BỘ GIAO THÔNG VẬN TẢI
TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

PHÂN TÍCH VÀ TỐI ƯU HÓA THUẬT TOÁN TÌM KIẾM TRÊN CƠ SỞ DỮ LIỆU

Tên học phần: Thực tập tốt nghiệp

Giảng viên hướng dẫn: TS Lê Quốc Tuấn

Người thực hiện

MSSV	Họ và tên
2151150040	Nguyễn Duy Nhật Huy

Thành Phố Hồ Chí Minh – 2024

Lời cam đoan

Tôi, người thực hiện đề tài, xác nhận rằng đây là kết quả của quá trình nghiên cứu và phát triển ứng dụng của tôi, được hướng dẫn và giám sát bởi TS. Lê Quốc Tuấn. Quá trình lựa chọn và thực hiện đề tài đã xuất phát từ nhu cầu thực tiễn và mong muốn sáng tạo của sinh, nhằm tạo ra một sản phẩm phần mềm có giá trị thực tế.

Tôi cam kết rằng nội dung nghiên cứu và các kết quả đều phản ánh trung thực và chưa từng được công bố trước đây. Các dữ liệu và thông tin sử dụng trong quá trình phân tích và đánh giá được tôi thu thập từ nhiều nguồn đáng tin cậy, đồng thời sẽ được chi tiết và minh bạch trong phần tài liệu tham khảo.

Tôi hoàn toàn chịu trách nhiệm về chất lượng và tính hợp lý của sản phẩm đã phát triển. Trong trường hợp phát hiện bất kỳ hành vi không trung thực hoặc vi phạm nào, tôi cam kết sẽ đối mặt và giải quyết mọi vấn đề một cách chín chắn và hiệu quả. Trường Đại học Giao thông vận tải TP Hồ Chí Minh không có liên quan đến những vấn đề tác quyền hoặc bản quyền nảy sinh từ sản phẩm phần mềm này do tôi tạo ra trong quá trình nghiên cứu và phát triển. Tôi cam kết tiếp tục duy trì tinh thần sáng tạo và chất lượng cao trong các hoạt động phát triển phần mềm sắp tới.

TP. Hồ Chí Minh, ngày 28 tháng 12 năm 2024

Người thực hiện đề tài

Nguyễn Duy Nhật Huy

Lời cảm ơn

Tôi, người thực hiện đề tài, xin bày tỏ lòng biết ơn trân trọng nhất đến TS. Lê Quốc Tuấn, người đã dành thời gian để hướng dẫn và hỗ trợ tôi. Những kiến thức và sự chỉ dẫn chi tiết của thầy không chỉ là nguồn động viên mà còn giúp tôi phát triển kỹ năng và hiểu biết sâu sắc hơn về lĩnh vực tôi nghiên cứu.

Chúng tôi cũng muốn gửi lời cảm ơn chân thành đến toàn thể các thầy cô giảng viên Khoa Công nghệ thông tin, cũng như Trường Đại học Giao thông vận tải TP Hồ Chí Minh. Những kiến thức quý báu mà các thầy cô đã chia sẻ đã là nền tảng vững chắc, giúp tôi hoàn thành đồ án này.

Mặc dù đã cố gắng hết sức, nhưng với hạn chế về kiến thức và kinh nghiệm, tôi hiểu rằng có thể xuất hiện thiếu sót. Do đó, tôi mong nhận được những lời nhận xét, góp ý quý báu từ quý thầy cô để đồ án ngày càng hoàn thiện và phát triển. Xin chân thành cảm ơn

Tóm tắt

Sau quá trình nghiên cứu, tìm hiểu và phát triển đề tài "Phân tích và tối ưu hóa thuật toán tìm kiếm trên cơ sở dữ liệu lớn", tôi tổng kết lại báo cáo Thực tập tốt nghiệp về đề tài này, bao gồm các nội dung sau đây:

- Chương 1: Giới thiệu đề tài
  Nêu lên thực trạng, lý do thực hiện đề tài, mục tiêu, phạm vi và phương pháp nghiên cứu
- Chương 2: Cơ sở lý thuyết
  Mô tả về thuật toán tìm kiếm, dữ liệu lớn (Big Data), các phương pháp và thuật toán tìm kiếm phổ biến (như B-Trees, Hash-based indexing, ElasticSearch).
- Chương 3: Phân tích vấn đề
  Mô tả bài toán cụ thể, đánh giá hiệu suất của các thuật toán tìm kiếm hiện tại trên dữ liệu lớn và phân tích hạn chế của các thuật toán hiện có.
- Chương 4: Tối ưu hóa thuật toán
  Mô tả về lý thuyết tối ưu hóa, cung cấp giải pháp sử dụng mô hình máy học và giả lập mô hình.
- Chương 5: Thực nghiệm
  Tiến hành thực nghiệm các thuật toán trên môi trường đã được cài đặt.
- Chương 6: Kết quả và đánh giá
  Mô tả môi trường thực nghiệm, kết quả thực hiện, so sánh kết quả giữa thuật toán gốc và thuật toán tối ưu.
- Chương 7: Kết luận và phương hướng phát triển
  Tóm tắt các điểm chính trong đề tài và đề xuất hướng nghiên cứu tiếp theo.

Mục lục

Chương 1: Giới thiệu đề tài

Trong thời đại bùng nổ thông tin như hiện nay, dữ liệu được tạo ra và lưu trữ ngày càng gia tăng một cách nhanh chóng từ các tác nhân khác nhau. Các tổ chức, doanh nghiệp và cá nhân đều sử dụng cơ sở dữ liệu lớn (Big Data) để thu thập, phân tích, khai thác thông tin nhằm phục vụ cho việc ra quyết định, nghiên cứu thị trường, dự đoán xu hướng, cũng như giải quyết hàng loạt vấn đề phức tạp trong thực tiễn. Do đó, việc tìm kiếm, truy xuất thông tin một cách hiệu quả từ nguồn dữ liệu khổng lồ này trở thành một nhu cầu cấp thiết.

