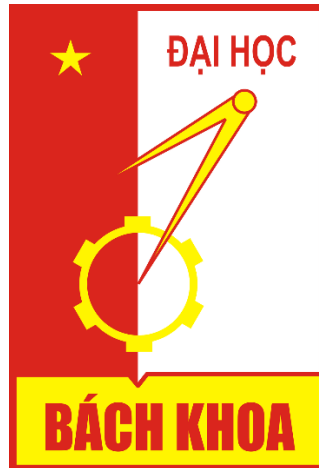


ĐẠI HỌC BÁCH KHOA HÀ NỘI

KHOA TOÁN – TIN



NHÓM 12 – BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN: HỆ THỐNG VÀ MẠNG MÁY TÍNH (MI4060)

CHỦ ĐỀ: Distributed Computing

ĐỀ TÀI: Thiết kế hệ thống hiệu suất cao
dựa trên trình thu thập dữ liệu thông tin web phân tán
Distributed Web Crawler


Giảng viên hướng dẫn: TS. Ngô Thị Hiền

Hà Nội, Tháng 01/2025

DANH SÁCH THÀNH VIÊN NHÓM

 **Danh sách thành viên:**

| STT | Họ và Tên | MSSV |
|-----|--------------------|----------|
| 1 | Nguyễn Trung Kiên | 20227180 |
| 2 | Lê Ngọc Trung Kiên | 20227236 |
| 3 | Nguyễn Doãn Nam | 20223449 |
| 4 | Vũ Lương Duy | 20227226 |
| 5 | Đình Trần Anh Tuấn | 20227206 |
| 6 | Lê Thế Quang | 20227203 |

 **Bảng phân công nhiệm vụ và đánh giá kết quả làm việc:**

| STT | Họ và Tên | Nhiệm vụ | Đánh giá |
|-----|--------------------|---|----------|
| 1 | Nguyễn Trung Kiên | <ul style="list-style-type: none">– Thiết kế Slide.– Kiểm thử Code và đánh giá hiệu suất sản phẩm.– Tổng hợp và thiết kế báo cáo bản cứng.– Thuyết trình sản phẩm. | 9,83 |
| 2 | Lê Ngọc Trung Kiên | <ul style="list-style-type: none">– Biên soạn nội dung.– Thiết kế Slide.– Thuyết trình sản phẩm. | 8,50 |
| 3 | Nguyễn Doãn Nam | <ul style="list-style-type: none">– Code mã nguồn hệ thống.– Kiểm thử Code và đánh giá hiệu suất sản phẩm.– Thuyết trình sản phẩm. | 9,33 |
| 4 | Vũ Lương Duy | <ul style="list-style-type: none">– Biên soạn nội dung.– Thiết kế Slide.– Thuyết trình sản phẩm. | 8,50 |
| 5 | Đình Trần Anh Tuấn | <ul style="list-style-type: none">– Biên soạn nội dung.– Thuyết trình sản phẩm. | 8,00 |
| 6 | Lê Thế Quang | <ul style="list-style-type: none">– Biên soạn nội dung.– Thuyết trình sản phẩm. | 5,67 |

LỜI MỞ ĐẦU

Trong thời đại số hóa, lượng thông tin được tạo ra hàng ngày trên các nền tảng trực tuyến, từ mạng xã hội, website thương mại điện tử, tin tức, đến các dịch vụ dữ liệu mở, đang gia tăng theo cấp số nhân. Khối lượng dữ liệu khổng lồ này không chỉ là cơ hội mà còn đặt ra thách thức lớn đối với các tổ chức trong việc thu thập, xử lý, và phân tích để phục vụ các mục tiêu kinh doanh, nghiên cứu và phát triển.

Đặc biệt, với sự phát triển của thương mại điện tử và các dịch vụ trực tuyến, yêu cầu về dữ liệu chính xác, được thu thập nhanh chóng và kịp thời để đưa ra các quyết định chiến lược ngày càng trở nên cấp thiết. Trong bối cảnh đó, các hệ thống xử lý dữ liệu truyền thống với hiệu suất hạn chế không thể đáp ứng được nhu cầu vận hành liên tục, linh hoạt và khả năng mở rộng.

Hệ thống thu thập dữ liệu phân tán (Distributed Crawling System) đã trở thành một giải pháp tối ưu để giải quyết bài toán trên. Thay vì dựa vào một hệ thống tập trung, cách tiếp cận phân tán cho phép chia nhỏ khối lượng công việc và triển khai đồng thời trên nhiều máy chủ. Điều này giúp cải thiện hiệu suất, giảm thời gian xử lý và tăng khả năng chịu lỗi khi hệ thống phải xử lý khối lượng lớn thông tin từ hàng trăm nghìn website với nhiều cấu trúc khác nhau.

Mục tiêu của đề tài "Thiết kế hệ thống hiệu suất cao dựa trên trình thu thập dữ liệu thông tin web phân tán" là cung cấp một giải pháp thiết kế hệ thống hiệu suất cao, không chỉ đáp ứng nhu cầu thu thập dữ liệu nhanh chóng mà còn đảm bảo khả năng mở rộng và bền vững cho các ứng dụng thực tiễn, đặc biệt trong lĩnh vực thương mại điện tử và các ngành phụ thuộc vào dữ liệu lớn.

MỤC LỤC

| | |
|--|-----------|
| DANH SÁCH THÀNH VIÊN NHÓM | 1 |
| PHẦN A. KIẾN THỨC CƠ SỞ | 5 |
| 1. Hệ thống phân tán | 5 |
| 1.1. Khái niệm..... | 5 |
| 1.2. Đặc điểm | 5 |
| 1.3. Ví dụ..... | 6 |
| 2. Trình thu thập dữ liệu Web phân tán..... | 7 |
| 2.1. Khái niệm..... | 7 |
| 2.2. Đặc điểm | 7 |
| 2.3. Chức năng | 7 |
| 2.4. Ứng dụng thực tiễn của Trình thu thập dữ liệu web phân tán..... | 9 |
| 2.5. Kiến trúc hệ thống..... | 10 |
| 2.6. Thách thức | 10 |
| 3. Hiệu suất cao và tối ưu hóa hệ thống..... | 12 |
| 3.1. Hiệu suất trong Trình thu thập dữ liệu web phân tán..... | 12 |
| 3.2. Các yếu tố cần tối ưu hóa hiệu suất | 19 |
| PHẦN B. NỘI DUNG CHÍNH..... | 32 |
| 4. Mô hình TCP/IP..... | 32 |
| 4.1. Tầng ứng dụng | 32 |
| 4.2. Tầng giao vận..... | 32 |
| 4.3. Tầng mạng | 33 |
| 4.4. Tầng vật lý | 33 |
| 5. Tính bảo mật | 34 |
| 6. Thuật toán | 37 |
| 6.1. Thuật toán Round-Robin..... | 37 |
| 6.2. Thuật toán xử lý URL trong hệ thống..... | 39 |
| 6.3. Thuật toán Retry và xử lý lỗi trong hệ thống..... | 44 |
| 6.4. Thuật toán phân phối công việc trong hệ thống..... | 48 |
| 7. Xây dựng hệ thống..... | 50 |
| 7.1. Kiến trúc hệ thống..... | 50 |
| 7.2. Mã nguồn | 51 |

| | | |
|---|--------------------------------|-----------|
| 7.3. | Chi tiết các tệp hệ thống..... | 74 |
| 7.4. | Thành phần chính..... | 76 |
| 7.5. | Luồng hoạt động | 76 |
| 8. | Đánh giá hiệu suất..... | 77 |
| 8.1. | Điểm mạnh..... | 77 |
| 8.2. | Hạn chế | 77 |
| PHẦN C. TỔNG KẾT – ĐÁNH GIÁ..... | | 78 |
| 9. | Kiến thức..... | 78 |
| 10. | Kỹ năng..... | 79 |
| 10.1. | Kỹ năng chuyên môn | 79 |
| 10.2. | Kỹ năng mềm..... | 79 |
| LỜI CẢM ƠN..... | | 80 |
| TÀI LIỆU THAM KHẢO | | 81 |
| PHỤ LỤC..... | | 82 |

PHẦN A. KIẾN THỨC CƠ SỞ

1. HỆ THỐNG PHÂN TÁN

1.1. Khái niệm

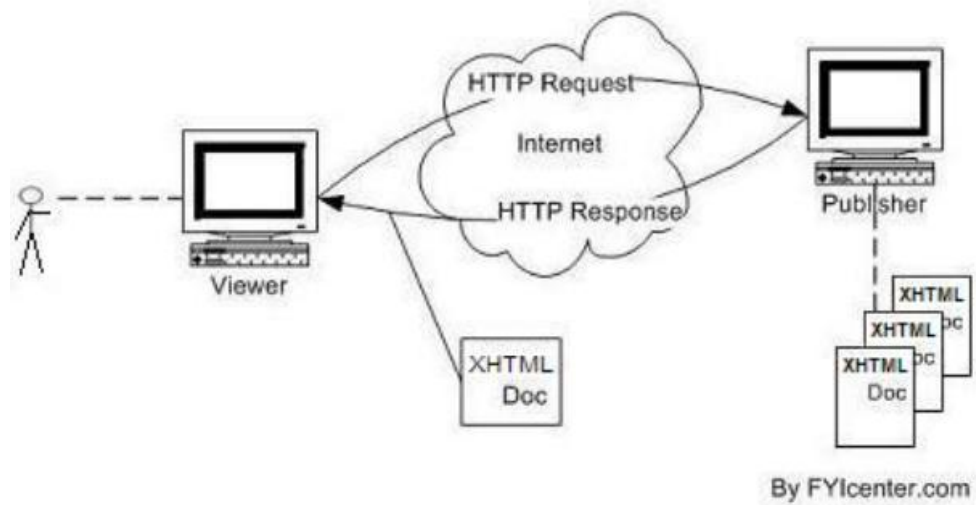
Hệ thống phân tán là một hệ thống có chức năng và dữ liệu phân tán trên các trạm (máy tính) và được kết nối với nhau bằng một mạng máy tính. Hệ thống này chia sẻ tài nguyên, dữ liệu và công việc giữa các máy tính, phối hợp chúng với nhau để hoạt động như một hệ thống duy nhất để đạt được các mục tiêu chung.

1.2. Đặc điểm

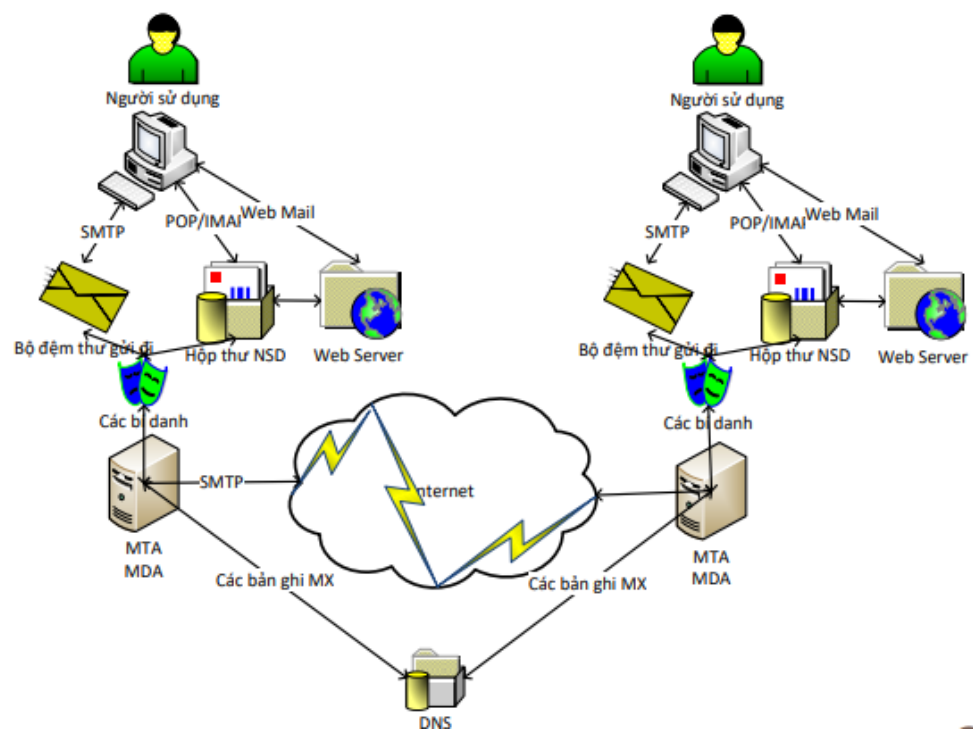
| Đặc điểm | Mô tả |
|--------------------|---|
| Chia sẻ tài nguyên | Cho phép các máy tính trong mạng sử dụng tài nguyên chung một cách hiệu quả và minh bạch, giúp tối ưu hóa việc sử dụng và tiết kiệm chi phí |
| Tính trong suốt | Tính trong suốt của hệ phân tán được hiểu như là sự che khuất đi các thành phần riêng biệt của hệ thống máy tính (phần cứng và phần mềm) đối với người sử dụng và những người lập trình ứng dụng. Người sử dụng có quyền truy cập đến dữ liệu đặt tại một điểm dữ liệu ở xa một cách tự động nhờ hệ thống mà không cần biết đến sự phân tán của tất cả dữ liệu trên mạng. |
| Tính mở | Có thể mở rộng, thay đổi hoặc nâng cấp hệ thống mà không làm gián đoạn hoạt động tại hiện tại và tương thích với nhiều nền tảng, giao tiếp chuẩn hoặc giao thức khác. |
| Tính co giãn | Có khả năng tự động mở rộng hoặc thu nhỏ tài nguyên để đáp ứng sự thay đổi về khối lượng công việc. Có thể bổ sung tài nguyên (như máy chủ, băng thông) khi nhu cầu tăng và giảm tài nguyên khi nhu cầu giảm, giúp tối ưu hóa chi phí. |

1.3. Ví dụ

a) WWW (Word Wide Web)



b) Email



2. TRÌNH THU THẬP DỮ LIỆU WEB PHÂN TÁN

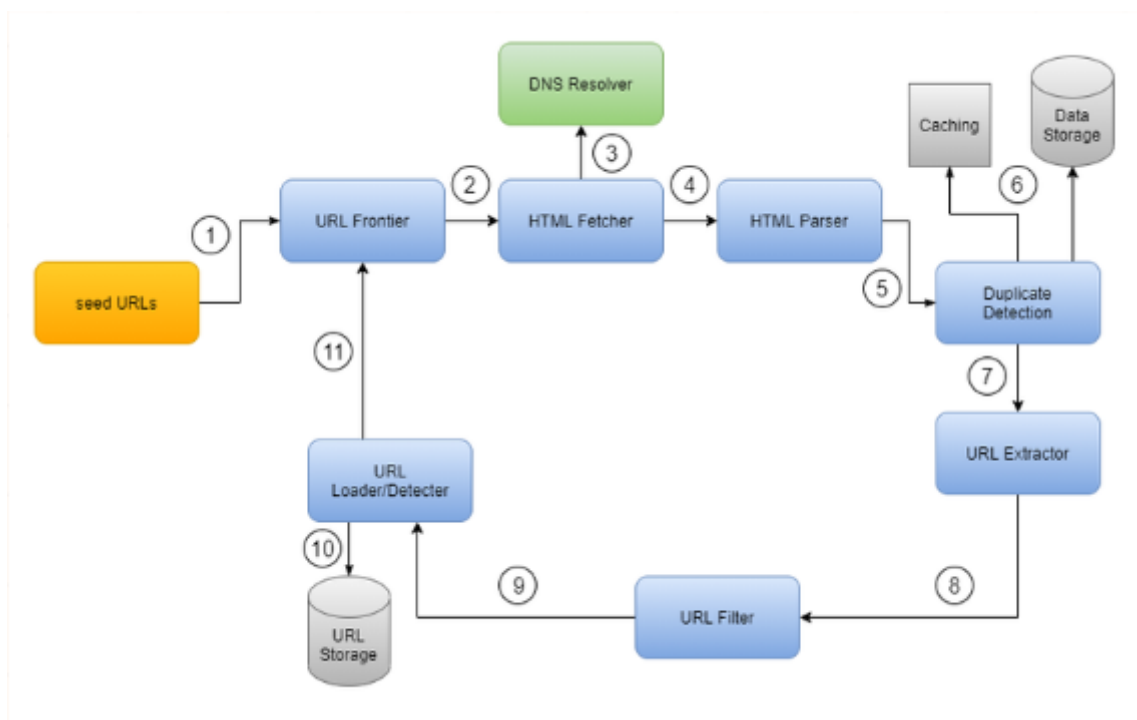
2.1. Khái niệm

Là kỹ thuật sử dụng nhiều máy tính để lập chỉ mục web thông qua việc thu thập dữ liệu tự động từ các trang web.

2.2. Đặc điểm

- Tái công việc được phân chia trên nhiều máy tính, thay vì tập trung vào một cụm máy chủ lớn.
- Cho phép người dùng tự nguyện cung cấp tài nguyên cá nhân như sức mạnh tính toán và băng thông để hỗ trợ thu thập dữ liệu.

