
- Khi có cập nhật mới trên Content Store:
    - Với Vector Embedding (ChromaDB): Chuyển đổi nội dung văn bản thành vector nhúng (embeddings). Lưu trữ vector cùng với metadata (url, title) để tìm kiếm sau này. Phương pháp này cho phép tìm kiếm ngữ nghĩa (semantic search) dựa trên độ tương đồng vector.
    - Với ElasticSearch: Đánh chỉ mục toàn văn (full-text indexing) cho nội dung. Hỗ trợ tìm kiếm từ khóa, phân tích văn bản.
- Có thể thêm cơ chế ranking kết quả.

7. Extract URLs

- Mục đích: Từ HTML có được, loang thêm được các url được gắn vào trang, nhằm lấy nhanh được các bài báo hơn, cũng như phục vụ khai phá cấu trúc trang web.
- Kết quả là sẽ có được một danh sách các url, và ta sẽ cần phải thông qua một bộ lọc để giảm tải cho hệ thống.
- Bộ lọc sẽ dùng Bloom Filter + Redis (Cache Aside) nhằm bỏ đi các url đã được crawl.
- Công nghệ:
    - Bloom Filter là một cấu trúc dữ liệu dạng xác suất, giúp xác định một thành phần có trong tập hợp hay không. Nó sẽ có % bị fasle positive, nghĩa là url đó chưa crawl nhưng lại nói crawl rồi. tuy nhiên % đó nhỏ và tùy thuộc vào cách ta setup. Nó có ưu điểm nếu như nói chưa crawl thì chắc chắn là chưa, nên nó sẽ tiết kiệm được kha khá thời gian và tài nguyên.
    - Sau khi dùng bloom filter nếu kết quả là đã crawl thì vẫn nên dùng cache aside vì nếu như trước có lỗi k retry được (dựa vào last_status và last_crawled) thì sẽ cần thử lại.

8. Monitoring & Logging

- Mục đích: Theo dõi hiệu năng, ghi log, và gửi cảnh báo (ELK Stack, Grafana, Prometheus)
- ElasticSearch: lưu trữ, tìm log nhanh
- LogStash: Thu thập, xử lý chuyển đổi logs, đẩy vào ElasticSearch
- Kiban hỗ trợ giao diện dashboard
- Cần thực hiện log và xử lý đối với các thao tác HTTP request, các thao tác check dup, log ra được % dup, % timeout , …