Tuy nhiên, cùng với sự phát triển mạnh mẽ của dữ liệu, các thuật toán tìm kiếm truyền thống đang gặp nhiều thách thức. Khi khối lượng dữ liệu tăng lên, thời gian truy xuất và chi phí lưu trữ cũng trở nên đắt đỏ, đòi hỏi các hệ thống phải tối ưu hóa các chỉ số như tốc độ truy vấn, hiệu suất bộ nhớ và độ chính xác. Vì vậy, nghiên cứu về "Phân tích và tối ưu hóa thuật toán tìm kiếm trên cơ sở dữ liệu lớn" không chỉ là yêu cầu mang tính học thuật mà còn có ý nghĩa thực tiễn rất cao trong bối cảnh kinh tế – xã hội hiện nay.

Bên cạnh đó, các công nghệ và kiến trúc hệ thống mới xuất hiện như ElasticSearch, Apache Solr hay kỹ thuật lập chỉ mục (indexing) dựa trên cây B-Tree, Hash-based indexing, cùng các chiến lược tìm kiếm phân tán đã mở ra nhiều hướng đi mới. Những cải tiến này hướng tới việc xử lý dữ liệu theo thời gian thực, truy xuất thông tin nhanh chóng và hỗ trợ người dùng đưa ra quyết định kịp thời. Từ đó, các doanh nghiệp có thể tận dụng ưu thế cạnh tranh trong kinh doanh, cơ quan nghiên cứu có thể xử lý nguồn dữ liệu khổng lồ một cách hiệu quả hơn.

Chính vì vậy, đề tài nghiên cứu này sẽ tập trung phân tích và tối ưu hóa các thuật toán tìm kiếm cho cơ sở dữ liệu lớn, bao gồm cả việc so sánh ưu nhược điểm của chúng, tiến hành xây dựng một mô hình thử nghiệm nhỏ, triển khai các thuật toán trên tập dữ liệu lớn, sau đó đo lường các chỉ số hiệu năng trước và sau quá trình tối ưu.

1. Đặt vấn đề

Hiện nay, các tổ chức và doanh nghiệp phải đối mặt với nhu cầu truy cập dữ liệu một cách tức thời, từ những câu lệnh tìm kiếm đơn giản đến các hệ thống phân tích phức tạp với hàng trăm triệu bản ghi. Người dùng không những kỳ vọng có được kết quả

một cách nhanh nhất, mà còn đòi hỏi độ chính xác cao, giảm thiểu sai sót và tối ưu hóa tài nguyên phần cứng.

Nếu như trước đây, các thuật toán tìm kiếm truyền thống như tuyến tính (linear search) hoặc nhị phân (binary search) hoạt động ổn định trên quy mô dữ liệu vừa và nhỏ, thì hiện tại, khi quy mô dữ liệu phát triển lên mức hàng triệu, hàng tỷ bản ghi, vấn đề về hiệu suất trở nên vô cùng nan giải. Thời gian thực thi truy vấn bị kéo dài, yêu cầu khả năng phân tán, tính sẵn sàng cao của hệ thống đòi hỏi những cải tiến vượt bậc trong thiết kế, kiến trúc và cách thức triển khai các giải pháp tìm kiếm.

Các phương pháp index như B-Tree, Hash-based index, hoặc các giải pháp tìm kiếm phân tán như ElasticSearch, Apache Solr ra đời nhằm giải quyết những nút thắt về hiệu năng. Tuy nhiên, khi áp dụng vào những khối dữ liệu đa dạng, nhiều dạng cấu trúc khác nhau (structured, semi-structured, unstructured) hoặc có tốc độ sinh dữ liệu cao, thì bản thân các phương pháp này cũng có những hạn chế nhất định. Chẳng hạn, B-Tree phù hợp cho những cấu trúc dữ liệu tuần tự, Hash-based index cho truy vấn tìm kiếm chính xác, hay ElasticSearch/Solr sử dụng inverted index để tối ưu cho tìm kiếm văn bản. Mỗi cách tiếp cận đều có ưu, nhược điểm riêng và đòi hỏi quá trình tối ưu hóa sao cho phù hợp với đặc thù của bài toán.

Trong bối cảnh đó, câu hỏi được đặt ra là: Làm thế nào để tìm kiếm một cách nhanh chóng, hiệu quả, đồng thời duy trì được tính linh hoạt và độ chính xác cao trên cơ sở dữ liệu lớn? Quá trình phân tích và tối ưu hóa các thuật toán tìm kiếm hiện có, đề xuất giải pháp kết hợp (nếu cần) giữa các phương pháp chỉ mục và triển khai hệ thống phân tán sẽ giúp giải quyết vấn đề này.

2. Mục tiêu nghiên cứu

Mục tiêu của đề tài nghiên cứu “Phân tích và tối ưu hóa thuật toán tìm kiếm trên cơ sở dữ liệu lớn” được cụ thể hóa như sau:

2.1 Phân tích các thuật toán tìm kiếm phổ biến

Làm rõ nguyên lý hoạt động, cấu trúc và cơ chế quản lý dữ liệu của các phương pháp như B-Tree, Hash-based indexing, ElasticSearch, Apache Solr,…

Đánh giá ưu điểm và hạn chế của từng thuật toán khi áp dụng trên dữ liệu có quy mô lớn và tốc độ sinh dữ liệu cao.

2.2 Đề xuất các phương pháp tối ưu hóa

Tìm hiểu và đề xuất cách cải tiến, tinh chỉnh, tối ưu cấu trúc và thuật toán để nâng cao tốc độ truy vấn, tiết kiệm tài nguyên bộ nhớ và giảm độ trễ.

Nghiên cứu các kỹ thuật phân tán, cân bằng tải, hay cơ chế bộ nhớ đệm (cache) để gia tăng hiệu năng trong môi trường dữ liệu lớn.

2.3 Xây dựng mô hình/hệ thống nhỏ để thử nghiệm

Triển khai thử nghiệm trên một tập dữ liệu lớn, đồng thời ghi nhận các chỉ số về thời gian truy vấn, hiệu suất bộ nhớ, độ chính xác.

So sánh hiệu suất của các thuật toán trước và sau khi áp dụng tối ưu để xác định mức độ cải thiện thực tế.

3. Phạm vi và giới hạn nghiên cứu

3.1 Phạm vi nghiên cứu

Nghiên cứu tập trung vào các cơ sở dữ liệu lớn, bao gồm cả dữ liệu có cấu trúc (structured data), bán cấu trúc (semi-structured data) và phi cấu trúc (unstructured data). Trong phạm vi luận văn/đề tài này, sẽ ưu tiên xem xét các dạng dữ liệu văn bản, nhật ký (logs), hoặc thông tin giao dịch (transactions) thường gặp.