2.3. Chức năng



a) Thu thập dữ liệu (Data Fetching)

- Thu thập dữ liệu từ các trang web: Hệ thống có khả năng gửi yêu cầu HTTP đến các trang web và thu thập nội dung HTML của các trang. Dữ liệu được thu thập từ các trang web có thể bao gồm văn bản, hình ảnh, liên kết và các tệp khác.
- Xử lý các trang web phức tạp: Hệ thống có thể xử lý các trang web động hoặc phức tạp, chẳng hạn như những trang cần JavaScript để hiển thị nội dung, thông qua việc sử dụng trình duyệt không giao diện (headless browsers).

- Kiểm tra các quy tắc trong robots.txt: Trước khi thu thập dữ liệu từ một trang web, hệ thống kiểm tra tệp robots.txt của trang đó để đảm bảo không vi phạm các quy định về việc thu thập dữ liệu.
- Lập lịch thu thập URL: Hệ thống có thể lập lịch và xác định các URL cần thu thập tiếp theo dựa trên chiến lược như Breadth-First Crawling hoặc Focused Crawling.

b) Quản lý hàng đợi (Queue Management)

- Phân phối công việc thu thập dữ liệu: Hệ thống sử dụng hàng đợi để quản lý và phân phối các URL cần thu thập. Các URL sẽ được đưa vào các hàng đợi và được xử lý theo thứ tự ưu tiên.
- Hỗ trợ hàng đợi phân tán: Hệ thống hỗ trợ các công nghệ hàng đợi phân tán như Redis và RabbitMQ để xử lý các URL trong môi trường phân tán, giúp tối ưu hóa hiệu suất và đảm bảo tính ổn định.

c) Lưu trữ dữ liệu (Data Storage)

- Lưu trữ dữ liệu thu thập được: Sau khi thu thập, dữ liệu sẽ được lưu trữ vào các cơ sở dữ liệu như MongoDB hoặc Elasticsearch, tùy thuộc vào yêu cầu tìm kiếm và truy vấn dữ liệu.
- Mã hóa dữ liệu: Dữ liệu sẽ được mã hóa trước khi lưu trữ để bảo vệ tính riêng tư và bảo mật của thông tin.
- Lưu trữ metadata: Các thông tin mô tả dữ liệu (metadata) sẽ được lưu trữ để phục vụ cho việc tìm kiếm, phân tích và truy cập dễ dàng.

d) Xử lý dữ liệu (Data Processing)

- Làm sạch dữ liệu (Data Cleaning): Hệ thống sẽ loại bỏ dữ liệu không cần thiết hoặc lỗi trong quá trình thu thập, đảm bảo chỉ lưu trữ những thông tin có giá trị.
- Chuyển đổi dữ liệu (Data Transformation): Dữ liệu thu thập được có thể được chuyển đổi thành các định dạng chuẩn để phù hợp với các mục đích lưu trữ hoặc phân tích sau này.
- Phân tích dữ liệu (Data Analysis): Dữ liệu được phân tích để rút ra các thông tin hữu ích, ví dụ như các báo cáo về xu hướng thị trường hoặc các mẫu hành vi của người dùng.

e) Bảo mật và Xác thực (Authentication and Security)

- Quản lý xác thực: Hệ thống sử dụng các mã thông báo như JWT (JSON Web Token) để xác thực người dùng và các dịch vụ. Điều này giúp bảo vệ hệ thống khỏi các truy cập trái phép.
- Quản lý quyền truy cập người dùng: Hệ thống kiểm soát quyền truy cập của người dùng vào các phần khác nhau của hệ thống, giúp đảm bảo rằng chỉ những người dùng hợp lệ và có quyền mới có thể truy cập vào dữ liệu nhạy cảm.
- Bảo mật hệ thống: Các biện pháp bảo mật như mã hóa dữ liệu, hash, và kiểm tra các lỗ hổng bảo mật sẽ được áp dụng để bảo vệ dữ liệu và hệ thống.

2.4. Ứng dụng thực tiễn của Trình thu thập dữ liệu web phân tán

- **Công cụ tìm kiếm.**

Các công cụ tìm kiếm như Google sử dụng trình thu thập dữ liệu web phân tán để quét và lập chỉ mục nội dung từ hàng triệu trang web. Việc sử dụng hệ thống phân tán giúp tối ưu hóa tốc độ thu thập và đảm bảo dữ liệu luôn được cập nhật nhanh chóng.

- **Giám sát giá cả trong thương mại điện tử.**

Các doanh nghiệp hoặc nền tảng thương mại điện tử sử dụng trình thu thập dữ liệu web phân tán để theo dõi giá cả và thông tin sản phẩm từ các đối thủ cạnh tranh, từ đó giúp điều chỉnh chiến lược giá và duy trì tính cạnh tranh.

- **Nghiên cứu thị trường và phân tích dữ liệu lớn.**

Các công ty nghiên cứu thị trường sử dụng trình thu thập dữ liệu phân tán để thu thập dữ liệu từ nhiều nguồn trực tuyến, giúp phân tích hành vi người tiêu dùng, phát hiện xu hướng thị trường và đưa ra các chiến lược kinh doanh phù hợp.

- **Phân tích xu hướng và nội dung mạng xã hội.**

Trình thu thập dữ liệu phân tán giúp theo dõi các nền tảng mạng xã hội (Facebook, Twitter, Instagram) để thu thập thông tin về các xu hướng, sự kiện, và phản hồi của người dùng, từ đó hỗ trợ các chiến dịch marketing hoặc nghiên cứu xã hội.

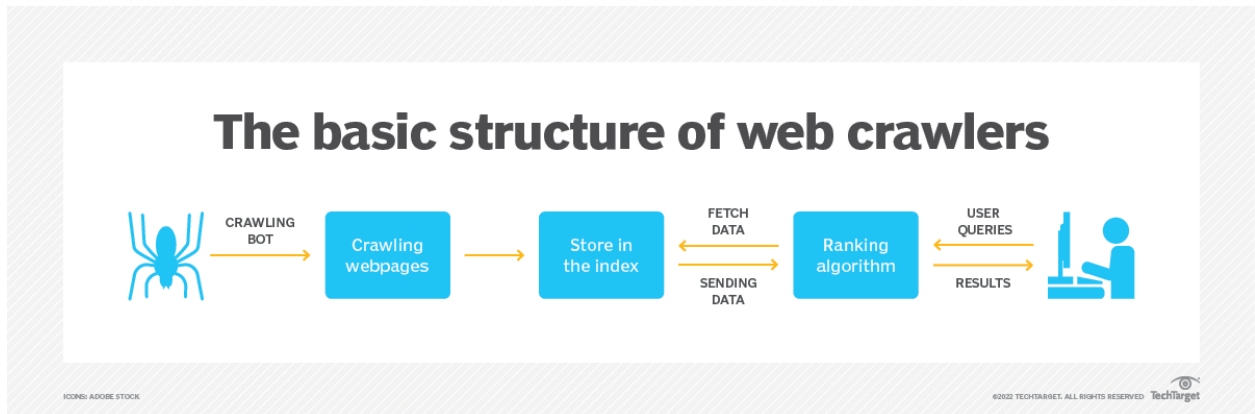
- **Giám sát bảo mật và kiểm tra các lỗ hổng trên website.**

Các công ty bảo mật sử dụng hệ thống thu thập dữ liệu phân tán để kiểm tra các website, phát hiện các lỗ hổng bảo mật, các liên kết bị hỏng hoặc vấn đề với cấu hình HTTPS, giúp bảo vệ website khỏi các mối đe dọa an ninh.

- **Cập nhật thông tin.**

Các nền tảng thông tin tài chính và tin tức sử dụng trình thu thập phân tán để thu thập dữ liệu từ các trang web liên quan đến thị trường chứng khoán, giá cổ phiếu, tin tức kinh tế, nhằm cung cấp thông tin thời gian thực cho người dùng.

2.5. Kiến trúc hệ thống



a) Crawlers (Bộ thu thập)

- Truy cập trang web, trích xuất liên kết và nội dung, sau đó chuyển dữ liệu đến bộ lập chỉ mục(Indexer).
- Hoạt động phân tán trên nhiều máy chủ để tối đa hóa hiệu suất.

b) Indexers (Bộ lập chỉ mục)

- Xử lý nội dung do Crawlers thu thập, phân tích và lập chỉ mục để truy xuất nhanh chóng.
- Quản lý lưu trữ và lập chỉ mục trên cơ sở dữ liệu phân tán.

c) Document Stores (Kho tài liệu)

- Lưu trữ nội dung đã lập chỉ mục trong cơ sở dữ liệu phân tán.
- Tính sẵn sàng cao: Dữ liệu được sao chép qua nhiều nút để tránh mất mát khi một nút gặp sự cố.

2.6. Thách thức

a) Hạn chế tốc độ

- Nhiều website áp dụng giới hạn tốc độ để tránh bị quá tải bởi các yêu cầu từ crawlers.
- Crawlers phải tuân theo chính sách lịch sự để tránh bị chặn hoặc giới hạn truy cập.

b) Nội dung trùng lặp

- Một nội dung có thể xuất hiện trên nhiều URL.
- Crawlers cần phát hiện và xử lý nội dung trùng lặp để tránh làm lộn xộn kết quả tìm kiếm.

3. HIỆU SUẤT CAO VÀ TỐI ƯU HÓA HỆ THỐNG

3.1. Hiệu suất trong Trình thu thập dữ liệu web phân tán

3.1.1. Hiệu suất

a) Khái niệm

- Hiệu suất là một đại lượng được sử dụng để đo lường mức độ hoàn thành theo đúng kế hoạch đã được đặt ra trước đó, nó là thước đo mức độ mà đầu vào được dùng cho một mục đích, nhiệm vụ, chức năng được yêu cầu (đầu ra).
- Hiệu suất của trình thu thập dữ liệu web phân tán đề cập đến khả năng của hệ thống trong việc khám phá, tải xuống, xử lý và lưu trữ nội dung web một cách tối ưu, đồng thời đáp ứng các yêu cầu về tốc độ, phạm vi, độ chính xác, khả năng mở rộng và khả năng chịu lỗi.

b) Các tiêu chí đánh giá hiệu suất

- **Hiệu suất tải xuống:**
 - Tốc độ tải xuống (Download speed): Đo lường lượng dữ liệu mà trình thu thập có thể tải xuống trong một đơn vị thời gian (ví dụ: byte/giây, trang/giây). Tốc độ này phụ thuộc vào nhiều yếu tố như băng thông mạng, tốc độ phản hồi của máy chủ web và hiệu quả của trình thu thập trong việc quản lý kết nối.
 - Số lượng trang được thu thập (Number of pages crawled): Đánh giá khả năng của trình thu thập trong việc khám phá và thu thập một số lượng lớn các trang web trong một khoảng thời gian nhất định.
 - Độ trễ (Latency): Đo lường thời gian cần thiết để tải xuống một trang web. Độ trễ thấp cho thấy hiệu suất tốt.
- **Phạm vi và độ chính xác:**
 - Phạm vi thu thập (Coverage): Đánh giá khả năng của trình thu thập trong việc khám phá và thu thập tất cả các trang web liên quan trong một tập hợp các trang web mục tiêu. Phạm vi rộng đảm bảo không bỏ sót thông tin quan trọng.
 - Độ chính xác (Accuracy): Đánh giá khả năng của trình thu thập trong việc thu thập chính xác nội dung của các trang web, bao gồm cả văn bản, hình ảnh, video và các dữ liệu khác.

- Tính mới (Freshness): Đánh giá khả năng của trình thu thập trong việc cập nhật dữ liệu thường xuyên để phản ánh những thay đổi trên các trang web. Đặc biệt quan trọng đối với các trang web có nội dung động.
- **Khả năng mở rộng và khả năng chịu lỗi:**
 - Khả năng mở rộng (Scalability): Đánh giá khả năng của trình thu thập trong việc xử lý một lượng lớn các trang web và yêu cầu thu thập đồng thời. Trình thu thập có khả năng mở rộng tốt có thể xử lý khối lượng công việc ngày càng tăng một cách hiệu quả.
 - Khả năng chịu lỗi (Fault tolerance): Đánh giá khả năng của trình thu thập trong việc tiếp tục hoạt động ngay cả khi gặp sự cố, chẳng hạn như lỗi mạng, lỗi máy chủ hoặc lỗi phần mềm.
- **Hiệu quả sử dụng tài nguyên:**
 - Mức sử dụng CPU và bộ nhớ (CPU and memory usage): Đánh giá lượng tài nguyên hệ thống mà trình thu thập sử dụng. Hiệu suất tốt đồng nghĩa với việc sử dụng tài nguyên một cách hiệu quả, tránh tình trạng quá tải hệ thống.
 - Băng thông mạng (Network bandwidth): Đánh giá lượng băng thông mạng mà trình thu thập sử dụng. Cần tối ưu để tránh tắc nghẽn mạng.

c) Yêu cầu công nghệ và công cụ hỗ trợ

Để đạt được hiệu suất cao trong trình thu thập dữ liệu web phân tán, cần kết hợp nhiều công nghệ và công cụ hỗ trợ.

- **Yêu cầu công nghệ:**

| | | |
|-------------------------|-----------------------|---|
| Hạ tầng hệ thống | Kiến trúc phân tán | <ul style="list-style-type: none"> – Sử dụng mô hình client-server hoặc peer-to-peer để phân tán tải giữa các node crawler. – Chạy trên các nền tảng hỗ trợ phân tán như Kubernetes, Docker Swarm, hoặc Apache Mesos. |
| | Cơ sở hạ tầng đám mây | <ul style="list-style-type: none"> – Tích hợp với các nhà cung cấp đám mây như AWS, Google Cloud, Azure để linh hoạt mở rộng tài nguyên. |
| | Lưu trữ phân tán | <ul style="list-style-type: none"> – Sử dụng hệ thống lưu trữ dữ liệu phân tán như Hadoop Distributed File System (HDFS) hoặc Amazon S3 để quản lý khối lượng lớn dữ liệu. |

| | | |
|----------------------------|-----------------------------|---|
| Hiệu năng mạng | Cân bằng tải | – Sử dụng Load Balancer (như HAProxy, NGINX) để phân phối đều yêu cầu đến các node crawler. |
| | Tối ưu hóa băng thông | – Tối ưu hóa các yêu cầu HTTP, giảm tải dữ liệu không cần thiết (như nén gzip). – Tận dụng giao thức HTTP/2 hoặc HTTP/3 để tăng tốc độ truyền tải. |
| Xử lý song song | Luồng dữ liệu không đồng bộ | – Sử dụng framework như asyncio (Python), Akka (Java/Scala) để xử lý nhiều yêu cầu đồng thời. |
| | Hàng đợi công việc | – Sử dụng hàng đợi tin nhắn như RabbitMQ, Kafka hoặc Redis để quản lý các tác vụ. |
| Quản lý và giám sát | Công cụ giám sát | – Sử dụng Prometheus, Grafana hoặc ELK Stack (Elasticsearch, Logstash, Kibana) để giám sát hiệu suất hệ thống. |
| | Tự động khôi phục | – Tích hợp tính năng khôi phục tự động (self-healing) khi có lỗi xảy ra. |

• **Công cụ hỗ trợ:**

| | | |
|------------------------------|-----------------------|--|
| Ngôn ngữ lập trình | Python | Framework như Scrapy, BeautifulSoup, hoặc Asyncio để xây dựng crawler. |
| | Node.js | Các thư viện như Puppeteer, Cheerio hoặc Playwright. |
| Framework và thư viện | Scrapy (Python) | Framework phổ biến hỗ trợ thu thập dữ liệu mạnh mẽ, dễ mở rộng. |
| | Puppeteer/ Playwright | Tương thích với các trang web động sử dụng JavaScript. |
| | SQL Database | PostgreSQL hoặc MySQL cho các trường hợp cần truy vấn phức tạp. |

| | | |
|---------------------------------------|----------------|--|
| Hệ thống quản lý cơ sở dữ liệu | NoSQL Database | MongoDB, Cassandra, hoặc DynamoDB để lưu dữ liệu không cấu trúc. |
|---------------------------------------|----------------|--|

3.1.2. Hiệu suất cao

a) Định nghĩa hiệu suất cao trong hệ thống phân tán

Hiệu suất cao trong hệ thống phân tán được định nghĩa là khả năng của hệ thống thực hiện các tác vụ một cách hiệu quả, nhanh chóng và đáng tin cậy trong môi trường phân tán. Các yếu tố chính để đánh giá hiệu suất cao bao gồm:

- Độ trễ thấp (Low Latency): Hệ thống phải phản hồi các yêu cầu trong thời gian ngắn, ngay cả khi có sự phân tán giữa nhiều nút mạng.
- Thông lượng cao (High Throughput): Hệ thống có thể xử lý một lượng lớn yêu cầu hoặc dữ liệu trong một đơn vị thời gian.
- Khả năng mở rộng (Scalability): Hệ thống có thể mở rộng bằng cách thêm tài nguyên (như máy chủ hoặc nút mạng) mà không làm giảm hiệu suất.
- Khả năng chịu lỗi (Fault Tolerance): Hệ thống vẫn duy trì được hoạt động và hiệu suất ngay cả khi có một hoặc nhiều thành phần bị lỗi.
- Cân bằng tải (Load Balancing): Hệ thống phân phối khối lượng công việc đều giữa các nút mạng để tránh tình trạng quá tải ở một số nút.
- Tối ưu sử dụng tài nguyên (Resource Utilization): Hệ thống sử dụng tài nguyên (CPU, RAM, băng thông mạng) một cách hiệu quả để đạt được hiệu suất tối đa.

b) Các yếu tố cấu thành hiệu suất cao: tốc độ, độ tin cậy, chi phí tối ưu

Hiệu suất cao trong hệ thống phân tán được cấu thành từ ba yếu tố chính: tốc độ, độ tin cậy, và chi phí tối ưu.