Các thuật toán và cấu trúc chỉ mục được lựa chọn để phân tích, đánh giá sẽ bao gồm B-Trees, Hash-based indexing, và các giải pháp tìm kiếm phân tán (ElasticSearch, Apache Solr). Đây là những phương pháp, công nghệ phổ biến, đại diện cho nhiều trường hợp sử dụng thực tế khác nhau.

3.2 Giới hạn nghiên cứu

Do thời gian và nguồn lực có hạn, nghiên cứu sẽ không triển khai trên quy mô lớn với dữ liệu khổng lồ của những hãng công nghệ lớn, mà thay vào đó, áp dụng trên một mô hình thực nghiệm cỡ trung bình, có thể dao động từ vài triệu đến hàng chục triệu bản ghi.

Không đi sâu vào tối ưu hóa phần cứng hay hạ tầng mạng, mà chủ yếu tập trung vào việc tối ưu cấp độ thuật toán, cấu trúc dữ liệu và triển khai thử nghiệm ở quy mô vừa đủ để đánh giá xu hướng và hiệu năng.

Độ chính xác trong nghiên cứu có thể bị ảnh hưởng bởi một số yếu tố khách quan như chất lượng dữ liệu đầu vào, độ tin cậy của hạ tầng, sai số trong quá trình đo đạc. Nghiên cứu sẽ cố gắng kiểm soát và giới hạn các sai số này ở mức thấp nhất.

4. Phương pháp nghiên cứu

4.1 Nghiên cứu tài liệu tổng quan

Thu thập, nghiên cứu các công trình khoa học, bài báo, sách, tài liệu chuyên ngành về các kỹ thuật tìm kiếm, cấu trúc index và các giải pháp tìm kiếm phân tán như ElasticSearch.

Phân tích các khái niệm quan trọng như B-Tree, Hash-based indexing, inverted index, map-reduce, sharding, replication, caching… để có cái nhìn tổng quan về cách thức và phạm vi ứng dụng của mỗi kỹ thuật.

4.2 Phân tích và so sánh thuật toán

Xây dựng tiêu chí so sánh dựa trên các chỉ số về tốc độ truy vấn, mức sử dụng bộ nhớ, độ chính xác của thuật toán.

Thực hiện phân tích lý thuyết để làm rõ vì sao một thuật toán, cấu trúc dữ liệu có thể phù hợp hơn trong một số trường hợp cụ thể.

4.3 Triển khai, đo lường và đánh giá

Tiến hành chạy thử nghiệm trên mô hình đã thiết kế, thu thập dữ liệu về thời gian phản hồi (latency), thời gian xử lý trung bình (average query time), mức sử dụng bộ nhớ (memory usage), độ chính xác (precision, recall).

Chương 2: Cơ sở lý thuyết

1. Giới thiệu về thuật toán tìm kiếm

Thuật toán tìm kiếm là một tập hợp các quy trình hoặc bước được thiết kế để xác định vị trí của một phần tử cụ thể trong một cấu trúc dữ liệu, hoặc để tìm kiếm các phần tử đáp ứng một điều kiện nhất định trong tập hợp dữ liệu. Trong ngữ cảnh cơ sở dữ liệu lớn, thuật toán tìm kiếm không chỉ nhằm mục đích tìm ra dữ liệu một cách nhanh chóng mà còn phải đảm bảo hiệu quả về mặt tài nguyên và khả năng mở rộng khi xử lý các khối lượng dữ liệu khổng lồ.[1]

Các thuật toán tìm kiếm có thể được phân loại dựa trên cấu trúc dữ liệu mà chúng thao tác, chẳng hạn như mảng, danh sách liên kết, cây, bảng băm, hay các hệ thống tìm kiếm phân tán. Mỗi loại thuật toán có những ưu điểm và hạn chế riêng, phù hợp với các yêu cầu và tình huống cụ thể trong việc quản lý và truy xuất dữ liệu.

Các Thuật Toán Tìm Kiếm Phổ Biến:

- Thuật Toán Tìm Kiếm Tuyến Tính (Linear Search): Là thuật toán tìm kiếm đơn giản nhất, trong đó mỗi phần tử trong cấu trúc dữ liệu được kiểm tra lần lượt từ đầu đến cuối cho đến khi tìm thấy phần tử cần tìm hoặc kết thúc danh sách. Thuật toán này có độ phức tạp thời gian là O(n), phù hợp với các tập dữ liệu nhỏ hoặc không có cấu trúc sắp xếp.

- Thuật Toán Tìm Kiếm Nhị Phân (Binary Search): Áp dụng cho các cấu trúc dữ liệu đã được sắp xếp, thuật toán này chia đôi khoảng tìm kiếm ở mỗi bước, giảm số lượng so sánh cần thiết. Độ phức tạp thời gian của thuật toán là O(log n), nhanh hơn đáng kể so với tìm kiếm tuyến tính đối với các tập dữ liệu lớn. Tuy nhiên, yêu cầu dữ liệu phải được sắp xếp trước.

- B-Tree: Là một cấu trúc dữ liệu dạng cây cân bằng, B-Tree được sử dụng rộng rãi trong các hệ quản trị cơ sở dữ liệu và hệ thống tệp để hỗ trợ tìm kiếm, chèn và xóa dữ liệu một cách hiệu quả. B-Tree cho phép truy cập dữ liệu với độ phức tạp thời gian O(log n) và tối ưu hóa việc sử dụng bộ nhớ đệm.

- Hash-based Indexing: Sử dụng hàm băm để ánh xạ các khóa dữ liệu đến các vị trí lưu trữ trong bảng băm, giúp truy xuất dữ liệu một cách nhanh chóng với

độ phức tạp thời gian trung bình là O(1). Tuy nhiên, hiệu suất có thể giảm xuống trong trường hợp có nhiều xung đột hàm băm.

- Inverted Index (ElasticSearch, Apache Solr): Được sử dụng chủ yếu trong các hệ thống tìm kiếm văn bản, inverted index ánh xạ từ các từ khóa đến các tài liệu chứa chúng. Điều này giúp thực hiện các truy vấn tìm kiếm văn bản một cách nhanh chóng và hiệu quả, đặc biệt trong các cơ sở dữ liệu lớn và phức tạp.

2. Tổng quan về dữ liệu lớn (Big Data)

2.1 Giới thiệu về dữ liệu lớn (Big Data)

Trong kỷ nguyên số hóa hiện nay, dữ liệu được tạo ra với tốc độ và khối lượng chưa từng thấy. Dữ liệu lớn (Big Data) không chỉ đơn thuần là lượng dữ liệu khổng lồ mà còn bao gồm các loại dữ liệu đa dạng và phức tạp từ nhiều nguồn khác nhau. Big Data đóng vai trò quan trọng trong việc hỗ trợ các quyết định chiến lược, tối ưu hóa quy trình kinh doanh, và khám phá các mẫu thông tin ẩn chứa trong dữ liệu.

Big Data được ứng dụng rộng rãi trong nhiều lĩnh vực như y tế, tài chính, bán lẻ, sản xuất, và truyền thông, giúp các tổ chức khai thác giá trị từ dữ liệu để đạt được lợi thế cạnh tranh. Tuy nhiên, việc quản lý, lưu trữ, xử lý và phân tích Big Data cũng đặt ra nhiều thách thức về kỹ thuật và quản trị, đòi hỏi các giải pháp tiên tiến và hiệu quả.[2]

2.2 Các đặc trưng của dữ liệu lớn

Để hiểu rõ hơn về Big Data, các nhà nghiên cứu đã xác định ra một số đặc trưng chính, thường được gọi là "5V" của Big Data:

a. Khối Lượng (Volume)

Khối lượng dữ liệu là đặc trưng cơ bản nhất của Big Data. Dữ liệu được tạo ra từ nhiều nguồn khác nhau như mạng xã hội, cảm biến IoT, giao dịch thương mại điện tử, và các hệ thống thông tin doanh nghiệp. Khối lượng dữ liệu lớn đòi hỏi các hệ thống lưu trữ và xử lý phải có khả năng mở rộng để đáp ứng nhu cầu ngày càng tăng.

Ví dụ: Facebook xử lý hàng trăm terabyte dữ liệu mỗi ngày từ các hoạt động của người dùng như bài viết, hình ảnh, video và tương tác xã hội.

b. Tốc Độ (Velocity)

Tốc độ tạo ra và xử lý dữ liệu là yếu tố quan trọng trong Big Data. Dữ liệu phải được thu thập, xử lý và phân tích trong thời gian thực hoặc gần thời gian thực để cung cấp thông tin kịp thời cho các quyết định.

Ví dụ: Hệ thống giao dịch chứng khoán cần xử lý hàng triệu giao dịch trong mỗi giây để đảm bảo tính chính xác và kịp thời của thông tin thị trường.

c. Đa Dạng (Variety)

Big Data không chỉ bao gồm dữ liệu có cấu trúc mà còn dữ liệu bán cấu trúc và phi cấu trúc. Dữ liệu có thể đến từ các định dạng khác nhau như văn bản, hình ảnh, video, âm thanh, và dữ liệu từ các thiết bị cảm biến.

Ví dụ: Trong lĩnh vực y tế, dữ liệu có thể bao gồm hồ sơ bệnh án điện tử (cấu trúc), hình ảnh y tế (phi cấu trúc), và dữ liệu từ thiết bị theo dõi sức khỏe (bán cấu trúc).

d. Độ Chính Xác (Veracity)

Độ chính xác đề cập đến chất lượng và độ tin cậy của dữ liệu. Dữ liệu lớn thường chứa nhiều thông tin không chính xác, nhiễu loạn hoặc mâu thuẫn, do đó việc đảm bảo độ chính xác là một thách thức lớn trong quá trình xử lý và phân tích.

Ví dụ: Dữ liệu từ mạng xã hội có thể chứa thông tin sai lệch hoặc không đầy đủ, đòi hỏi các thuật toán lọc và làm sạch dữ liệu hiệu quả.

e. Giá Trị (Value)

Giá trị của Big Data được xác định bởi khả năng khai thác thông tin hữu ích từ dữ liệu để tạo ra lợi ích kinh doanh hoặc xã hội. Việc chuyển đổi dữ liệu thô thành thông tin có giá trị yêu cầu các kỹ thuật phân tích và trực quan hóa dữ liệu tiên tiến.

Ví dụ: Phân tích dữ liệu khách hàng để dự đoán xu hướng mua sắm và cá nhân hóa chiến lược tiếp thị, từ đó tăng doanh thu và sự hài lòng của khách hàng.

2.3 Những thách thức trong việc tìm kiếm trên dữ liệu lớn

a. Khối Lượng Dữ Liệu Lớn (Volume)

Dữ liệu lớn thường bao gồm hàng terabyte đến petabyte thông tin, tạo ra bởi các nguồn dữ liệu đa dạng như mạng xã hội, cảm biến IoT, giao dịch thương mại điện tử, và nhiều hệ thống khác. Khối lượng dữ liệu khổng lồ này đặt ra yêu cầu cao về khả năng lưu trữ, xử lý và truy xuất dữ liệu một cách hiệu quả.

Thách Thức:

- Hiệu Suất Truy Vấn: Các thuật toán tìm kiếm truyền thống như tìm kiếm tuyến tính (Linear Search) hay thậm chí là tìm kiếm nhị phân (Binary Search) không còn phù hợp với khối lượng dữ liệu lớn, do độ phức tạp thời gian của chúng không thể đáp ứng được yêu cầu về tốc độ truy vấn.
- Quản Lý Lưu Trữ: Việc lưu trữ và quản lý khối lượng dữ liệu lớn đòi hỏi các hệ thống lưu trữ phân tán và khả năng mở rộng (scalability) cao, như Hadoop Distributed File System (HDFS) hay các dịch vụ đám mây như Amazon S3.

Ví Dụ: Một hệ thống tìm kiếm văn bản như Elasticsearch phải xử lý hàng tỷ tài liệu, đảm bảo rằng các truy vấn tìm kiếm được thực hiện trong thời gian thực hoặc gần thời gian thực.

b. Tốc Độ Dữ Liệu (Velocity)

Dữ liệu không chỉ lớn mà còn được tạo ra và cập nhật với tốc độ cao. Điều này yêu cầu các hệ thống tìm kiếm phải xử lý và cập nhật dữ liệu một cách nhanh chóng để đảm bảo tính nhất quán và khả năng truy xuất dữ liệu mới nhất.