• Tốc độ (Speed):

Đây là khả năng của hệ thống xử lý nhanh chóng các tác vụ và phản hồi yêu cầu trong thời gian ngắn nhất có thể. Tốc độ bao gồm hai khía cạnh.

- Độ trễ thấp (Low Latency): Thời gian xử lý từ khi yêu cầu được gửi đến khi nhận được phản hồi phải càng nhỏ càng tốt.
- Thông lượng cao (High Throughput): Số lượng tác vụ hoặc yêu cầu mà hệ thống xử lý được trong một đơn vị thời gian phải lớn.

Tốc độ phụ thuộc vào:

- Thiết kế thuật toán hiệu quả.

- Mạng truyền tải nhanh chóng và ít tắc nghẽn.
- Khả năng tối ưu sử dụng tài nguyên hệ thống (CPU, RAM, ổ cứng,...).

- **Độ tin cậy (Reliability):**

Độ tin cậy thể hiện khả năng hệ thống hoạt động chính xác, ổn định và duy trì dịch vụ ngay cả khi gặp sự cố. Điều này bao gồm:

- Khả năng chịu lỗi (Fault Tolerance): Hệ thống vẫn duy trì hoạt động khi có lỗi xảy ra ở một hoặc nhiều thành phần.
- Khả năng tự phục hồi (Self-healing): Tự động khôi phục trạng thái bình thường sau sự cố.
- Tính nhất quán (Consistency): Dữ liệu trên các nút trong hệ thống phải đồng bộ và chính xác.

Độ tin cậy đảm bảo trải nghiệm người dùng không bị gián đoạn và giảm thiểu rủi ro mất dữ liệu.

- **Chi phí tối ưu (Cost Optimization):**

Chi phí tối ưu đề cập đến việc giảm thiểu tài nguyên cần sử dụng mà vẫn đảm bảo hiệu suất cao. Các yếu tố chi phí bao gồm:

- Chi phí phần cứng: Tối ưu số lượng và loại máy chủ, lưu trữ, và thiết bị mạng.
- Chi phí vận hành: Bao gồm điện năng, bảo trì và quản lý hệ thống.
- Chi phí cơ hội: Giảm thiểu thời gian gián đoạn hoặc giảm năng suất do lỗi hệ thống.

Chi phí tối ưu đòi hỏi sự cân bằng giữa hiệu năng và tài nguyên đầu tư, thường đạt được thông qua:

- Sử dụng các mô hình tính toán tiết kiệm như điện toán đám mây.
- Áp dụng chiến lược cân bằng tải thông minh.
- Tối ưu thuật toán và cấu trúc dữ liệu.

c) *Vai trò của hệ thống hiệu suất cao*

Hệ thống hiệu suất cao đóng vai trò quan trọng trong việc đảm bảo các ứng dụng và dịch vụ hoạt động hiệu quả, đáp ứng tốt nhu cầu của người dùng và doanh nghiệp. Dưới đây là những vai trò chính của một hệ thống hiệu suất cao:

| Vai trò | Mô tả |
|---|--|
| Đảm bảo trải nghiệm người dùng tốt | <ul style="list-style-type: none"> Tốc độ phản hồi nhanh: Người dùng kỳ vọng các ứng dụng và dịch vụ trực tuyến phản hồi nhanh chóng, ngay cả khi tải cao. Dịch vụ ổn định: Một hệ thống hiệu suất cao giảm thiểu gián đoạn, đảm bảo tính ổn định trong thời gian dài. Hệ thống hiệu suất cao trực tiếp ảnh hưởng đến sự hài lòng của người dùng và duy trì sự trung thành với dịch vụ. |
| Hỗ trợ xử lý lượng lớn dữ liệu và người dùng | <ul style="list-style-type: none"> Khả năng mở rộng: Các hệ thống hiệu suất cao có thể xử lý hàng triệu yêu cầu hoặc người dùng đồng thời mà không ảnh hưởng đến hiệu năng. Phân tích dữ liệu nhanh: Giúp xử lý và phân tích khối lượng lớn dữ liệu trong thời gian ngắn, đặc biệt quan trọng với các lĩnh vực như tài chính, y tế, và thương mại điện tử. Điều này cho phép doanh nghiệp mở rộng quy mô mà không làm giảm chất lượng dịch vụ. |
| Tăng cường độ tin cậy và sẵn sàng của dịch vụ | <ul style="list-style-type: none"> Hạn chế thời gian ngừng hoạt động: Một hệ thống hiệu suất cao thường tích hợp khả năng chịu lỗi và tự phục hồi, giúp giảm thiểu thời gian ngừng hoạt động. Bảo vệ dữ liệu và tính toàn vẹn: Hệ thống đáng tin cậy giúp bảo vệ dữ liệu và đảm bảo rằng các giao dịch hoặc thao tác được thực hiện chính xác. Vai trò này đặc biệt quan trọng trong các ngành yêu cầu độ tin cậy cao như ngân hàng, y tế, và viễn thông. |
| Tối ưu chi phí vận hành | <ul style="list-style-type: none"> Sử dụng hiệu quả tài nguyên: Hệ thống hiệu suất cao giúp giảm chi phí phần cứng, điện năng và các chi phí vận hành khác thông qua việc tối ưu hóa sử dụng tài nguyên. Giảm chi phí cơ hội: Hạn chế sự cố hoặc chậm trễ giúp doanh nghiệp tránh mất doanh thu hoặc uy tín. |

| | |
|--|---|
| | – Điều này hỗ trợ doanh nghiệp duy trì lợi nhuận trong khi vẫn cung cấp dịch vụ chất lượng cao. |
|--|---|

d) Tầm quan trọng của hiệu suất cao trong Trình thu thập dữ liệu web phân tán

Hiệu suất cao là yếu tố then chốt cho sự thành công của trình thu thập dữ liệu web phân tán. Nó không chỉ giúp thu thập dữ liệu nhanh chóng mà còn đảm bảo chất lượng dữ liệu, tiết kiệm chi phí và nâng cao hiệu quả của các ứng dụng sử dụng dữ liệu web. Việc đầu tư vào việc tối ưu hiệu suất của trình thu thập là một đầu tư quan trọng và mang lại nhiều lợi ích dài hạn.

- Thu thập dữ liệu quy mô lớn:
 - Xử lý khối lượng dữ liệu khổng lồ: Web ngày càng mở rộng với hàng tỷ trang web. Hiệu suất cao cho phép trình thu thập xử lý và thu thập một lượng lớn dữ liệu trong thời gian hợp lý.
 - Đáp ứng nhu cầu thu thập dữ liệu ngày càng tăng: Nhu cầu về dữ liệu web liên tục tăng lên trong nhiều lĩnh vực như tìm kiếm, phân tích thị trường, giám sát mạng xã hội... Hiệu suất cao giúp đáp ứng kịp thời nhu cầu này.
- Tiết kiệm thời gian và chi phí:

Trình thu thập dữ liệu web phân tán thường hoạt động trên quy mô lớn, truy cập hàng triệu hoặc thậm chí hàng tỷ trang web. Hiệu suất cao giúp:

 - Tăng tốc quá trình thu thập dữ liệu, đảm bảo dữ liệu được thu thập trong thời gian ngắn nhất.
 - Đáp ứng kịp thời các yêu cầu phân tích hoặc sử dụng dữ liệu, đặc biệt trong các lĩnh vực đòi hỏi tính thời gian, như theo dõi tin tức, giá cả sản phẩm, hoặc xu hướng thị trường.
 - Tối ưu hóa chi phí vận hành: Việc sử dụng tài nguyên hiệu quả hơn (CPU, bộ nhớ, băng thông) giúp giảm chi phí vận hành hệ thống.
- Cải thiện chất lượng dữ liệu:
 - Thu thập dữ liệu mới và cập nhật: Hiệu suất cao cho phép thu thập dữ liệu thường xuyên hơn, đảm bảo tính mới và cập nhật của dữ liệu, đặc biệt quan trọng đối với các trang web có nội dung động.
 - Giảm thiểu dữ liệu lỗi thời: Việc thu thập dữ liệu nhanh chóng giúp giảm thiểu lượng dữ liệu lỗi thời, không còn giá trị sử dụng.
- Nâng cao hiệu quả ứng dụng:

- Cung cấp dữ liệu kịp thời cho các ứng dụng: Các ứng dụng dựa trên dữ liệu web như công cụ tìm kiếm, hệ thống phân tích dữ liệu... cần dữ liệu mới và đầy đủ. Hiệu suất cao của trình thu thập đảm bảo cung cấp dữ liệu kịp thời cho các ứng dụng này.
- Cải thiện trải nghiệm người dùng: Dữ liệu mới và chính xác giúp cải thiện trải nghiệm người dùng khi sử dụng các ứng dụng dựa trên dữ liệu web.

3.2. Các yếu tố cần tối ưu hóa hiệu suất

3.2.1. Phân tải công việc (Load Balancing)

a) Giới thiệu

Phân tải công việc (Load Balancing) là kỹ thuật phân phối đều các yêu cầu công việc hoặc lưu lượng truy cập đến các tài nguyên trong hệ thống (như máy chủ, nút mạng) để đảm bảo hiệu suất, độ tin cậy và khả năng mở rộng. Đây là yếu tố cốt lõi trong việc duy trì hiệu suất cao của hệ thống phân tán.

Sử dụng các thuật toán phân tải thông minh, như Round Robin hoặc Least Connections, để tránh tình trạng quá tải tại các node.

Các thành phần trong hệ thống phân tải:

- Load Balancer: Phần cứng hoặc phần mềm chịu trách nhiệm phân phối lưu lượng.
- Các nút mạng (Nodes): Các máy chủ hoặc tài nguyên thực hiện công việc.
- Cơ chế giám sát: Theo dõi trạng thái của các nút để đảm bảo phân tải chính xác.

b) Tầm quan trọng của phân tải công việc

- Tránh quá tải: Ngăn ngừa tình trạng một nút bị quá tải trong khi các nút khác rảnh rỗi.
- Tăng hiệu suất: Phân phối đồng đều giúp tối ưu sử dụng tài nguyên hệ thống, giảm độ trễ và tăng thông lượng.
- Tăng độ tin cậy: Nếu một nút gặp sự cố, các yêu cầu có thể được chuyển đến nút khác.
- Hỗ trợ mở rộng: Khi thêm tài nguyên mới (như máy chủ), hệ thống vẫn có thể phân phối công việc hiệu quả.

c) Các thuật toán phân tải công việc phổ biến

| Tên thuật toán | Cách hoạt động | Ưu điểm | Nhược điểm |
|-----------------------|--|--|--|
| Round Robin | Phân phối các yêu cầu đến các nút theo thứ tự tuần tự, lặp lại từ đầu khi hết danh sách. | <ul style="list-style-type: none"> Đơn giản và dễ triển khai. Phù hợp với các hệ thống có các nút có cấu hình tương tự nhau. | Không tính đến trạng thái hiện tại của các nút (ví dụ: tải trọng hoặc kết nối đang hoạt động). |
| Least Connections | Chuyển yêu cầu đến nút có số lượng kết nối hoạt động ít nhất. | Hiệu quả trong các hệ thống mà khối lượng công việc không đồng đều giữa các yêu cầu. | Cần theo dõi trạng thái của các nút, đòi hỏi tài nguyên quản lý cao hơn. |
| Least Response Time | Chuyển yêu cầu đến nút có thời gian phản hồi trung bình thấp nhất. | Cải thiện tốc độ phản hồi tổng thể của hệ thống. | Cần theo dõi thời gian phản hồi của từng nút, yêu cầu công cụ giám sát phức tạp. |
| Weighted Round Robin | Gán trọng số (weight) cho mỗi nút dựa trên khả năng xử lý của chúng. Nút mạnh hơn sẽ nhận nhiều yêu cầu hơn. | Tận dụng tối đa năng lực của từng nút trong hệ thống không đồng nhất. | Cần tính toán và cấu hình trọng số hợp lý. |
| IP Hash | Sử dụng giá trị băm của địa chỉ IP khách hàng để xác định nút nào sẽ xử lý yêu cầu. | Phù hợp với các ứng dụng yêu cầu duy trì phiên làm việc (session persistence). | Phân phối không đều nếu tập hợp địa chỉ IP khách hàng không đồng đều. |

d) Tối ưu hóa băng thông mạng khi giao tiếp giữa master và worker

Tối ưu hóa băng thông mạng trong phân tải công việc (Load Balancing)

Trong hệ thống phân tán, việc giao tiếp giữa master (điều phối) và worker (nút xử lý) có thể tiêu tốn nhiều băng thông mạng, đặc biệt trong các hệ thống có khối lượng công việc lớn hoặc yêu cầu dữ liệu phức tạp. Để tối ưu hóa băng thông mạng trong phân tải công việc, cần áp dụng các chiến lược và phương pháp thông minh để giảm tải lượng dữ liệu truyền, tối ưu luồng thông tin và đảm bảo hiệu quả.

Các chiến lược tối ưu hóa băng thông mạng:

| Chiến lược | Mô tả |
|---|---|
| Giảm thiểu lượng dữ liệu truyền giữa master và worker | <ul style="list-style-type: none">Chỉ truyền thông tin cần thiết: Master chỉ nên gửi các thông tin tối thiểu như:<ul style="list-style-type: none">Vị trí dữ liệu.Chỉ thị thực hiện (instruction).Trạng thái cần cập nhật.Dữ liệu thô (raw data) nên được truyền trực tiếp từ nguồn đến worker thay vì qua master.Nén dữ liệu: Sử dụng các thuật toán nén như Gzip hoặc Snappy để giảm kích thước dữ liệu khi truyền qua mạng.Sử dụng chỉ mục hoặc tham chiếu: Thay vì gửi toàn bộ dữ liệu, chỉ gửi tham chiếu hoặc chỉ mục trỏ tới dữ liệu được lưu trong một kho chung mà worker có thể truy cập. |
| Áp dụng kiến trúc phân cấp cho load balancer | <ul style="list-style-type: none">Master phân cấp (Hierarchical Master): Thay vì chỉ có một master điều phối toàn bộ hệ thống, có thể sử dụng kiến trúc master phân cấp, nơi:<ul style="list-style-type: none">Master cấp cao (global master) phân chia công việc cho các master cấp thấp (regional master). |

| | |
|---|--|
| | <ul style="list-style-type: none"> – Master cấp thấp điều phối công việc trực tiếp cho worker trong vùng của nó. – Kiến trúc này giúp giảm thiểu băng thông mạng giữa master và worker khi số lượng worker lớn. <ul style="list-style-type: none"> • Edge Computing: Đưa các tác vụ nhẹ hoặc sơ bộ (preprocessing) ra gần worker để giảm lượng dữ liệu cần truyền giữa master và worker trung tâm |
| Cân nhắc chiến lược phân tải thông minh | <ul style="list-style-type: none"> • Static Load Balancing: Phân chia công việc cố định từ trước, hạn chế việc liên lạc thường xuyên giữa master và worker trong quá trình xử lý. • Dynamic Load Balancing với nhóm nhỏ (Clustered Dynamic Balancing): Phân chia công việc động dựa trên cụm worker thay vì từng worker riêng lẻ. Điều này làm giảm số lần giao tiếp cần thiết. • Task Bundling: Gửi các yêu cầu công việc theo nhóm (batches) thay vì từng yêu cầu riêng lẻ, giảm số lượng gói tin và tối ưu băng thông. |
| Tối ưu hóa giao thức và cơ chế truyền thông | <ul style="list-style-type: none"> • Sử dụng giao thức hiệu quả: <ul style="list-style-type: none"> – Ưu tiên các giao thức nhẹ như UDP cho các tác vụ yêu cầu phản hồi nhanh mà không cần tính toán vẹn dữ liệu cao. – Áp dụng gRPC hoặc HTTP/2 để tận dụng tính năng nén và truyền song song. • Tối ưu thời gian giữ kết nối (Connection Pooling): Giữ các kết nối mạng lâu dài giữa master và worker thay vì tạo mới mỗi lần giao tiếp. |

3.2.2. Quản lý bộ nhớ hiệu quả

a) Sử dụng bộ nhớ đệm (cache) để giảm số lượng truy cập đĩa

Sử dụng bộ nhớ đệm (cache) là một trong những kỹ thuật quản lý bộ nhớ hiệu quả, giúp giảm tải cho việc truy cập dữ liệu từ đĩa cứng hoặc các nguồn dữ liệu chậm khác. Dưới đây là một số điểm quan trọng về cách sử dụng cache:

- Khái niệm bộ nhớ đệm (cache):