Thách Thức:

- Xử Lý Dữ Liệu Thời Gian Thực: Các thuật toán tìm kiếm phải có khả năng cập nhật chỉ mục (index) và xử lý các truy vấn mới ngay lập tức khi dữ liệu được thêm vào.
- Đồng Bộ Hóa Dữ Liệu: Đảm bảo rằng các nút trong hệ thống phân tán luôn đồng bộ hóa để tránh việc truy vấn trả về kết quả không chính xác hoặc không đầy đủ.

Ví Dụ: Trong các hệ thống giao dịch tài chính, việc xử lý hàng triệu giao dịch mỗi giây yêu cầu các thuật toán tìm kiếm có thể cập nhật và truy xuất dữ liệu gần như ngay lập tức.

c. Đa Dạng Dữ Liệu (Variety)

Dữ liệu lớn bao gồm nhiều loại dữ liệu khác nhau như dữ liệu có cấu trúc (structured data), dữ liệu bán cấu trúc (semi-structured data), và dữ liệu phi cấu trúc (unstructured data). Sự đa dạng này tạo ra những khó khăn trong việc lập chỉ mục và tìm kiếm hiệu quả.

Thách Thức:

- Phân Tích và Chuyển Đổi Dữ Liệu: Các hệ thống tìm kiếm cần có khả năng phân tích và chuyển đổi dữ liệu từ các định dạng khác nhau thành một định dạng thống nhất để dễ dàng lập chỉ mục và truy vấn.
- Xây Dựng Chỉ Mục Phù Hợp: Việc thiết kế các cấu trúc chỉ mục (index) phù hợp cho từng loại dữ liệu là một thách thức lớn. Ví dụ, inverted index thích hợp cho dữ liệu văn bản, trong khi các dữ liệu số hoặc dữ liệu không gian yêu cầu các cấu trúc chỉ mục khác như R-Tree hoặc KD-Tree.

Ví Dụ: Trong lĩnh vực y tế, dữ liệu bao gồm hồ sơ bệnh án điện tử (cấu trúc), hình ảnh y tế (phi cấu trúc), và dữ liệu từ các thiết bị theo dõi sức khỏe (bán cấu trúc), đòi hỏi hệ thống tìm kiếm phải xử lý và lập chỉ mục đa dạng các loại dữ liệu này một cách hiệu quả.

d. Tính Phân Tán và Đồng Bộ (Distributed and Consistency)

Các hệ thống dữ liệu lớn thường được triển khai trên các cụm máy tính phân tán để tăng khả năng mở rộng và hiệu suất. Tuy nhiên, việc duy trì tính nhất quán và đồng bộ hóa dữ liệu trên nhiều nút là một thách thức đáng kể.

Thách Thức:

- Quản Lý Dữ Liệu Phân Tán: Đảm bảo rằng các bản sao dữ liệu trên các nút khác nhau luôn được cập nhật và đồng bộ, tránh tình trạng dữ liệu bị mâu thuẫn hoặc lỗi thời.

- Chia Tải (Load Balancing): Phân phối đều tải truy vấn và xử lý dữ liệu giữa các nút để tránh tình trạng quá tải ở một số nút nhất định, từ đó tối ưu hóa hiệu suất tổng thể của hệ thống.

Ví Dụ: Hệ thống tìm kiếm phân tán như Apache Solr hoặc Elasticsearch phải đảm bảo rằng khi một tài liệu mới được thêm vào, tất cả các bản sao của nó trên các nút khác nhau đều được cập nhật kịp thời để tránh việc truy vấn trả về kết quả không đầy đủ.

e. Tối Ưu Hóa Chỉ Mục (Index Optimization)

Việc xây dựng và duy trì các chỉ mục hiệu quả là yếu tố then chốt để đảm bảo hiệu suất tìm kiếm cao. Tuy nhiên, trên dữ liệu lớn, việc tối ưu hóa chỉ mục trở nên phức tạp hơn do yêu cầu về tốc độ cập nhật và khả năng mở rộng.

Thách Thức:

- Kích Thước Chỉ Mục: Với khối lượng dữ liệu lớn, kích thước chỉ mục có thể trở nên cực kỳ lớn, gây ra vấn đề về lưu trữ và hiệu suất truy xuất.
- Cập Nhật Chỉ Mục: Đảm bảo rằng chỉ mục luôn được cập nhật kịp thời khi dữ liệu thay đổi mà không làm giảm hiệu suất truy vấn.

Ví Dụ: Trong Elasticsearch, việc tối ưu hóa cấu hình chỉ mục và sử dụng các kỹ thuật như sharding và replication là cần thiết để quản lý kích thước và hiệu suất của chỉ mục trên dữ liệu lớn.

f. Bảo Mật và Quyền Riêng Tư (Security and Privacy)

Với lượng dữ liệu lớn thường đi kèm với các dữ liệu nhạy cảm và thông tin cá nhân, việc bảo vệ dữ liệu khỏi các mối đe dọa an ninh và đảm bảo quyền riêng tư của người dùng là một thách thức quan trọng.

Thách Thức:

- Mã Hóa Dữ Liệu: Đảm bảo rằng dữ liệu được mã hóa cả khi lưu trữ và khi truyền tải để ngăn chặn việc truy cập trái phép.

- Kiểm Soát Truy Cập: Thiết lập các cơ chế kiểm soát truy cập chặt chẽ để đảm bảo rằng chỉ những người dùng được ủy quyền mới có thể truy cập và thực hiện các truy vấn tìm kiếm trên dữ liệu.

Ví Dụ: Trong các hệ thống y tế, dữ liệu bệnh nhân phải được bảo vệ nghiêm ngặt để tuân thủ các quy định về quyền riêng tư và bảo mật thông tin cá nhân.

g. Độ Trễ (Latency)

Độ trễ trong việc phản hồi các truy vấn tìm kiếm là một yếu tố quan trọng ảnh hưởng đến trải nghiệm người dùng. Đối với các ứng dụng yêu cầu thời gian thực hoặc gần thời gian thực, việc giảm thiểu độ trễ là thiết yếu.