- Bộ nhớ đệm là một vùng lưu trữ tạm thời, nhanh, thường được đặt gần CPU hoặc trong RAM.
- Dữ liệu thường xuyên được truy cập hoặc dự đoán sẽ được truy cập được lưu trữ ở đây, giúp tăng tốc quá trình xử lý.
- Lợi ích của việc sử dụng cache:
 - Tăng tốc độ truy cập dữ liệu: Truy cập bộ nhớ đệm nhanh hơn rất nhiều so với truy cập đĩa cứng.
 - Giảm tải cho hệ thống I/O: Hệ thống không cần thực hiện quá nhiều thao tác đọc/ghi đĩa.
 - Cải thiện hiệu năng: Giảm độ trễ và tăng thông lượng cho các ứng dụng.
- Cách sử dụng cache hiệu quả:
 - Phân tích dữ liệu thường xuyên truy cập: Xác định dữ liệu nào được sử dụng thường xuyên hoặc theo mẫu để lưu trữ trong bộ nhớ đệm.
 - Áp dụng các thuật toán quản lý cache:
 - LRU (Least Recently Used): Loại bỏ dữ liệu ít được sử dụng gần đây nhất.
 - FIFO (First In, First Out): Loại bỏ dữ liệu được lưu vào đầu tiên khi cache đầy.
 - LFU (Least Frequently Used): Loại bỏ dữ liệu ít được sử dụng nhất.
 - Cân bằng kích thước cache: Đảm bảo kích thước bộ nhớ đệm phù hợp với tài nguyên hệ thống để tránh tiêu tốn quá nhiều RAM hoặc làm giảm hiệu năng chung.
- Ứng dụng thực tế:
 - Hệ điều hành: Hệ điều hành sử dụng cache để lưu trữ dữ liệu đĩa và các trang bộ nhớ.
 - Cơ sở dữ liệu: Các hệ thống quản lý cơ sở dữ liệu như MySQL, PostgreSQL dùng cache để lưu trữ các kết quả truy vấn.
 - Web application: Sử dụng cache trong trình duyệt, CDN, hoặc dịch vụ như Redis, Memcached để lưu trữ kết quả tạm thời của các trang web hoặc API.

b) Phân mảnh dữ liệu "URL đã thấy" giữa các node crawler để tối ưu không gian lưu trữ

Phân mảnh dữ liệu "URL đã thấy" giữa các node crawler là một chiến lược hiệu quả để tối ưu hóa không gian lưu trữ và cải thiện hiệu suất của hệ thống crawling. Đây là một số cách tiếp cận và kỹ thuật bạn có thể sử dụng:

- Khái niệm cơ bản:
- Dữ liệu "URL đã thấy" (Visited URLs): Danh sách hoặc tập hợp các URL đã được crawler truy cập, giúp tránh truy cập lại và giảm trùng lặp.
- Phân mảnh: Chia nhỏ dữ liệu này và phân phối chúng giữa các node crawler để tối ưu hóa tài nguyên.

- Cách phân mảnh:

| Tên | Nguyên lý | Ưu điểm | Nhược điểm |
|--------------------------|--|---|--|
| Hash-based Partitioning | Sử dụng một hàm băm (hash function) để ánh xạ URL thành một giá trị băm duy nhất, sau đó gán URL đó cho một node dựa trên giá trị băm. | <ul style="list-style-type: none"> – Phân phối đồng đều URL giữa các node. – Truy vấn và kiểm tra tính tồn tại của URL nhanh chóng. | Khi thêm hoặc xóa node, cần sử dụng Consistent Hashing để giảm tác động đến dữ liệu hiện có. |
| Range-based Partitioning | Phân chia không gian URL thành các dải (range) dựa trên một tiêu chí, chẳng hạn như chữ cái đầu tiên, độ dài URL, hoặc miền (domain). | Đơn giản, dễ thực hiện. | Phân phối dữ liệu không đồng đều nếu URL tập trung vào một số dải cụ thể. |
| Sharding theo domain | Gán toàn bộ các URL thuộc một miền (domain) hoặc miền phụ (subdomain) cho một node cụ thể. | <ul style="list-style-type: none"> – Giảm độ phức tạp khi xử lý dữ liệu từ một miền – Tăng khả năng tối ưu hóa và nén dữ liệu URL.. | Cần xử lý các miền lớn có quá nhiều URL. |

- Quản lý dữ liệu phân mảnh:

| Tên | Mô tả |
|------------------------------|---|
| Distributed Hash Table (DHT) | <ul style="list-style-type: none"> – Sử dụng cấu trúc DHT để lưu trữ và truy xuất URL trên nhiều node. – Mỗi node chỉ lưu trữ một phần nhỏ của dữ liệu và biết cách định tuyến truy vấn đến node phù hợp. |
| Bloom Filter | <ul style="list-style-type: none"> – Sử dụng Bloom Filter để kiểm tra nhanh xem URL đã được thấy hay chưa mà không cần lưu trữ toàn bộ dữ liệu. – Bloom Filter có thể triển khai trên từng node và phân phối theo chiến lược phân mảnh. |
| Ghi log phân tán | Dữ liệu "URL đã thấy" có thể được ghi vào log phân tán như Apache Kafka để đồng bộ hóa giữa các node và xử lý truy vấn. |

- Ưu và nhược điểm của phân mảnh:

| Ưu điểm | Nhược điểm |
|--|--|
| <ul style="list-style-type: none"> – Tối ưu không gian lưu trữ: Mỗi node chỉ lưu trữ một phần dữ liệu, giảm áp lực lên từng node. – Tăng khả năng mở rộng: Khi hệ thống mở rộng, chỉ cần thêm node để xử lý thêm dữ liệu. – Hiệu suất cao: Phân tán khối lượng công việc và truy vấn giữa các node. | <ul style="list-style-type: none"> – Phức tạp trong quản lý: Cần cơ chế phân mảnh, tái phân mảnh khi thêm hoặc xóa node. – Đồng bộ hóa dữ liệu: Dữ liệu phân tán yêu cầu cơ chế đảm bảo tính nhất quán hoặc xử lý lỗi. |

c) Lưu trữ dữ liệu theo batch và xử lý dữ liệu lớn

Lưu trữ dữ liệu theo batch (lô dữ liệu) và xử lý dữ liệu lớn (big data) là hai khái niệm quan trọng trong việc quản lý và phân tích dữ liệu hiệu quả khi đối mặt với khối lượng dữ liệu khổng lồ.

* Lưu trữ dữ liệu theo batch:

- Khái niệm batch storage:
 - Batching: Dữ liệu được lưu trữ và xử lý thành từng lô lớn thay vì xử lý từng bản ghi riêng lẻ.
 - Ứng dụng:
 - Giảm tần suất truy cập vào cơ sở dữ liệu hoặc hệ thống lưu trữ.
 - Tối ưu hóa hiệu suất trong các hệ thống xử lý khối lượng dữ liệu lớn.
- Kỹ thuật lưu trữ dữ liệu theo batch:
 - Sử dụng hệ thống lưu trữ chuyên dụng:
 - HDFS (Hadoop Distributed File System): Lý tưởng cho việc lưu trữ dữ liệu lớn theo batch trên hệ thống phân tán.
 - Amazon S3, Google Cloud Storage, Azure Blob Storage: Giải pháp lưu trữ đám mây phổ biến.
 - Data Lake: Một kho lưu trữ dữ liệu thô với khả năng lưu trữ dữ liệu theo batch.
 - Gộp dữ liệu trước khi lưu trữ:
 - Thu thập dữ liệu từ nhiều nguồn vào bộ nhớ tạm.
 - Ghi dữ liệu thành một lô lớn để giảm tải và tiết kiệm không gian lưu trữ.
 - Tổ chức dữ liệu:
 - Sắp xếp dữ liệu theo thời gian (partition theo ngày, tháng, năm).
 - Tối ưu hóa truy vấn bằng cách sử dụng format file hiệu quả như Parquet hoặc ORC.
- Ưu và nhược điểm:

| Ưu điểm | Nhược điểm |
|---|--|
| <ul style="list-style-type: none">– Tăng tốc độ xử lý khi làm việc với dữ liệu lớn.– Giảm chi phí lưu trữ và truy vấn so với xử lý dữ liệu nhỏ lẻ. | Độ trễ cao hơn so với xử lý dữ liệu theo thời gian thực (real-time). |

*Xử lý dữ liệu lớn:

- Khái niệm big data processing:
 - Big Data: Dữ liệu có khối lượng lớn, tốc độ tạo ra nhanh, và đa dạng về loại hình (3V: Volume, Velocity, Variety).

- Mục tiêu:
 - Phân tích, trích xuất thông tin từ dữ liệu lớn.
 - Đưa ra quyết định kinh doanh, cải thiện quy trình hoặc dự đoán xu hướng.
- Cách tiếp cận xử lý big data
 - Batch Processing
 - Đặc điểm: Xử lý dữ liệu lớn thành các lô trong khoảng thời gian nhất định.
 - Công cụ phổ biến:
 - > Apache Hadoop: Framework phổ biến cho xử lý batch với mô hình MapReduce.
 - > Apache Spark (Batch Mode): Xử lý batch nhanh hơn Hadoop nhờ sử dụng bộ nhớ (in-memory).
 - Stream Processing (Xử lý luồng)
 - Đặc điểm: Xử lý dữ liệu ngay khi chúng đến hệ thống.
 - Công cụ phổ biến:
 - > Apache Kafka + Spark Streaming/Flink/Storm: Phân tích dữ liệu theo thời gian thực.
 - So sánh với batch: Tăng độ nhanh nhạy, nhưng cần cơ sở hạ tầng mạnh hơn.
 - Hybrid Processing
 - Kết hợp batch và streaming để xử lý dữ liệu theo cả lô và luồng.
- Các bước xử lý dữ liệu lớn
 - Thu thập dữ liệu:
 - Sử dụng công cụ như Kafka, Flume, hoặc Kinesis để thu thập dữ liệu từ nhiều nguồn.
 - Lưu trữ dữ liệu:
 - Lưu trữ dữ liệu trong HDFS, Data Lake, hoặc cơ sở dữ liệu NoSQL như MongoDB, Cassandra.
 - Xử lý dữ liệu:
 - Batch: Spark, Hadoop.
 - Streaming: Flink, Spark Streaming.
 - Trực quan hóa và phân tích:
 - Dùng công cụ như Tableau, Power BI, hoặc Dashboards tùy chỉnh để trình bày dữ liệu.

- Ưu và nhược điểm

| Ưu điểm | Nhược điểm |
|--|--|
| <ul style="list-style-type: none"> – Khả năng xử lý dữ liệu khổng lồ trong thời gian ngắn. – Khả năng mở rộng hệ thống theo nhu cầu. | <p>Yêu cầu phần cứng mạnh và quản trị hệ thống phức tạp.</p> |

- Ứng dụng thực tế

- Thương mại điện tử: Phân tích hành vi khách hàng từ dữ liệu giao dịch và log truy cập.
- Tài chính: Phát hiện gian lận từ luồng giao dịch thời gian thực.
- Y tế: Lưu trữ và phân tích dữ liệu bệnh án, hình ảnh y khoa.

3.2.3. Chịu lỗi

a) Tự động phục hồi khi một hoặc nhiều worker gặp lỗi

Chịu lỗi (Fault Tolerance) là một trong những yếu tố quan trọng trong việc tối ưu hóa hiệu suất của hệ thống phân tán hoặc các ứng dụng quy mô lớn. Để đạt được khả năng chịu lỗi hiệu quả và tự động phục hồi, ta cần tập trung vào các yếu tố sau:

- Phân phối công việc và tái phân phối tự động
 - Sử dụng cơ chế quản lý worker để phát hiện các worker gặp lỗi (thông qua heartbeat hoặc health checks).
 - Hỗ trợ tái phân phối công việc từ worker lỗi sang các worker khác mà không ảnh hưởng đến toàn bộ hệ thống.
- Hệ thống lưu trạng thái (State Management)
 - Checkpointing: Lưu trạng thái trung gian của hệ thống theo chu kỳ để đảm bảo khôi phục dữ liệu từ trạng thái gần nhất khi xảy ra sự cố.
 - Event Sourcing: Lưu lại toàn bộ lịch sử các sự kiện để tái hiện trạng thái hệ thống sau lỗi.
- Cơ chế tự động phát hiện lỗi
 - Áp dụng các cơ chế như leader election (chọn một leader mới nếu leader hiện tại gặp lỗi).
 - Sử dụng công cụ giám sát như Prometheus, Grafana, hoặc tích hợp với dịch vụ Cloud như AWS CloudWatch để theo dõi trạng thái của hệ thống.
- Tự động phục hồi (Self-Healing)

- Kích hoạt lại các worker bị lỗi thông qua orchestrator như Kubernetes.
- Sử dụng các chiến lược như restart policy hoặc auto-scaling để thay thế các worker không hoạt động.
- Kiểm thử chịu lỗi (Fault Injection Testing)
 - Sử dụng các công cụ như Chaos Monkey hoặc Gremlin để giả lập các lỗi và đảm bảo hệ thống có thể phục hồi hiệu quả.
- Tích hợp với các mô hình sao lưu và phục hồi
 - Thiết lập backup cho dữ liệu quan trọng (hot backup, cold backup).
 - Sử dụng các cơ chế khôi phục nhanh (disaster recovery).

b) Đồng bộ định kỳ trạng thái hệ thống lên đĩa để giảm thiểu mất mát dữ liệu khi có sự cố

Đồng bộ định kỳ trạng thái hệ thống lên đĩa là một chiến lược quan trọng để tối ưu hóa hiệu suất và đảm bảo khả năng chịu lỗi (Fault Tolerance).

- Checkpointing
 - Mô tả: Checkpointing là kỹ thuật lưu trạng thái hiện tại của hệ thống (bao gồm dữ liệu, cấu hình, hoặc trạng thái ứng dụng) lên đĩa hoặc bộ nhớ lâu dài để sử dụng khi cần khôi phục sau sự cố.
 - Ưu điểm:
 - Giảm thiểu mất mát dữ liệu bằng cách lưu trữ dữ liệu định kỳ.
 - Tăng tốc độ khôi phục bằng cách tái sử dụng trạng thái đã lưu thay vì khởi động lại từ đầu.
- Tần suất đồng bộ:
 - Tần suất đồng bộ cần được cân nhắc để cân bằng giữa hiệu năng và rủi ro mất mát dữ liệu.
 - Ví dụ: Checkpoint mỗi 5 phút hoặc sau một số lượng giao dịch nhất định.
- Write-Ahead Logging (WAL)
 - Mô tả: Ghi log trước khi thực hiện thay đổi trạng thái chính. Điều này đảm bảo rằng nếu hệ thống gặp sự cố, các thay đổi có thể được phát lại từ log để khôi phục trạng thái.
 - Ứng dụng:
 - Các hệ quản trị cơ sở dữ liệu như PostgreSQL, MySQL.
 - Distributed file systems (HDFS, etc.).
 - Lợi ích:

- Hạn chế mất dữ liệu xuống mức tối thiểu.
 - Tăng tính toàn vẹn dữ liệu khi hệ thống bị lỗi.
- Snapshot
 - Mô tả: Chụp nhanh (snapshot) là việc lưu trạng thái đầy đủ của toàn bộ hệ thống hoặc dữ liệu quan trọng vào một thời điểm cụ thể.
 - Ưu điểm:
 - Phục hồi nhanh chóng từ trạng thái snapshot.
 - Hữu ích cho hệ thống lưu trữ hoặc cơ sở dữ liệu phân tán.
 - Ứng dụng:
 - VMware, Docker, và Kubernetes cung cấp cơ chế snapshot cho ứng dụng hoặc container.
 - Các hệ thống phân tán như Cassandra, Elasticsearch.
- Công nghệ đồng bộ dữ liệu
 - Sử dụng cơ chế đồng bộ mạnh mẽ:
 - `fsync()` hoặc `fdatasync()`: Đảm bảo dữ liệu thực sự được ghi xuống đĩa trước khi báo hoàn tất.
 - Memory-Mapped Files: Hỗ trợ đồng bộ hóa hiệu quả giữa bộ nhớ và đĩa.
 - Giảm overhead của I/O:
 - Sử dụng SSD để tăng tốc độ ghi dữ liệu.
 - Áp dụng kỹ thuật batching để giảm tần suất ghi nhỏ lẻ.
- Distributed State Checkpointing
 - Trong hệ thống phân tán:
 - Đồng bộ trạng thái từ các node (worker) khác nhau lên một lưu trữ trung tâm (ví dụ: HDFS, S3).
 - Sử dụng framework hỗ trợ checkpoint tự động như Apache Flink, Spark Streaming.
 - Atomic Consistency:
 - Đảm bảo tính nhất quán giữa các checkpoint của các node khác nhau để tránh mất mát hoặc sai lệch dữ liệu.
- Kỹ thuật lưu trữ bền vững (Durable Storage)
 - Cơ sở hạ tầng lưu trữ:
 - Sử dụng các giải pháp lưu trữ bền vững như Amazon S3, Google Cloud Storage, hoặc Azure Blob Storage để lưu checkpoint.