Thách Thức:

- Tối Ưu Hóa Truy Vấn: Các thuật toán tìm kiếm phải được thiết kế để thực hiện các truy vấn một cách nhanh chóng và hiệu quả.
- Phân Tán Tính Toán: Sử dụng các kỹ thuật phân tán tính toán để giảm tải xử lý trên từng nút và tăng tốc độ phản hồi.

Ví Dụ: Trong các ứng dụng thương mại điện tử, việc tìm kiếm sản phẩm nhanh chóng và chính xác giúp cải thiện trải nghiệm mua sắm của khách hàng và tăng tỷ lệ chuyển đổi.

Chương 3: Phân tích vấn đề

1. Mô tả bài toán

Trong bối cảnh dữ liệu ngày càng trở nên phong phú và phức tạp, việc quản lý và truy xuất thông tin từ các cơ sở dữ liệu lớn (Big Data) đã trở thành một thách thức đáng kể đối với các tổ chức và doanh nghiệp. Bài toán chính mà đề tài nghiên cứu "Phân tích và tối ưu hóa thuật toán tìm kiếm trên cơ sở dữ liệu lớn" hướng tới là:

Làm thế nào để thiết kế và triển khai các thuật toán tìm kiếm hiệu quả, tối ưu hóa thời gian truy vấn và sử dụng tài nguyên hệ thống trên các cơ sở dữ liệu lớn với khối lượng, tốc độ và đa dạng dữ liệu ngày càng tăng?

Yêu cầu cụ thể:

- Tốc độ: Thuật toán cần tối ưu hóa thời gian thực thi truy vấn, giảm thiểu độ trễ (latency) để mang lại trải nghiệm tốt cho người dùng. Cần đảm bảo tốc độ truy vấn nhanh chóng ngay cả khi quy mô dữ liệu tăng lên.
- Độ chính xác: Kết quả tìm kiếm cần có độ chính xác cao, trả về đúng và đủ các kết quả liên quan đến truy vấn của người dùng, hạn chế tối đa kết quả sai lệch hoặc không liên quan.
- Tối ưu tài nguyên: Thuật toán cần sử dụng hiệu quả tài nguyên hệ thống (CPU, bộ nhớ, lưu trữ), tránh lãng phí và giảm chi phí vận hành.

Dữ liệu đầu vào:

- Loại dữ liệu: Nghiên cứu này sẽ tập trung vào các loại dữ liệu phổ biến trong môi trường Big Data như:
    - Dữ liệu văn bản (Text Data): Bao gồm các tài liệu, bài viết, email, tin nhắn, v.v.
    - Dữ liệu nhật ký (Log Data): Ghi lại hoạt động của hệ thống, ứng dụng, thiết bị, v.v.
    - Dữ liệu giao dịch (Transaction Data): Thông tin về các giao dịch mua bán, thanh toán, chuyển khoản, v.v.

- Quy mô dữ liệu: Để đánh giá hiệu quả của thuật toán, nghiên cứu sẽ sử dụng tập dữ liệu có kích thước lớn, mô phỏng môi trường Big Data thực tế. Quy mô dữ liệu dự kiến từ vài triệu đến hàng chục triệu bản ghi.
- Định dạng dữ liệu: Dữ liệu đầu vào có thể ở các định dạng khác nhau như CSV, JSON, XML, v.v.

Mục tiêu:

- Đề xuất được thuật toán tìm kiếm tối ưu cho cơ sở dữ liệu lớn, đáp ứng các yêu cầu về tốc độ, khả năng mở rộng, độ chính xác và tối ưu tài nguyên.
- Xây dựng mô hình thử nghiệm chứng minh tính khả thi và hiệu quả của thuật toán đề xuất.
- Cung cấp báo cáo phân tích chi tiết về hiệu năng của các thuật toán, làm cơ sở cho việc lựa chọn và triển khai giải pháp tìm kiếm phù hợp trong thực tế.

2. B-tree

2.1. Giới thiệu về B-Tree

2.1.1 Khái niệm và vai trò

B-Tree là một cấu trúc dữ liệu dạng cây tự cân bằng, được thiết kế để tối ưu hóa việc lưu trữ và truy vấn dữ liệu, đặc biệt hiệu quả trên các thiết bị lưu trữ thứ cấp như đĩa cứng. B-Tree là dạng tổng quát hóa của cây tìm kiếm nhị phân (Binary Search Tree - BST), cho phép mỗi nút có nhiều hơn hai nút con. Cấu trúc này giúp giảm chiều cao của cây, từ đó giảm số lần truy cập đĩa cần thiết trong các thao tác tìm kiếm, chèn và xóa.

Trong các hệ thống cơ sở dữ liệu và hệ thống tập tin, việc truy cập dữ liệu trên đĩa cứng thường chậm hơn rất nhiều so với truy cập dữ liệu trong bộ nhớ chính (RAM). Do đó, việc giảm thiểu số lần truy cập đĩa là yếu tố then chốt để nâng cao hiệu suất. B-Tree, với khả năng giảm chiều cao của cây và tối ưu hóa việc đọc/ghi dữ liệu theo khối (block-based), đã trở thành một trong những cấu trúc dữ liệu quan trọng nhất trong các ứng dụng lưu trữ và truy vấn dữ liệu lớn[3].

2.1.2 Lịch sử phát triển

B-Tree được phát minh bởi Rudolf Bayer và Edward M. McCreight vào năm 1971 khi làm việc tại Boeing Scientific Research Labs. Tên gọi "B-Tree" không được giải thích cụ thể bởi các tác giả, nhưng người ta thường cho rằng "B" có thể là viết tắt của "Balanced," "Broad," "Bushy," "Bayer," hoặc "Boeing."

2.2. Đặc điểm và cấu trúc của B-Tree

2.2.1. Tính cân bằng (Self-Balancing)

Tính cân bằng là đặc điểm quan trọng nhất của B-Tree, đảm bảo hiệu suất ổn định trong các thao tác tìm kiếm, chèn và xóa. Tất cả các nút lá (leaf nodes) của B-Tree đều nằm ở cùng một mức, nghĩa là đường dẫn tìm kiếm từ nút gốc đến bất kỳ nút lá nào đều có độ dài như nhau. Điều này khác biệt so với BST, nơi mà cây có thể bị lệch về một phía, dẫn đến thời gian tìm kiếm trong trường hợp xấu nhất là O(n) (với n là số lượng phần tử).