- Replication:
 - Sao lưu dữ liệu theo chiến lược multi-region replication để bảo vệ trước lỗi phần cứng hoặc thiên tai.
- Tối ưu hóa tần suất và kích thước checkpoint
 - Cân bằng chi phí và hiệu năng:
 - Tạo checkpoint thường xuyên sẽ tăng tính an toàn nhưng gây áp lực lớn lên hiệu năng hệ thống.
 - Giảm kích thước checkpoint bằng cách chỉ lưu dữ liệu thay đổi (incremental checkpoint).
- Công cụ hỗ trợ
 - Framework hỗ trợ checkpoint:
 - Flink: Tự động checkpoint trạng thái ứng dụng và phục hồi từ sự cố.
 - Kafka Streams: Hỗ trợ lưu trạng thái ứng dụng streaming.
 - Công cụ lưu trữ log:
 - Etcd, Zookeeper: Lưu trạng thái phân tán của hệ thống và hỗ trợ phục hồi nhanh chóng.
- Kiểm tra và giám sát
 - Định kỳ kiểm tra tính toàn vẹn của checkpoint (verify integrity).
 - Giám sát hiệu suất ghi checkpoint để tối ưu hóa tần suất và dung lượng.

PHẦN B. NỘI DUNG CHÍNH

4. MÔ HÌNH TCP/IP

4.1. Tầng ứng dụng

- FastAPI:
 - FastAPI là một framework ở lớp ứng dụng, tạo ra các REST API phục vụ giao tiếp giữa client và server (crawler node).
 - Các endpoint như /crawl, /status xử lý yêu cầu từ client và trả về dữ liệu dưới dạng JSON.
- HTTP và REST API:
 - Hệ thống sử dụng giao thức HTTP để truyền dữ liệu (GET, POST).
 - REST API đảm bảo các endpoint được thiết kế theo tiêu chuẩn, dễ dàng mở rộng.
- Tuân thủ Robots.txt:
 - Hàm can_fetch_url phân tích tệp robots.txt của website mục tiêu, một quy tắc thuộc lớp ứng dụng giúp crawler không truy cập vào các khu vực bị cấm.
- JSON Format:
 - Dữ liệu được truyền qua API sử dụng định dạng JSON, một chuẩn trao đổi dữ liệu phổ biến ở lớp ứng dụng.

4.2. Tầng giao vận

- TCP (Transmission Control Protocol):
 - Hệ thống sử dụng HTTP, một giao thức hoạt động trên TCP. TCP đảm bảo dữ liệu (gói tin HTTP) được truyền tải một cách đáng tin cậy, đúng thứ tự.
 - Nếu một yêu cầu bị mất hoặc không thành công, TCP sẽ đảm bảo việc tái truyền gói tin.
- Cơ chế retry (khởi động lại node):
 - check_node_health phát hiện lỗi của một node crawler và cố gắng khởi động lại. Trong mạng thực tế, đây là cách ứng dụng xử lý các vấn đề về kết nối và truyền dữ liệu.
- Port Management:

- Các cổng từ 9000-9005 được định nghĩa để các node crawler hoạt động độc lập.
- Transport Layer định tuyến các gói tin đến đúng ứng dụng qua cổng mạng.

4.3. Tầng mạng

- Địa chỉ IP:
 - Mặc dù code sử dụng localhost (127.0.0.1) để phát triển, trong môi trường thực tế, địa chỉ IP sẽ được sử dụng để định tuyến gói tin giữa client và các node crawler.
- HTTP Requests:
 - Mỗi yêu cầu API (HTTP GET/POST) được đóng gói trong các gói IP để truyền qua mạng.
- Khả năng triển khai phân tán:
 - Khi triển khai trên mạng rộng (WAN), các crawler node có thể ở các địa chỉ IP khác nhau. Lớp Internet sẽ định tuyến gói tin đến đúng địa chỉ.

4.4. Tầng vật lý

- Tài nguyên phần cứng
 - CPU: Giám sát qua `psutil.cpu_percent()` trong health check
 - RAM: Theo dõi qua `psutil.virtual_memory().percent`
 - Disk I/O: Lưu trữ dữ liệu crawl vào thư mục "storage" trên ổ đĩa thông qua `hard_disk_store()`
- Mạng
 - Sử dụng TCP/IP qua các cổng từ 5000 trở lên cho mỗi node
 - Giao tiếp HTTP/HTTPS bất đồng bộ giữa các node qua `aiohttp`
 - Load balancing giữa các node theo round-robin
 - Timeout 5s cho health check và 3s cho node status check
- Quản lý lỗi phần cứng
 - Health monitor kiểm tra định kỳ 10s/lần
 - Tự động khởi động lại node khi phát hiện lỗi
 - Giới hạn số lượng node tối đa (`MAX_PORTS = 6`)

5. TÍNH BẢO MẬT

Các vấn đề bảo mật trong Distributed Web Crawler:

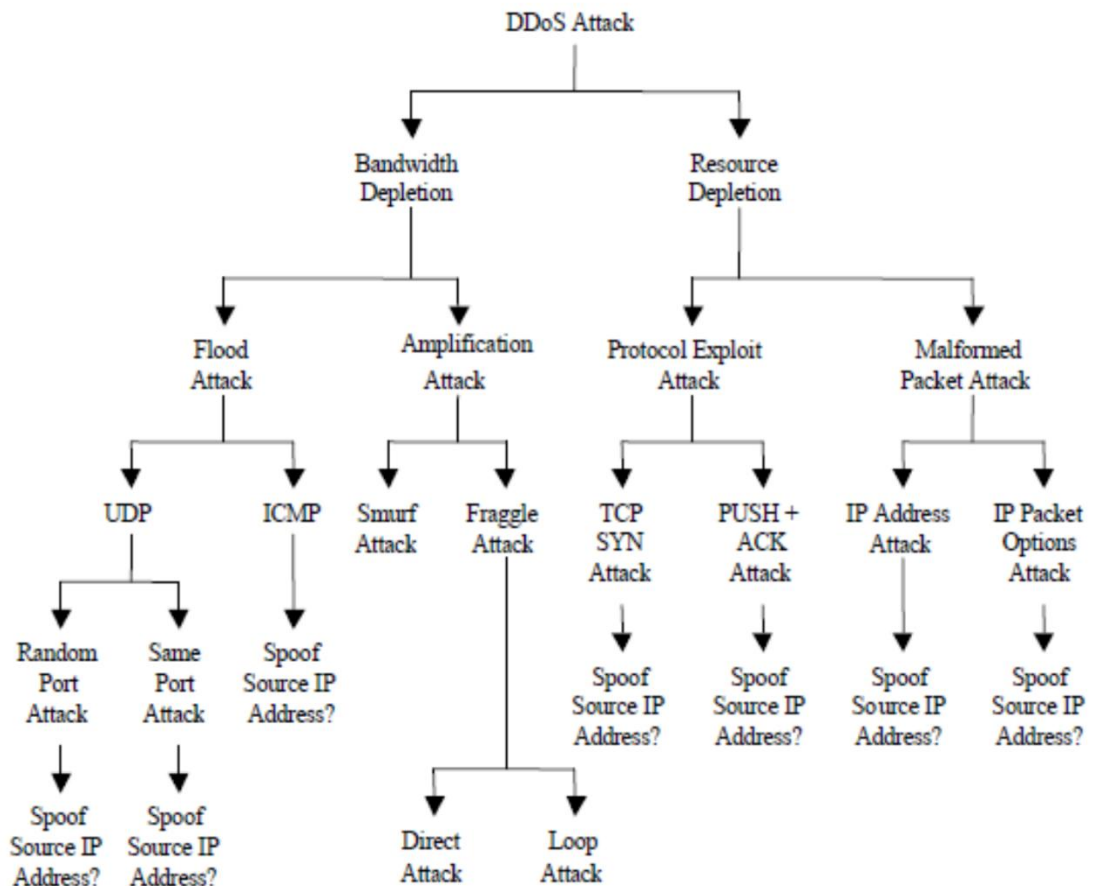
DDoS (Distributed Denial of Service) là một loại tấn công từ chối dịch vụ nhằm làm quá tải một máy chủ, một mạng máy tính hoặc một dịch vụ trực tuyến bằng cách gửi một lượng lớn lưu lượng truy cập giả mạo từ nhiều nguồn khác nhau.

Nguy cơ bị phát hiện và chặn từ phía server mục tiêu

- Phát hiện dấu hiệu của một cuộc tấn công DDOS (Detect and prevent agent):
 - Lưu lượng truy cập website tăng đột biến và xuất phát từ một hoặc 1 dải IP.
 - Khi sử dụng, mạng hoặc mạng của hệ thống bị chậm bất thường khi truy cập vào Website hay mở các tệp dữ liệu mạng Internet vẫn đang ổn định.
 - Không thể truy cập vào một trang của Website.
 - Website, cửa hàng trực tuyến hoặc dịch vụ ngoại tuyến hoàn toàn.
 - Nhận được nhiều thư rác trong tài khoản một cách bất thường.
- Tấn công từ chối dịch vụ (DDoS) đối với crawler
 - Phân loại tấn công từ chối dịch vụ phân tán:

Các loại tấn công DDoS có rất nhiều biến thể, nên việc phân loại cũng có rất nhiều cách khác nhau. Tuy nhiên, giới chuyên môn thường chia các kiểu tấn công DDoS thành 2 dạng chính, dựa vào mục đích của kẻ tấn công:

- > Tấn công DDoS làm cạn kiệt băng thông: Các cuộc tấn công DDoS băng thông cố gắng làm quá tải khả năng của tài nguyên. Máy chủ sẽ bị ngập với các yêu cầu, mạng sẽ bị tắc nghẽn với lưu lượng truy cập
- > Tấn công DDoS theo giao thức lạm dụng các giao thức để làm quá tải một tài nguyên cụ thể, thường là máy chủ, tường lửa (Firewall) hoặc bộ cân bằng tải (Load Balancer)



Ngoài việc phân loại như trên, có thể phân loại tấn công DDos dựa trên mô hình OSI 07 tầng. Xu hướng các cuộc tấn công DDos cho thấy thủ phạm thường biến đổi các cuộc tấn công theo mô hình OSI. Các cuộc tấn công được phân loại như sau:

- > Các cuộc tấn công IP nhằm vào bảng thông – tấn công vào tầng 3 (tầng mạng).
- > Các cuộc tấn công TCP trên máy chủ sockets – tấn công vào tầng 4 (tầng vận chuyển).
- > Các cuộc tấn công HTTP trên máy chủ web – tấn công vào tầng 7 (tầng ứng dụng).
- > Tấn công vào ứng dụng web, đánh vào tài nguyên CPU – tấn công trên tầng 7.

Ngày nay, hệ thống phòng thủ DDos liên tục được hoàn thiện và đa dạng, nhưng thường tập trung ở tầng thấp trong mô hình OSI. Do đó các cuộc tấn công vào tầng ứng dụng (Tầng 7) đang ngày càng phổ biến.

Khi phân tích một cuộc tấn công DDos nhằm vào Tầng 7, phải nghiên cứu các tầng khác. Do cuộc tấn công vào Tầng 7 luôn được ngụy trang và đi kèm với các cuộc tấn công nhằm vào tầng khác. Về bản chất, kẻ tấn công vào Tầng 7 sẽ tạo ra một giao diện cho người sử dụng như trình duyệt, các dịch vụ email, hình ảnh và những ứng dụng khác để gửi thông tin qua giao thức (SMTP, HTTP).

Một cuộc tấn công DDos vào Tầng 7 thường nhằm mục đích và mục tiêu cụ thể như: làm gián đoạn giao dịch, cản trở truy cập vào cơ sở dữ liệu. Kiểu tấn công này đòi hỏi nguồn lực ít hơn và đi kèm với các cuộc tấn công ở Tầng khác như tầng mạng. Một cuộc tấn công tầng ứng dụng sẽ được ngụy trang giống như những truy cập hợp pháp và nó có mục tiêu cụ thể là các ứng dụng. Cuộc tấn công có thể làm gián đoạn các chức năng cụ thể của dịch vụ như phản hồi thông tin, tìm kiếm ...

Chiến lược bảo mật hiệu quả:

- Phát hiện và ngăn chặn: Người sử dụng Internet sẽ tự đề phòng không để bị lợi dụng tấn công các hệ thống khác, phải liên tục tiến hành bảo vệ các thiết bị truy cập mạng của mình
- Ngăn chặn mã độc: Kiểm tra tính hợp lệ của nội dung thu thập được để ngăn chặn crawler tải mã độc từ các nguồn không đáng tin cậy
- Bảo vệ nội dung thu thập: Xác thực tính toàn vẹn của dữ liệu được thu thập. Loại bỏ hoặc làm sạch các nội dung nguy hiểm như mã độc trước khi lưu trữ hoặc xử lý.

Các công cụ hỗ trợ bảo mật cho hệ thống crawler

- Sử dụng tường lửa ứng dụng web (Web Application Firewall - WAF)
- Phòng tránh rò rỉ dữ liệu nhạy cảm

6. THUẬT TOÁN

6.1. Thuật toán Round-Robin

6.1.1. Giới thiệu

Round Robin là một thuật toán đơn giản nhưng hiệu quả trong việc phân phối tải giữa các node trong hệ thống phân tán. Trong hệ thống của bạn, Round Robin được sử dụng để lựa chọn node tiếp theo để xử lý child URLs khi crawling.

6.1.2. Cơ chế hoạt động

Mục tiêu: Đảm bảo phân phối đều các URL cần xử lý giữa các node, tránh tình trạng quá tải ở một node trong khi các node khác không hoạt động.

Cơ chế:

- Danh sách các node (URLS): Danh sách này chứa các URL của tất cả các node đang hoạt động, ví dụ:

$$URLS = ["http://localhost:5000", "http://localhost:5001", "http://localhost:5002"]$$

- Biến trạng thái `current_node_index`:
 - Chỉ số của node được chọn lần cuối cùng.
 - Bắt đầu từ 0 và tăng dần đến $\text{len}(URLS) - 1$, sau đó quay lại 0.
- Lựa chọn node tiếp theo (`get_next_url`):
 - Hàm `get_next_url` trong `generate_urls.py` triển khai Round Robin:

```
1 def get_next_url(list_of_urls,
2   current_node_index):
3   url = list_of_urls[current_node_index]
4   current_node_index = (current_node_index + 1) %
   len(list_of_urls)
5   return url, current_node_index
```
 - Cách hoạt động:
 - Lấy URL tại vị trí `current_node_index` trong danh sách.
 - Tăng chỉ số lên 1.
 - Nếu đã đến cuối danh sách, quay lại node đầu tiên (vòng xoay).

6.1.3. Vai trò của Round-Robin trong hệ thống

a) *Phân phối tải đều*

- Giảm thiểu nguy cơ một node phải xử lý quá nhiều URL trong khi các node khác nhàn rỗi.
- Đảm bảo tất cả các node được sử dụng hiệu quả.

b) *Đơn giản và hiệu quả:*

- Không yêu cầu thông tin phức tạp về trạng thái của các node.
- Triển khai dễ dàng với chi phí tính toán thấp

c) *Tối ưu crawling phân tán:*

- Child URLs từ một node được phân phối tự động đến các node khác, đảm bảo crawling đồng thời.

6.2. Thuật toán xử lý URL trong hệ thống

6.2.1. Vai trò

- Tránh lặp lại việc crawl: Đảm bảo mỗi URL chỉ được xử lý một lần.
- Hiệu quả và chính xác: Xác định nhanh chóng URL đã được xử lý hay chưa.
- Bảo toàn tài nguyên: Giảm tải cho các node và tối ưu hóa việc lưu trữ.

6.2.2. Khởi tạo cơ sở dữ liệu

Hệ thống sử dụng SQLite làm cơ sở dữ liệu đơn giản để quản lý trạng thái URL. Được thực hiện khi hệ thống bắt đầu, đảm bảo bảng seen_urls tồn tại.

```
1 def initialize_db():
2     conn = sqlite3.connect('url_seen.db')
3     c = conn.cursor()
4     c.execute('''
5     CREATE TABLE IF NOT EXISTS seen_urls (
6         url TEXT PRIMARY KEY
7     )
8     ''')
9     conn.commit()
10    conn.close()
```

6.2.3. Thiết kế chính

a) *Quản lý trạng thái URL*

- Thành phần chính:
 - Database (SQLite):
 - Bảng seen_urls chứa thông tin về các URL đã được crawl.
 - Mỗi URL được lưu trữ dưới dạng chuỗi văn bản (TEXT).
 - Các chức năng chính:
 - initialize_db: Tạo bảng trong SQLite nếu chưa tồn tại.
 - is_url_seen: Kiểm tra xem URL đã được crawl hay chưa.
 - mark_url_as_seen: Đánh dấu URL là đã được crawl.
- Cấu trúc bảng seen_urls:

```
1 CREATE TABLE IF NOT EXISTS seen_urls (
2     url TEXT PRIMARY KEY
3 )
```


b) Quy trình quản lý

Bước 1. Khi crawler nhận một URL:

Kiểm tra trong cơ sở dữ liệu:

- Nếu URL đã tồn tại, bỏ qua.
- Nếu không tồn tại, đánh dấu URL và bắt đầu crawl.

Bước 2. Sau khi xử lý URL:

Lưu trạng thái URL vào cơ sở dữ liệu

6.2.4. Triển khai thuật toán

a) Kiểm tra trạng thái URL

- Kiểm tra trạng thái URL: Hàm `is_url_seen` kiểm tra xem URL đã tồn tại trong bảng `seen_urls` hay chưa.

```
1 def is_url_seen(url):
2     conn = sqlite3.connect('url_seen.db')
3     c = conn.cursor()
4     c.execute('SELECT 1 FROM seen_urls WHERE url = ?',
5               (url,))
6     result = c.fetchone()
7     conn.close()
8     return result is not None
```

- Quy trình:

Bước 1. Mở kết nối đến cơ sở dữ liệu.

Bước 2. Thực hiện truy vấn tìm URL trong bảng `seen_urls`.

Bước 3. Trả về:

- True: Nếu URL đã tồn tại.
- False: Nếu URL chưa tồn tại.

b) Đánh dấu URL đã xử lý

- Đánh dấu URL đã xử lý: Sau khi xử lý xong một URL, nó được thêm vào bảng `seen_urls`:

```
1 def mark_url_as_seen(url):
2     conn = sqlite3.connect('url_seen.db')
3     c = conn.cursor()
4     c.execute('INSERT OR IGNORE INTO seen_urls (url)
5               VALUES (?)', (url,))
```

| | |
|---|----------------------------|
| 5 | <code>conn.commit()</code> |
| 6 | <code>conn.close()</code> |

- Quy trình:

Bước 1. Mở kết nối đến cơ sở dữ liệu.

Bước 2. Sử dụng INSERT OR IGNORE để đảm bảo không thêm trùng lặp:

- Nếu URL đã tồn tại, câu lệnh bị bỏ qua.

Bước 3. Đóng kết nối.

c) Hoạt động trong hệ thống

- Khi bắt đầu crawl:
 - Mỗi URL được kiểm tra bởi `is_url_seen` trước khi xử lý.
 - Nếu URL chưa được xử lý:
 - > Đánh dấu URL bằng `mark_url_as_seen`.
 - > Bắt đầu crawl nội dung.
- Child URLs:
 - Sau khi phân tích HTML của URL hiện tại, tất cả child URLs được kiểm tra trạng thái trước khi phân phối hoặc xử lý.

6.2.5. Xử lý URL con

a) Trích xuất child URL từ nội dung

- Sau khi tải thành công nội dung HTML từ một URL, hệ thống sử dụng BeautifulSoup để phân tích nội dung và trích xuất các liên kết (thẻ ``).
- Mỗi liên kết được chuyển đổi thành dạng URL đầy đủ (absolute URL) bằng cách sử dụng `urljoin`, đảm bảo chúng sẵn sàng cho quá trình xử lý tiếp theo

b) Kiểm tra child URL

Lọc URL hợp lệ: Mỗi URL được kiểm tra:

- Định dạng hợp lệ (ví dụ: đúng cấu trúc, không trống).
- Tránh các liên kết không cần thiết như `mailto:` hoặc `javascript:`.

c) Kiểm tra tính hợp lệ: robots.txt

- Mục đích:

- Tuân thủ quy định của website về các URL được phép hoặc không được phép crawl.
- Hoạt động:
 - Sử dụng `urllib.robotparser` để tải và đọc file `robots.txt` của website.
 - Kiểm tra xem user-agent có quyền truy cập URL không.
- Triển khai:

```

1  def can_fetch_url(url):
2      parsed_url = urlparse(url)
3      base_url                                     =
        f"{parsed_url.scheme}://{parsed_url.netloc}"
4      robots_url = f"{base_url}/robots.txt"
5
6      rp = robotparser.RobotFileParser()
7      rp.set_url(robots_url)
8      try:
9          rp.read()
10     except Exception as e:
11         return True # Mặc định cho phép nếu không thể
        đọc file robots.txt
12
13     user_agent = "MyFancySpider"
14     return rp.can_fetch(user_agent, url)

```

6.2.6. Phân phối child URL đến các node khác

- Cơ chế phân phối:
 - Các child URL được phân phối đến các node trong hệ thống dựa trên thuật toán Round Robin hoặc các phương pháp cân bằng tải khác `generate_urlsnode`.
- Hạn chế số lượng: Một giới hạn tối đa (ví dụ: 8 URL) được áp dụng để tránh việc xử lý quá tải tại một node.

6.2.7. Đánh dấu trạng thái URL

- Tránh trùng lặp: Trước khi phân phối, child URL được kiểm tra trong cơ sở dữ liệu để xem liệu chúng đã được crawl trước đó chưa. Nếu đã xử lý, URL đó sẽ bị bỏ qua `database_manager`.

- Đánh dấu là đã xử lý: Child URL mới được thêm vào cơ sở dữ liệu với trạng thái "đã crawl" sau khi chúng được phân phối thành công database_manager.

6.2.8. Xử lý child URL ở các mức độ sâu hơn (Levels)

- Nếu hệ thống yêu cầu crawl theo mức độ (levels):
 - Child URL được xử lý ở mức độ hiện tại sẽ tiếp tục trích xuất thêm các child URL của riêng chúng.
 - Quá trình này lặp lại cho đến khi đạt đến mức giới hạn hoặc không còn child URL nào mới.
- Lưu trữ kết quả:
 - Nội dung của mỗi child URL được tải và lưu trữ, cùng với thông tin về các URL liên kết đến nó, tạo thành một đồ thị liên kết (adjacency list) node
- Tạo đồ thị quan hệ giữa các URL
 - Sau khi crawl child URL, hệ thống có thể tổng hợp dữ liệu và biểu diễn mối quan hệ giữa các URL gốc và child URL dưới dạng đồ thị.
 - Điều này hỗ trợ việc phân tích cấu trúc liên kết của website make_graph_function.

6.3. Thuật toán Retry và xử lý lỗi trong hệ thống

6.3.1. Giới thiệu

Thuật toán retry và xử lý lỗi được thiết kế để đảm bảo tính ổn định của hệ thống crawler. Khi một node hoặc một URL gặp lỗi trong quá trình xử lý, thuật toán sẽ cố gắng khắc phục tự động, giảm thiểu sự gián đoạn.

6.3.2. Vai trò

- Đảm bảo hệ thống hoạt động liên tục:
Nếu một node không phản hồi hoặc một URL không thể được crawl, hệ thống cố gắng khắc phục thay vì dừng hoạt động.
- Giảm thiểu lỗi tạm thời:
Nhiều lỗi trong hệ thống phân tán chỉ là tạm thời (ví dụ: lỗi kết nối, quá tải). Thuật toán retry giúp vượt qua những lỗi này.
- Tăng tính ổn định:
Hệ thống không dễ bị gián đoạn bởi lỗi của một node hoặc một tác vụ.

6.3.3. Cách hoạt động

a) *Retry khi một node không phản hồi*

Khi Health Monitor phát hiện một node không phản hồi, hệ thống sẽ cố gắng khởi động lại node đó một số lần.

Triển khai trong `health_monitor.py`:

```
1  async def retry_node(node_index, total_nodes):
2      port = 5000 + node_index
3      try:
4          print(f"Restarting node on port {port}...")
5          subprocess.Popen(['python', 'node.py',
6                             str(port), str(total_nodes)], shell=True)
7          await asyncio.sleep(5) # Chờ node khởi động
8          print(f"Node on port {port} restarted
9              successfully.")
9      except Exception as e:
10         print(f"Failed to restart node on port
11             {port}: {e}")
```

Cách hoạt động:

Bước 1. Lấy chỉ số node không phản hồi (`node_index`).

Bước 2. Tính tổng tương ứng của node ($5000 + \text{node_index}$).

Bước 3. Thử khởi động lại node bằng cách chạy lại script `node.py`.

Bước 4. Đợi một khoảng thời gian để node khởi động.

Xử lý lỗi: Nếu việc khởi động lại thất bại, lỗi được ghi lại và node có thể được đánh dấu là không khả dụng.

b) *Retry khi crawl URL thất bại*

Khi một URL không thể được crawl (do lỗi kết nối, hết thời gian chờ, hoặc bị từ chối bởi server), hệ thống sẽ cố gắng retry.

Triển khai trong `node.py`:

```
1  async def send_crawl_request(worker_url, child_url,
    levels):
2      params = {"website": child_url, "levels": levels}
3      async with aiohttp.ClientSession() as session:
4          try:
5              async with
session.post(f"{worker_url}crawl", json=params) as
response:
6                  if response.status == 200:
7                      print(f"Successfully sent crawl
request to {worker_url}")
8                  else:
9                      print(f"Failed to send crawl
request to {worker_url} with status
{response.status}")
10                 except Exception as e:
11                     print(f"Error sending request to
{worker_url}: {e}")
```

Cách hoạt động:

Bước 1. Gửi yêu cầu HTTP POST đến node xử lý URL (`worker_url/crawl`).

Bước 2. Nếu không nhận được phản hồi thành công (`status != 200`), báo lỗi và có thể thử lại.

Bước 3. Xử lý ngoại lệ (timeout, lỗi kết nối) và ghi log.

c) **Logic Retry trong các trường hợp cụ thể**

- Retry URL với giới hạn số lần:

Khi crawl một URL, hệ thống có thể retry một số lần giới hạn trước khi bỏ qua URL đó.

Ví dụ triển khai trong util.py:

```
1 def node_retry(tries, node, num_nodes):
2     if tries == 0:
3         print(f"Node {node} is marked as unreachable
4         after {tries} retries.")
5         return
6     try:
7         port = 5000 + node
8         subprocess.Popen(['python', 'node.py',
9         str(port), str(num_nodes)], shell=True)
10    except Exception as e:
11        print(f"Retry failed for node {node}: {e}")
12        node_retry(tries - 1, node, num_nodes)
```

Cách hoạt động:

- Thử một số lần nhất định (biến tries).
 - Giảm số lần retry mỗi khi thất bại.
 - Nếu vượt quá giới hạn, đánh dấu node là không khả
- Retry định kỳ:
Trong Health Monitor, trạng thái của các node được kiểm tra định kỳ:

```
1 async def periodic_healthcheck(node_count):
2     urls = make_urls(node_count)
3     while True:
4         async with aiohttp.ClientSession() as session:
5             tasks = [check_node_health(session, url) for
6             url in urls]
7             results = await asyncio.gather(*tasks)
```

```
8         unreachable_nodes = [urls.index(result["url"])
    for result in results if result["status"] ==
    "unreachable"]
9         for node_index in unreachable_nodes:
10             await retry_node(node_index, node_count)
11
12         await asyncio.sleep(10) # Kiểm tra lại sau 10
    giây
```


6.4. Thuật toán phân phối công việc trong hệ thống

6.4.1. Giới thiệu

Hệ thống crawler của bạn sử dụng thuật toán phân phối công việc để đảm bảo các child URLs được phân phối đồng đều và hiệu quả giữa các node. Đây là một thành phần quan trọng, giúp hệ thống tận dụng sức mạnh của các node để đạt hiệu suất cao và giảm thiểu tình trạng quá tải tại một node duy nhất.

6.4.2. Mục tiêu

- Cân bằng tải:
Đảm bảo tất cả các node trong hệ thống được sử dụng đồng đều, tránh tình trạng quá tải.
- Tận dụng khả năng song song:
Tăng tốc độ crawling bằng cách phân phối các child URLs đến nhiều node.
- Đảm bảo tính hiệu quả:
Giảm thiểu độ trễ và tối ưu hóa tài nguyên của hệ thống.

6.4.3. Cách hoạt động

a) Thuật toán sử dụng: Round-Robin

- Mục đích: Phân phối lần lượt các child URLs đến các node theo thứ tự.
- Cơ chế:
 - Dựa trên danh sách URLs chứa địa chỉ các node trong hệ thống.
 - Sử dụng chỉ số `current_node_index` để xác định node tiếp theo trong danh sách.
 - Sau mỗi lần phân phối, chỉ số được tăng lên, và quay lại đầu danh sách khi đến cuối.

b) Phân phối child URLs

- Sau khi crawl một URL, hệ thống tìm các child URLs từ nội dung HTML và phân phối chúng đến các node khác để tiếp tục crawl.
- Cách hoạt động:
 - Với mỗi URL con (`child_url`), chọn một node tiếp theo từ danh sách URLs bằng `get_next_url`.
 - Tạo một tác vụ (task) để gửi yêu cầu crawl đến node được chọn.
 - Dùng `asyncio.gather` để thực hiện đồng thời tất cả các tác vụ.

c) Gửi yêu cầu đến node

Các child URLs được gửi đến endpoint /crawl của node thông qua HTTP POST.

Triển khai trong send_crawl_request:

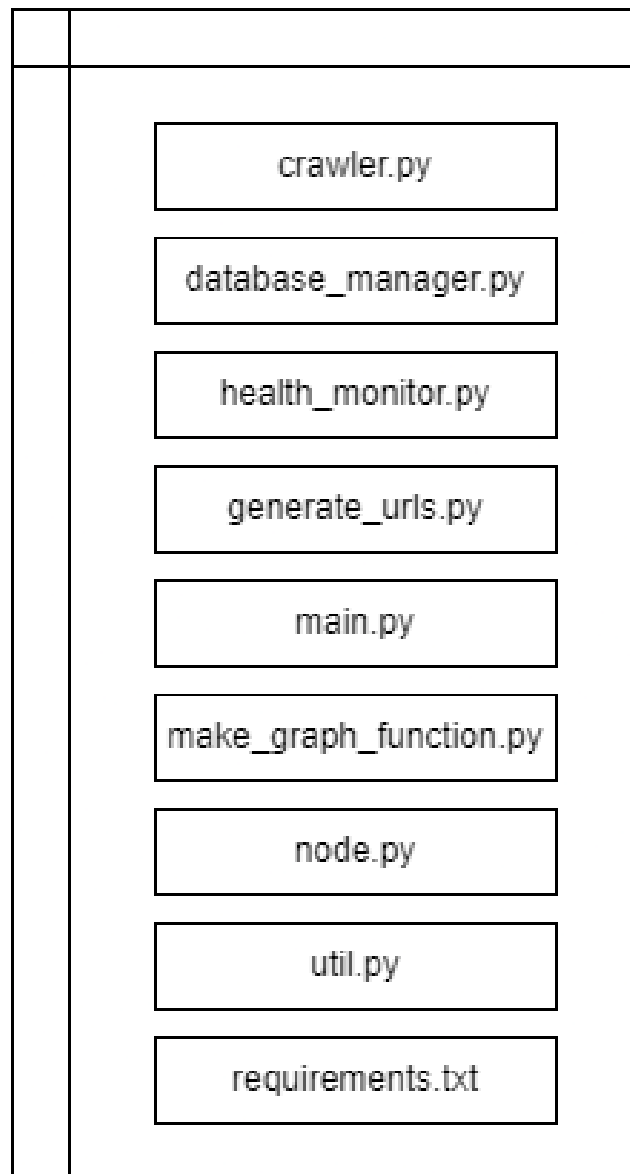
```
1  async def send_crawl_request(worker_url, child_url,
    levels):
2      params = {"website": child_url, "levels": levels}
3      async with aiohttp.ClientSession() as session:
4          try:
5              async with
session.post(f"{worker_url}crawl", json=params) as
response:
6                  if response.status == 200:
7                      print(f"Successfully sent crawl
request to {worker_url}")
8                  else:
9                      print(f"Failed to send crawl
request to {worker_url} with status {response.status}")
10             except Exception as e:
11                 print(f"Error sending request to
{worker_url}: {e}")
```

Cách hoạt động:

- Bước 1.** Tạo payload JSON chứa URL cần crawl và độ sâu (levels).
- Bước 2.** Gửi yêu cầu HTTP POST đến node được chọn qua endpoint /crawl.
- Bước 3.** Ghi nhận trạng thái phản hồi:
 - Thành công (status == 200).
 - Thất bại (ghi log và có thể retry).