Tính tự cân bằng của B-Tree đảm bảo thời gian tìm kiếm, chèn, và xóa luôn ở mức **O(log n)**, mang lại hiệu suất ổn định và dễ dự đoán, không phụ thuộc vào thứ tự dữ liệu được chèn vào.

2.2.2. Bậc của cây (Order/Minimum Degree - t)

Bậc của cây, thường được ký hiệu là **t** (minimum degree), là một tham số quan trọng định nghĩa số lượng khóa tối thiểu và tối đa mà mỗi nút (ngoại trừ nút gốc) có thể chứa. Cụ thể:

- **Số lượng khóa tối thiểu:** Mỗi nút (trừ nút gốc) phải chứa ít nhất **t-1** khóa. Nút gốc có thể chứa ít hơn t-1 khóa, nhưng tối thiểu là 1 khóa.
- **Số lượng khóa tối đa:** Mỗi nút có thể chứa tối đa **2t-1** khóa.
- **Số lượng nút con:** Một nút với **k** khóa sẽ có **k+1** nút con.

Ví dụ, với B-Tree bậc t = 3:

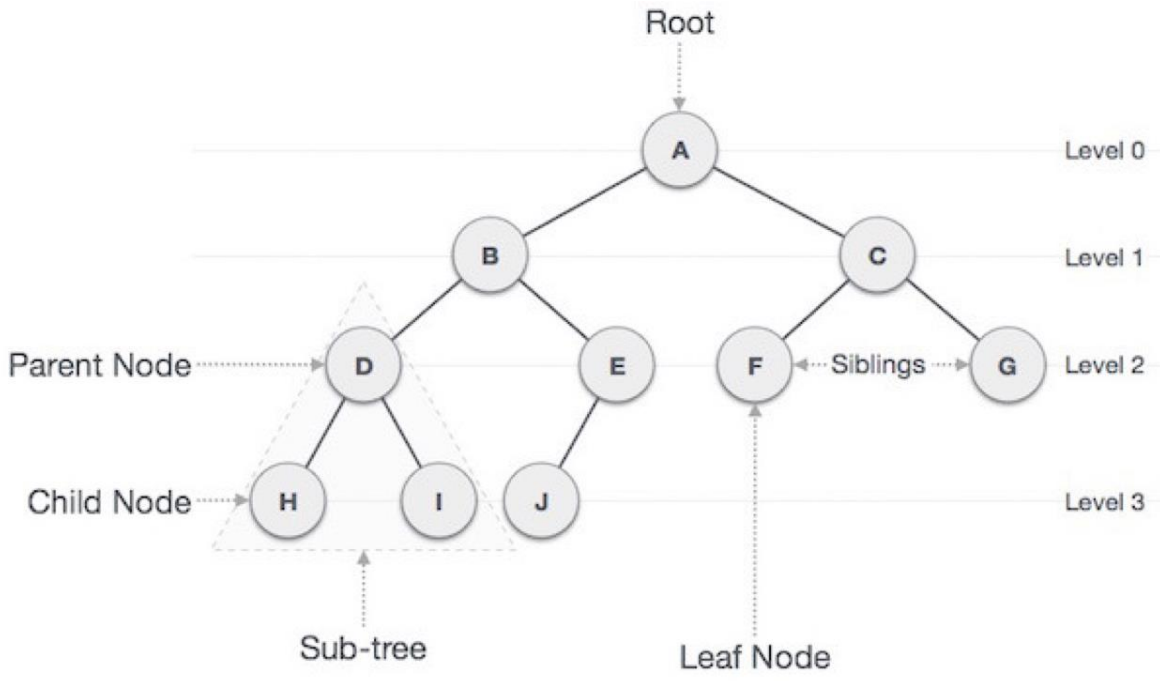
- Mỗi nút (trừ nút gốc) phải chứa ít nhất 2 khóa.
- Mỗi nút có thể chứa tối đa 5 khóa.
- Một nút có 4 khóa sẽ có 5 nút con.

Tham số **t** ảnh hưởng trực tiếp đến chiều cao của cây và hiệu suất của các thao tác. Giá trị **t** lớn hơn sẽ dẫn đến cây thấp hơn, giảm số lần truy cập đĩa, nhưng cũng có thể làm tăng thời gian tìm kiếm trong mỗi nút. Việc lựa chọn giá trị **t** phù hợp phụ thuộc vào đặc điểm cụ thể của hệ thống và dữ liệu.

2.2.3. Cấu trúc nút (Node Structure)

Mỗi nút trong B-Tree bao gồm các thành phần sau:

- **Khóa (Keys):** Các khóa được lưu trữ theo thứ tự tăng dần trong mỗi nút.
- **Con trỏ đến nút con (Child Pointers):** Mỗi khóa trong nút có một con trỏ liên kết đến một nút con. Con trỏ nằm bên trái của khóa trỏ đến nút con chứa các khóa nhỏ hơn khóa đó. Con trỏ nằm bên phải của khóa trỏ đến nút con chứa các khóa lớn hơn khóa đó. Nếu nút không có nút con(nút lá), con trỏ này sẽ mang giá trị null.
- **Con trỏ đến dữ liệu (Data Pointers - Tùy chọn):** Trong một số triển khai, B-Tree có thể lưu trữ con trỏ đến dữ liệu thực tế liên kết với mỗi khóa. Tuy nhiên, trong nhiều trường hợp (ví dụ: B+ Tree), dữ liệu thực tế chỉ được lưu trữ ở các nút lá, và các nút trong chỉ chứa khóa và con trỏ đến các nút con.
- **Số lượng khóa hiện tại (n):** Một số nguyên lưu trữ số lượng khóa hiện có trong nút.