7. XÂY DỰNG HỆ THỐNG

7.1. Kiến trúc hệ thống



7.2. Mã nguồn

7.2.1. crawler.py

```
1  import aiohttp
2  from bs4 import BeautifulSoup
3  import validators
4  import urllib.robotparser as robotparser
5  from urllib.parse import urlparse
6  import hashlib
7  import os
8  from urllib.parse import urljoin
9  from database_manager import is_url_seen, mark_url_as_seen
10 import re
11
12 def sanitize_filename(name):
13     """Loại bỏ ký tự không hợp lệ khỏi tên file."""
14     return re.sub(r'[\<>:"/\|?*]', '_', name)
15
16 def hard_disk_store(text, url):
17     """Lưu nội dung HTML vào file trong thư mục storage với tên dựa trên
18 thẻ <title>."""
19     storage_dir = "storage"
20     os.makedirs(storage_dir, exist_ok=True)
21
22     # Phân tích HTML để lấy thẻ <title>
23     soup = BeautifulSoup(text, "html.parser")
24     title = soup.title.string if soup.title else "untitled"
25     sanitized_title = sanitize_filename(title)[:100] # Giới hạn độ dài
26     tên
27
28     # Thêm hash của URL để tránh trùng lặp
29     filename =
30     f"{sanitized_title}_{hashlib.md5(url.encode()).hexdigest()}.html"
31     filepath = os.path.join(storage_dir, filename)
32
33     if os.path.exists(filepath):
```

```

34         print(f"File {filename} already exists, skipping...")
35         return
36
37     print("HTML stored in:", filepath)
38
39     with open(filepath, "w+", encoding="utf-8") as file:
40         file.write(text)
41
42 def validate_url(url):
43     try:
44         if validators.url(url):
45             return True
46         return False
47     except:
48         return False
49
50 def can_fetch_url(url):
51     """
52     Kiểm tra URL có được phép crawl hay không dựa vào file robots.txt.
53     """
54     parsed_url = urlparse(url)
55     base_url = f"{parsed_url.scheme}://{parsed_url.netloc}"
56     robots_url = f"{base_url}/robots.txt"
57
58     rp = robotparser.RobotFileParser()
59     rp.set_url(robots_url)
60     try:
61         rp.read()
62     except Exception as e:
63         print(f"Không thể đọc {robots_url}. Mặc định cho phép: {e}")
64         return True # Mặc định cho phép nếu không thể đọc file
65 robots.txt
66     user_agent = "trahabo" # Tên user-agent của crawler (đặt tên riêng
67 của bạn)
68     return rp.can_fetch(user_agent, url)

```

```

69
70  async def crawl(url):
71      # Kiểm tra nếu URL đã được crawl
72      if is_url_seen(url):
73          print(f"URL {url} already seen, skipping...")
74          return []
75      # Đánh dấu URL đã được xử lý
76      mark_url_as_seen(url)
77
78      if not can_fetch_url(url):
79          print(f"URL {url} is disallowed by robots.txt")
80          return []
81      try:
82          async with aiohttp.ClientSession() as session:
83              async with session.get(url, timeout=5) as response:
84                  html = await response.text()
85                  hard_disk_store(html, url) # Lưu dữ liệu vào ổ đĩa
86                  print(f"Crawled URL: {url}")
87                  # Phân tích HTML để lấy child URLs
88                  # Lấy các URL từ tag <a>
89                  soup = BeautifulSoup(html, "html.parser")
90                  a_urls = [
91                      urljoin(url, link.get('href'))
92                      for link in soup.find_all('a') if link.get('href')
93                  ]
94                  # Lấy các URL từ tag <link>
95                  link_urls = [
96                      urljoin(url, link.get('href'))
97                      for link in soup.find_all('link') if link.get('href')
98                  ]
99                  child_urls = a_urls + link_urls
100                 return child_urls
101      except Exception as e:
102          print(f"Error crawling {url}: {e}")
103          return []

```

7.2.2. database_manager.py

```
1  # database_manager.py
2  import sqlite3
3
4  # Tạo kết nối và bảng SQLite
5  def initialize_db():
6      conn = sqlite3.connect('url_seen.db')
7      c = conn.cursor()
8      c.execute('''
9          CREATE TABLE IF NOT EXISTS seen_urls (
10             url TEXT PRIMARY KEY
11          )
12      ''')
13      conn.commit()
14      conn.close()
15
16  # Kiểm tra URL đã được crawl
17  def is_url_seen(url):
18      conn = sqlite3.connect('url_seen.db')
19      c = conn.cursor()
20      c.execute('SELECT 1 FROM seen_urls WHERE url = ?', (url,))
21      result = c.fetchone()
22      conn.close()
23      return result is not None
24
25  # Đánh dấu URL là đã crawl
26  def mark_url_as_seen(url):
27      conn = sqlite3.connect('url_seen.db')
28      c = conn.cursor()
29      c.execute('INSERT OR IGNORE INTO seen_urls (url) VALUES (?)', (url,))
30      conn.commit()
31      conn.close()
32
33  def clear_seen_urls():
34      """
```

```
35     Xóa toàn bộ dữ liệu trong bảng seen_urls.  
36     ""  
37     conn = sqlite3.connect('url_seen.db')  
38     c = conn.cursor()  
39     c.execute('DELETE FROM seen_urls')  
40     conn.commit()  
41     conn.close()  
42     print("Database cleared: all URLs removed.")
```


7.2.3. generate_urls.py

```
1  def make_urls(number_of_ports):
2      URLs = []
3      for i in range(number_of_ports):
4          url = "http://localhost:" + str(5000+i) + "/"
5          URLs.append(url)
6      return URLs
7      URLs = []
8
9  # Round Robin implementation
10 current_node_index = 0 # Biến toàn cục để lưu trạng thái vòng xoay
11
12 def get_next_url(list_of_urls, current_node_index):
13     url = list_of_urls[current_node_index]
14     current_node_index = (current_node_index + 1) % len(list_of_urls)
15     return url, current_node_index
16
17
18 def check_if_exists_and_print_html(name):
19     try:
20         f = open(name, 'r')
21         Lines = f.readlines()
22         for line in Lines:
23             print(line)
24         f.close()
25     except IOError:
26         print("File not accessible")
27     return
```

7.2.4. health_monitor.py

```
1  import asyncio
2  import aiohttp
3  import subprocess
4  from datetime import datetime
5  from generate_urls import make_urls
6
7  async def check_node_health(session, url):
8      """
9      Gửi yêu cầu kiểm tra trạng thái của một node.
10     """
11     try:
12         async with session.get(url + "status", timeout=5) as response:
13             if response.status == 200:
14                 data = await response.json()
15                 return {"url": url, "status": "healthy", "details": data}
16     except Exception as e:
17         print(f"[{datetime.now()}] Failed to reach {url}: {e}")
18         return {"url": url, "status": "unreachable", "details": None}
19
20 async def retry_node(node_index, total_nodes):
21     """
22     Thử khởi động lại node không phản hồi.
23     """
24     port = 5000 + node_index
25     try:
26         print(f"[{datetime.now()}] Restarting node on port {port}...")
27         subprocess.Popen(['python', 'node.py', str(port),
28 str(total_nodes)], shell=True)
29         await asyncio.sleep(5) # Chờ node khởi động
30         print(f"[{datetime.now()}] Node on port {port} restarted
31 successfully.")
32     except Exception as e:
33         print(f"[{datetime.now()}] Failed to restart node on port {port}:
34 {e}")
```

```

35  async def periodic_healthcheck(node_count):
36      """
37      Kiểm tra định kỳ trạng thái của các node.
38      """
39      urls = make_urls(node_count)
40      while True:
41          print(f"[{datetime.now()}] Checking node statuses...")
42          async with aiohttp.ClientSession() as session:
43              tasks = [check_node_health(session, url) for url in urls]
44              results = await asyncio.gather(*tasks)
45              unreachable_nodes = []
46              for result in results:
47                  if result["status"] == "healthy":
48                      print(f"[{datetime.now()}] Node {result['url']} is
49 healthy. Details: {result['details']}")
50                  else:
51                      print(f"[{datetime.now()}] Node {result['url']} is
52 unreachable.")
53                      unreachable_nodes.append(urls.index(result["url"]))
54                      #Xử lý các node không hoạt động
55                      for node_index in unreachable_nodes:
56                          await retry_node(node_index, node_count)
57
58                      print(f"[{datetime.now()}] Health check completed. Waiting for
59 next round...")
60                      await asyncio.sleep(10) # Lặp lại sau 10 giây
61
62  if __name__ == "__main__":
63      import sys
64      try:
65          node_count = int(sys.argv[1])
66          asyncio.run(periodic_healthcheck(node_count))
67      except Exception as e:
68          print("Usage: python health_monitor.py <node_count>")
69          print(f"Error: {e}")

```

7.2.5. main.py

```
1  import asyncio
2  import aiohttp
3  import json
4  from util import start_all_nodes, start_health_monitor, MAX_PORTS
5  from generate_urls import make_urls, get_next_url,
6  check_if_exists_and_print_html
7  from database_manager import initialize_db, clear_seen_urls
8
9  current_node_index = 0
10 URLs = []
11
12 async def main():
13     global current_node_index, URLs
14     initialize_db()
15     clear_seen_urls()
16
17     number_of_ports = int(input("Enter number of nodes: "))
18     if number_of_ports > MAX_PORTS:
19         print("Only", MAX_PORTS, "nodes created")
20         URLs.extend(make_urls(MAX_PORTS))
21     else:
22         URLs.extend(make_urls(number_of_ports))
23
24     if len(URLS) == 0:
25         raise Exception("Configuration error")
26
27     print(URLS)
28
29     try:
30         start_all_nodes(number_of_ports)
31         start_health_monitor(number_of_ports)
32     except Exception as e:
33         print("Initial node starting failed")
34         print(e)
```

```

35         raise e
36     while True:
37         command = input("=> Enter Command: ")
38         if command == "end":
39             break
40         elif command == "crawl":
41             website = input("Enter Website: ")
42             levels = int(input("Enter levels: "))
43             random_url, current_node_index = get_next_url(URLS,
44 current_node_index)
45             url = random_url + "crawl"
46             params = {"website": website, "levels": levels}
47
48             await crawl_website(url, params)
49             continue
50
51         elif command == "show_html":
52             name = input('EnterFile html:')
53             print(name)
54             check_if_exists_and_print_html(name)
55             continue
56
57         elif command == "exit":
58             exit(0)
59         else:
60             print("Invalid Command. Try Again.")
61             continue
62
63     async def crawl_website(url, params):
64         async with aiohttp.ClientSession() as session:
65             try:
66                 async with session.post(url, json=params) as response:
67                     if response.status == 200:
68                         print("Crawling successful")
69                         print(await response.text())

```

```
70         else:
71             print(f"Server error: {response.status}, {await
72 response.text()}")
73         except Exception as e:
74             print(f"Error sending request: {e}")
75
76     # Chạy chương trình bất đồng bộ
77     if __name__ == "__main__":
78         asyncio.run(main())
```

7.2.6. node.py

```
1  from fastapi import FastAPI, HTTPException
2  import asyncio
3  import aiohttp
4  from pydantic import BaseModel
5  from crawler import crawl, can_fetch_url # Hàm crawl bất đồng bộ
6  from generate_urls import make_urls, get_next_url
7  import psutil
8  from pydantic import BaseModel, field_validator, ValidationError
9  from fastapi.exceptions import RequestValidationError
10 from fastapi.responses import JSONResponse
11 import logging
12
13 app = FastAPI()
14
15 # Biến toàn cục
16 current_node_index = 0
17 URLs = []
18 limiter = 8 # Giới hạn số lượng child URL
19 child_adjacency = {}
20 current_tasks = 0
21
22 class CrawlRequest(BaseModel):
23     website: str
24     levels: int
25
26     @field_validator("levels")
27     def validate_levels(cls, value):
28         if value <= 0:
29             raise ValueError("Levels must be greater than 0")
30         return value
31
32 @app.get("/")
33 async def root():
34     """
```

```

35     Endpoint mặc định cho root (/) của server.
36     """
37     return {"message": "Welcome to the distributed crawler node!",
38 "status": "running"}
39
40
41 @app.get("/status")
42 async def status():
43     """
44     Route kiểm tra trạng thái node.
45     """
46     return {
47         "status": "healthy",
48         "cpu_usage": psutil.cpu_percent(interval=1),
49         "memory_usage": psutil.virtual_memory().percent,
50         "current_tasks": len(child_adjacency),
51     }
52
53 @app.exception_handler(RequestValidationError)
54 async def validation_exception_handler(request, exc):
55     """
56     Ghi đè xử lý lỗi validation, bỏ qua lỗi hoặc trả phản hồi tùy chỉnh.
57     """
58     # Không log lỗi, chỉ trả về phản hồi mặc định
59     logging.info("Validation error occurred but ignored.")
60     return JSONResponse(
61         status_code=200,
62         content={"message": "Validation error ignored."},
63     )
64
65 @app.post("/crawl")
66 async def crawl_endpoint(request: CrawlRequest):
67     print(f"Received payload: {request.model_dump()}")
68     """
69     Route xử lý yêu cầu crawl.

```



```

70         """
71         global current_node_index, current_tasks
72         try:
73             current_tasks += 1
74             print(f"Received request: {request.model_dump()}") # Log payload
75
76             website = request.website
77             levels = request.levels
78
79             if not can_fetch_url(website):
80                 return {"status": "disallowed", "message": f"URL {website} is
81 disallowed by robots.txt"}
82
83             # Crawl URL
84             print(f"Crawling website: {website} at levels: {levels}")
85             child_urls = await crawl(website)
86             print(f"Found child URLs: {child_urls}")
87
88             # Giới hạn số lượng child URLs
89             if len(child_urls) > limiter:
90                 child_urls = child_urls[:limiter]
91             # Phân phối child URLs cho các node khác
92             tasks = []
93             for child_url in child_urls:
94                 worker_url, current_node_index = get_next_url(URLS,
95 current_node_index)
96                 tasks.append(send_crawl_request(worker_url, child_url, levels
97 - 1))
98             await asyncio.gather(*tasks)
99
100             # Cập nhật adjacency list
101             if website not in child_adjacency:
102                 child_adjacency[website] = child_urls
103             else:
104                 child_adjacency[website] += child_urls

```

```

105         return {"status": "Crawling completed", "found_urls": child_urls}
106     except ValidationError as e:
107         raise HTTPException(status_code=400, detail=str(e))
108     except Exception as e:
109         print(f"Unexpected error: {e}")
110         raise HTTPException(status_code=500, detail="Internal server
111 error")
112     finally:
113         current_tasks -= 1
114         print(f"Current tasks: {current_tasks}")
115
116 @app.get("/make_graph")
117 async def make_graph(website: str, depth: int):
118     """
119     Route xử lý tạo đồ thị liên kết.
120     """
121     print(f"Processing graph for website: {website} at depth: {depth}")
122     result = {}
123     if website in child_adjacency:
124         result[website] = child_adjacency[website][:depth]
125     else:
126         result[website] = []
127
128     return result
129
130 @app.get("/check")
131 async def check():
132     """
133     Route kiểm tra trạng thái của các node khác.
134     """
135     answers = []
136     async with aiohttp.ClientSession() as session:
137         tasks = [check_node_status(session, url) for url in URLs]
138         results = await asyncio.gather(*tasks)
139         answers.extend(results)

```

```

140     return {"status": "Nodes checked", "results": answers}
141
142 @app.get("/health")
143 async def health():
144     """
145     Route kiểm tra trạng thái node hiện tại.
146     """
147     return {"host": "localhost", "status": "healthy"}
148
149 async def send_crawl_request(worker_url, child_url, levels):
150     """
151     Gửi yêu cầu crawl không đồng bộ tới các node khác.
152     """
153     params = {"website": child_url, "levels": levels}
154     headers = {"Content-Type": "application/json"}
155     print(f"Payload being sent: {params}")
156     async with aiohttp.ClientSession(headers=headers) as session:
157         try:
158             async with session.post(f"{worker_url}crawl", json=params) as
159 response:
160                 if response.status == 200:
161                     print(f"Successfully sent crawl request to
162 {worker_url}")
163                 else:
164                     print(f"Failed to send crawl request to {worker_url}
165 with status {response.status}")
166                 except Exception as e:
167                     print(f"Error sending request to {worker_url}: {e}")
168
169 async def check_node_status(session, url):
170     """
171     Kiểm tra trạng thái của một node khác.
172     """
173     try:
174         async with session.get(url + "status", timeout=3) as response:

```

```
175         return await response.json()
176     except Exception as e:
177         return {"url": url, "status": "unreachable", "error": str(e)}
178
179 if __name__ == "__main__":
180     import uvicorn
181     import sys
182     from generate_urls import make_urls
183     try:
184         port_number = int(sys.argv[1])
185         number_of_nodes = int(sys.argv[2])
186     except Exception:
187         print("Usage: python node.py <port_number> <number_of_nodes>")
188         exit(1)
189
190     # Tạo danh sách URL của các node
191     URLs = make_urls(number_of_nodes)
192     print(f"Starting server at port {port_number}")
193     uvicorn.run(app, host="0.0.0.0", port=port_number)
```

7.2.7. util.py

```
1  import subprocess
2  MAX_PORTS = 6
3
4  def start_all_nodes(n):
5      for i in range(n):
6          port = 5000 + i
7          # Use 'start' in cmd to open a new terminal
8          subprocess.Popen(['start', 'cmd', '/k', 'python', 'node.py',
9 str(port), str(n)], shell=True)
10 def start_a_nodes(n, num_nodes):
11     port = 5000 + n
12     # Run the node script directly
13     subprocess.Popen(['start', 'cmd', '/k', 'python', 'node.py',
14 str(port), str(n)], shell=True)
15 def start_health_monitor(num_nodes):
16     subprocess.Popen(['start', 'cmd', '/k', 'python', 'health_monitor.py',
17 str(num_nodes)], shell=True)
18 def node_retry(tries, node, num_nodes):
19     if tries == 0:
20         print(f"Node {node} is marked as unreachable after {tries}
21 retries.")
22         return
23     try:
24         port = 5000 + node
25         subprocess.Popen(['start', 'cmd', '/k', 'python', 'node.py',
26 str(port), str(num_nodes)], shell=True)
27     except Exception as e:
28         print(f"Retry failed for node {node}: {e}")
29         node_retry(tries - 1, node, num_nodes)
```

7.2.8. make_graph_function.py

```
1  import dash
2  import dash as dcc
3  import dash as html
4  import dash_cytoscape as cyto
5  from dash.dependencies import Input, Output
6  import plotly.express as px
7
8  import plotly.graph_objects as go # or plotly.express as px
9  fig = go.Figure() # or any Plotly Express function e.g. px.bar(...)
10
11 def make_graph_from_adjacency_list(data):
12     # data is a huge json dict
13     # you have to filter out redundancies, and make the final graph out
14     of it
15     edges = []
16     # convert data into graph elements
17     for key in data.keys():
18         for node in data[key]:
19             edges.append([key, node])
20     for edge in edges:
21         print(edge)
```

7.2.9. requirements.txt

```
1 aiohappyeyeballs==2.4.4
2 aiohttp==3.11.11
3 aiosignal==1.3.2
4 amqp==5.3.1
5 annotated-types==0.7.0
6 anyio==4.7.0
7 asttokens==2.4.1
8 async-timeout==5.0.1
9 attrs==24.3.0
10 beautifulsoup4==4.12.3
11 billiard==4.2.1
12 blinker==1.9.0
13 bs4==0.0.2
14 celery==5.4.0
15 certifi==2024.8.30
16 cffi==1.17.1
17 charset-normalizer==3.4.0
18 click==8.1.7
19 click-didyoumean==0.3.1
20 click-plugins==1.1.1
21 click-repl==0.3.0
22 colorama==0.4.6
23 comm==0.2.2
24 contourpy==1.3.1
25 cycler==0.12.1
26 dash==2.18.2
27 dash-core-components==2.0.0
28 dash-cytoscape==1.0.2
29 dash-html-components==2.0.0
30 dash-table==5.0.0
31 debugpy==1.8.7
32 decorator==5.1.1
33 dnspython==2.7.0
34 exceptiongroup==1.2.2
```

```
35  executing==2.1.0
36  fastapi==0.115.6
37  findlibs==0.0.5
38  findspark==2.0.1
39  Flask==3.0.3
40  fonttools==4.55.0
41  frozenlist==1.5.0
42  h11==0.14.0
43  idna==3.10
44  importlib_metadata==8.5.0
45  ipykernel==6.29.5
46  ipython==8.28.0
47  itsdangerous==2.2.0
48  jedi==0.19.1
49  Jinja2==3.1.4
50  jupyter_client==8.6.3
51  jupyter_core==5.7.2
52  kiwisolver==1.4.7
53  kombu==5.4.2
54  MarkupSafe==3.0.2
55  matplotlib==3.9.2
56  matplotlib-inline==0.1.7
57  multidict==6.1.0
58  nest-asyncio==1.6.0
59  networkx==3.4.2
60  numpy==2.1.2
61  packaging==24.1
62  pandas==2.2.3
63  parso==0.8.4
64  pillow==11.0.0
65  platformdirs==4.3.6
66  plotly==5.24.1
67  plumbum==1.9.0
68  prompt_toolkit==3.0.48
69  propcache==0.2.1
```



```
70  psutil==6.0.0
71  pure_eval==0.2.3
72  py4j==0.10.9.7
73  pycparser==2.22
74  pydantic==2.10.4
75  pydantic_core==2.27.2
76  Pygments==2.18.0
77  pymongo==4.10.1
78  pyodc==1.4.1
79  pyparsing==3.2.0
80  pyspark==3.5.3
81  pystyle==2.9
82  python-dateutil==2.9.0.post0
83  pytz==2024.2
84  pywin32==307
85  pyzmq==26.2.0
86  redis==5.2.0
87  requests==2.32.3
88  retrying==1.3.4
89  rpyc==6.0.1
90  schedule==1.2.2
91  six==1.16.0
92  sniffio==1.3.1
93  soupsieve==2.6
94  stack-data==0.6.3
95  starlette==0.41.3
96  tenacity==9.0.0
97  termcolor==2.5.0
98  tornado==6.4.1
99  traitlets==5.14.3
100 typing_extensions==4.12.2
101 tzdata==2024.2
102 urllib3==2.2.3
103 uvicorn==0.34.0
104 validators==0.34.0
```

```
105  vine==5.1.0
106  wcwidth==0.2.13
107  Werkzeug==3.0.6
108  yarl==1.18.3
109  zipp==3.21.0
```

7.3. Chi tiết các tệp hệ thống

7.3.1. crawler.py

- Chức năng cốt lõi: Máy chủ crawling.
- Nhiệm vụ:
 - Phân tích HTML.
 - Trích xuất URL.
 - Kiểm tra tệp robots.txt.
 - Lưu trữ cục bộ.
- Hàm chính:
 - `crawl()`: Thực hiện việc crawl dữ liệu từ URL.
 - `validate_url()`: Xác thực URL hợp lệ.
 - `hard_disk_store()`: Lưu dữ liệu HTML vào ổ đĩa cục bộ.

7.3.2. database_manager.py

- Quản lý cơ sở dữ liệu SQLite: Theo dõi các URL đã được crawl.
- Tính năng:
 - Một bảng đơn giản: `seen_urls`.
- Hàm chính:
 - `initialize_db()`: Khởi tạo cơ sở dữ liệu.
 - `is_url_seen()`: Kiểm tra URL đã được crawl chưa.
 - `mark_url_as_seen()`: Đánh dấu URL đã crawl.

7.3.3. generate_urls.py

- Tiện ích tạo URL:
 - Sinh URL cho các node (ví dụ: `localhost:5000+`).
 - Áp dụng cơ chế cân bằng tải round-robin.
- Hàm chính:
 - `make_urls()`: Sinh danh sách URL.
 - `get_next_url()`: Lấy URL kế tiếp theo thứ tự.

7.3.4. health_monitor.py

- Hệ thống giám sát:
 - Hoạt động không đồng bộ (async).
 - Kiểm tra trạng thái các node mỗi 10 giây.
 - Tự động khởi động lại node nếu xảy ra lỗi.

- Hàm chính:
 - `periodic_healthcheck()`: Thực hiện kiểm tra sức khỏe định kỳ.

7.3.5. **main.py**

- Điểm khởi động của hệ thống:
 - Cung cấp giao diện dòng lệnh (CLI).
 - Khởi tạo các node và hệ thống giám sát sức khỏe.
- Xử lý các lệnh người dùng:
 - `crawl`: Bắt đầu crawl.
 - `show_html`: Hiển thị nội dung HTML đã crawl.

7.3.6. **node.py**

Máy chủ FastAPI:

- REST endpoints chính:
 - `/crawl`: Bắt đầu crawl.
 - `/status`: Kiểm tra trạng thái.
 - `/health`: Theo dõi sức khỏe node.
- Quản lý tác vụ crawler.
- Hạn chế số lượng child URLs và thực hiện xác thực.

7.3.7. **util.py**

- Tiện ích quản lý node:
 - Khởi động và khôi phục các tiến trình của node.
 - Định nghĩa hằng số `MAX_PORTS = 6` (giới hạn số node).
- Hàm chính:
 - `start_all_nodes()`: Khởi động tất cả node.
 - `start_health_monitor()`: Khởi chạy hệ thống giám sát.

7.3.8. **make_graph_function.py**

- Tiện ích trực quan hóa (chưa hoàn thiện):
 - Sử dụng Dash/Plotly để tạo đồ thị.
- Hàm chính:
 - `make_graph_from_adjacency_list()`: Tạo đồ thị từ danh sách liên kết.

7.4. Thành phần chính

7.4.1. Quản lý Node

- `node.py`: Máy chủ FastAPI xử lý crawling và phân phối công việc.
- `health_monitor.py`: Theo dõi và khôi phục trạng thái node.
- `util.py`: Tiện ích quản lý node.

7.4.2. Crawling

- `crawler.py`: Trình crawler cốt lõi với BeautifulSoup, lưu HTML và trích xuất URL.
- `database_manager.py`: Cơ sở dữ liệu SQLite theo dõi các URL đã crawl.

7.4.3. Điều phối:

- `main.py`: CLI điều khiển toàn hệ thống.
- `generate_urls.py`: Sinh URL và phân phối qua cơ chế round-robin.

7.5. Luồng hoạt động

Bước 1. Khởi động hệ thống:

Người dùng chạy `main.py` và chỉ định số lượng node cần khởi tạo.

Bước 2. Khởi tạo các node:

Các node được tạo trên các cổng từ 5000 trở đi.

Bước 3. Theo dõi:

Hệ thống giám sát trạng thái node và tự động khởi động lại khi có lỗi.

Bước 4. Phân phối yêu cầu:

Yêu cầu crawl được gửi đến các node qua cơ chế round-robin.

Bước 5. Thực hiện crawl:

Mỗi node xử lý URL được chỉ định, lưu nội dung HTML và phân phối child URLs cho các node khác.

8. ĐÁNH GIÁ HIỆU SUẤT

8.1. Điểm mạnh

- Hiệu suất cao trong môi trường cân bằng:
 - Với các node đồng nhất, hệ thống sử dụng thuật toán Round Robin để phân phối tải, đạt hiệu suất tối đa.
- Khả năng song song tốt:
 - Crawler có thể xử lý nhiều URL đồng thời nhờ kiến trúc phân tán và sử dụng asyncio.
- Chịu lỗi ổn định:
 - Health Monitor đảm bảo hệ thống không bị gián đoạn lâu nếu một node gặp lỗi.
- Khả năng mở rộng:
 - Khi thêm nhiều node, throughput tăng gần tuyến tính, miễn là tài nguyên mạng và máy chủ không bị giới hạn.

8.2. Hạn chế

- Hiệu suất giảm khi gặp node lỗi
 - Nếu một node không phản hồi, hệ thống cần thời gian để retry hoặc khởi động lại node, dẫn đến độ trễ tăng.
- Phân phối không tối ưu trong môi trường không đồng nhất
 - Round Robin không tối ưu khi các node có cấu hình khác nhau. Node yếu hơn có thể bị quá tải.
- Chi phí quản lý trạng thái:
 - Với SQLite, việc kiểm tra trạng thái URL có thể trở nên chậm khi số lượng URL rất lớn.
- Hiệu quả retry:
 - Retry nhiều lần cho các URL không khả dụng (ví dụ: bị chặn bởi robots.txt) làm lãng phí tài nguyên.

PHẦN C. TỔNG KẾT – ĐÁNH GIÁ

9. KIẾN THỨC

- Hiểu rõ mô hình hệ thống phân tán (Distributed Systems):
 - Nắm được nguyên tắc thiết kế hệ thống phân tán, bao gồm các khái niệm về tính song song, tính nhất quán, khả năng chịu lỗi và khả năng mở rộng.
 - Tìm hiểu sâu về cách phối hợp các thành phần trong hệ thống để đảm bảo hiệu suất và độ tin cậy.
- Kiến thức về giao thức mạng:
 - Hiểu cơ bản về các giao thức truyền thông như HTTP/HTTPS, TCP/IP, và cách áp dụng chúng trong quá trình phát triển crawler.
 - Nắm bắt vai trò của tầng ứng dụng trong việc trao đổi dữ liệu giữa các máy chủ và client.
- Ứng dụng công nghệ hiện đại:
 - Học cách sử dụng các công nghệ hỗ trợ hệ thống phân tán như Redis (hàng đợi URL), Elasticsearch (lưu trữ và tìm kiếm dữ liệu), và Docker/Kubernetes (triển khai hệ thống).
- Tối ưu hóa hiệu suất hệ thống:
 - Tìm hiểu và áp dụng các thuật toán điều phối công việc để cải thiện tốc độ thu thập dữ liệu.
 - Xây dựng các cơ chế xử lý lỗi nhằm tăng độ ổn định của hệ thống khi đối mặt với lượng dữ liệu lớn.

10. KỸ NĂNG

10.1. Kỹ năng chuyên môn

- Lập trình bằng Python:
 - Sử dụng thành thạo các thư viện Python như Scrapy để thu thập dữ liệu web và Requests để truy cập API.
 - Thiết kế và triển khai mã nguồn đảm bảo tính hiệu quả, dễ mở rộng và bảo trì.
- Tư duy logic và giải quyết vấn đề:
 - Thiết kế luồng xử lý URL sao cho tránh được trùng lặp và tối ưu hóa tài nguyên hệ thống.
 - Phân tích, đánh giá và xử lý các lỗi trong quá trình crawler hoạt động.
- Triển khai và kiểm thử hệ thống:
 - Sử dụng Docker để triển khai hệ thống trên môi trường thực tế, đảm bảo tính khả chuyển và tính đồng nhất.
 - Thực hiện các bài kiểm tra hiệu năng nhằm xác định khả năng đáp ứng của hệ thống trước các tập dữ liệu lớn.

10.2. Kỹ năng mềm

- Làm việc nhóm hiệu quả:
 - Phân công công việc hợp lý giữa các thành viên, phối hợp để hoàn thành dự án đúng hạn.
 - Sử dụng các công cụ quản lý dự án như Trello, GitHub để theo dõi tiến độ và kiểm soát chất lượng mã nguồn.
- Kỹ năng trình bày và báo cáo:
 - Tập trung vào việc trình bày rõ ràng kiến thức và kết quả đạt được trong báo cáo.
 - Luyện tập khả năng giải thích các khía cạnh kỹ thuật phức tạp theo cách dễ hiểu khi thuyết trình.

LỜI CẢM ƠN

Trước hết, nhóm chúng em xin gửi lời cảm ơn sâu sắc đến Tiến sĩ Ngô Thị Hiền, giảng viên hướng dẫn học phần Hệ thống và mạng máy tính, thuộc Khoa Toán-Tin, Trường Đại học Bách Khoa Hà Nội. Với sự tận tình hướng dẫn, những góp ý quý báu cùng sự hỗ trợ xuyên suốt quá trình thực hiện, cô đã giúp nhóm chúng em hoàn thành đề tài "Thiết kế hệ thống hiệu suất cao dựa trên trình thu thập dữ liệu thông tin web phân tán".

Sự chỉ bảo chi tiết, sát sao và những gợi mở thực tiễn của cô không chỉ giúp nhóm hiểu sâu hơn về kiến thức lý thuyết mà còn định hướng rõ ràng trong quá trình nghiên cứu và triển khai. Những đóng góp của cô là động lực quan trọng để nhóm chúng em vượt qua khó khăn, hoàn thiện đề tài một cách tốt nhất.

Nhóm chúng em cũng xin gửi lời cảm ơn chân thành đến các thầy cô trong Khoa Toán-Tin, những người đã truyền đạt nền tảng kiến thức và kỹ năng quan trọng trong suốt quá trình học tập tại trường, tạo tiền đề vững chắc để nhóm có thể tiếp cận và giải quyết bài toán nghiên cứu của mình.

Bên cạnh đó, nhóm cũng xin cảm ơn sự hỗ trợ, động viên từ gia đình và bạn bè – những người luôn đồng hành cùng nhóm trong suốt thời gian thực hiện đề tài.

Đề tài này không chỉ là kết quả của quá trình học tập và nghiên cứu mà còn là cơ hội để nhóm chúng em học hỏi thêm nhiều kiến thức mới, rèn luyện kỹ năng làm việc nhóm, và tích lũy kinh nghiệm cho chặng đường học tập và phát triển nghề nghiệp sau này.

Nhóm chúng em xin chân thành cảm ơn!

NHÓM 12
Group 12

(Nhóm 12)

TÀI LIỆU THAM KHẢO

- **Ian Robinson, Jim Webber, Emil Eifrem.** *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, 2020.
- **Benoit Dupont, Michael Backes, Rainer Böhme.** *Handbook of Web Crawlers and Data Aggregation Techniques*. Springer, 2021.
- **Günther, O., Schmidt, H. W.** *Distributed Systems: Principles and Paradigms*. Addison-Wesley, 2022.
- **Scrapy Documentation.** *Scrapy 2.9.0 Documentation*. Truy cập tại: <https://docs.scrapy.org>
- **Apache Nutch Project.** *Apache Nutch Documentation*. Truy cập tại: <https://nutch.apache.org>
- **Tan, K. M., Tay, J. Y.** *Optimizing Distributed Web Crawlers*. ACM Digital Library, 2020.
- **Shroff, G.** *Enterprise Cloud Computing: Technology, Architecture, Applications*. Cambridge University Press, 2021.
- **Cheng, G., Qu, Y.** *Efficient Data Extraction for E-commerce Websites*. IEEE Transactions on Knowledge and Data Engineering, 2023.
- **Meng, W., Lu, W., & Liu, C.** *Principles of High-Performance Web Crawlers*. Journal of Web Engineering, 2022.
- **Nguyen, D. T.**, *Distributed Web Crawling System for Big Data*. Proceedings of the Vietnam Symposium on Information Systems, 2021.
- **TS. Ngô Thị Hiền**, *Slide bài giảng học phần Hệ thống mạng máy tính* (Khoa Toán-Tin, Đại học Bách Khoa Hà Nội)
- **TS. Nguyễn Đình Hân**, *Slide bài giảng học phần Hệ thống mạng máy tính* (Khoa Toán-Tin, Đại học Bách Khoa Hà Nội)

PHỤ LỤC



(Mã QR: Thư mục toàn bộ báo cáo của nhóm – Google Drive)

(Sheet: Kết quả đánh giá thành viên của Nhóm. Dựa trên Google Form đánh giá ẩn danh.)



≈ HẾT ≈