Hình 3.1 - Minh họa cấu trúc cây B-tree

2.2.4. Phân bố khóa

Các khóa trong B-Tree được phân bố theo nguyên tắc sau:

- Các khóa trong mỗi nút được sắp xếp theo thứ tự tăng dần.
- Giữa hai khóa liên tiếp **k(i)** và **k(i+1)** trong một nút, tồn tại một con trỏ **p(i)** trỏ đến nút con.
- Tất cả các khóa trong cây con được trỏ bởi **p(i)** đều lớn hơn hoặc bằng **k(i)** và nhỏ hơn **k(i+1).**

2.3. Các thao tác trên B-Tree

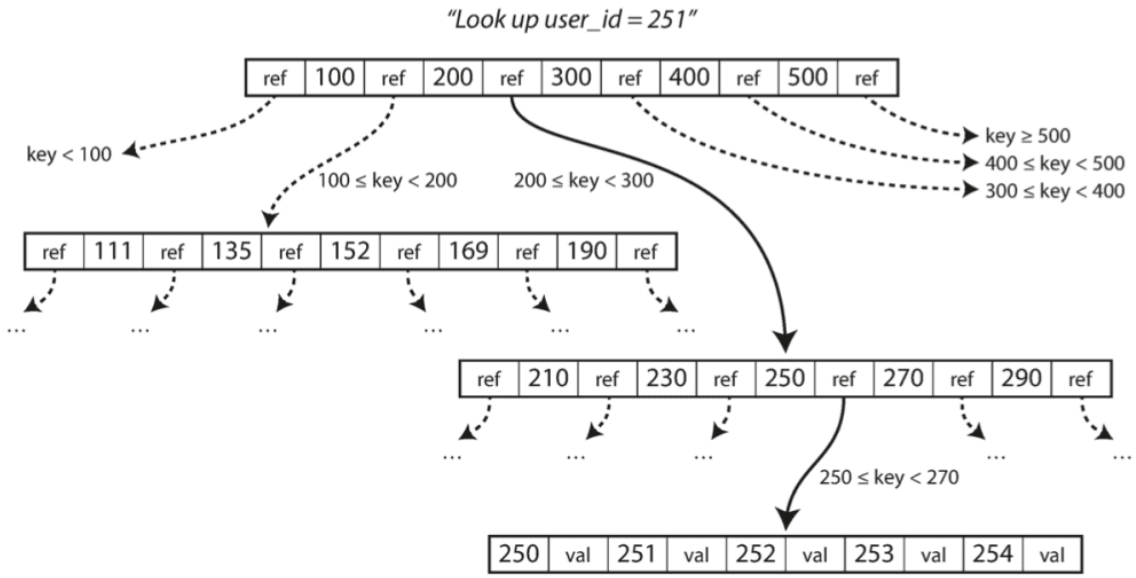
2.3.1. Tìm kiếm (Search)

Thao tác tìm kiếm trong B-Tree tương tự như tìm kiếm trong BST, nhưng được mở rộng để xử lý nhiều khóa trong mỗi nút. Quá trình tìm kiếm bắt đầu từ nút gốc:

- So sánh khóa cần tìm với các khóa trong nút hiện tại: Nếu tìm thấy khóa cần tìm trong nút, trả về vị trí của khóa (và dữ liệu liên quan nếu có).

- Di chuyển xuống nút con: Nếu không tìm thấy khóa, xác định vị trí con trỏ p(i) tương ứng dựa trên giá trị của khóa cần tìm (khóa cần tìm nằm giữa k(i) và k(i+1).
- Lặp lại: Di chuyển đến nút con được trỏ bởi p(i)và lặp lại các bước 1 và 2 cho đến khi tìm thấy khóa hoặc đến nút lá mà không tìm thấy.

Độ phức tạp: O(t log(t)n), trong đó t là bậc của cây và n là số nút.



Hình 3.2 – Truy vấn tìm kiếm trong B-Tree

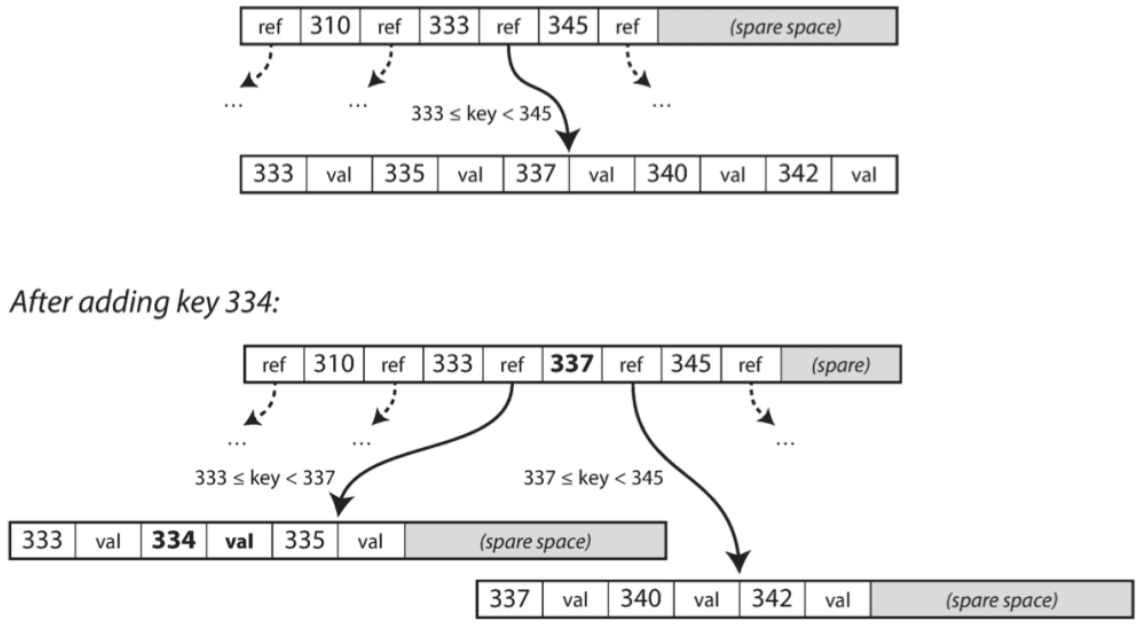
2.3.2. Chèn (Insertion)

Thao tác chèn một khóa mới vào B-Tree phức tạp hơn tìm kiếm vì cần đảm bảo tính cân bằng của cây. Quá trình chèn bao gồm các bước sau:

- Tìm vị trí chèn: Sử dụng thuật toán tìm kiếm để xác định nút lá mà khóa mới nên được chèn vào.
- Chèn khóa vào nút lá:
    - Nếu nút lá chưa đầy (số lượng khóa < 2t-1), chèn khóa mới vào đúng vị trí trong nút, duy trì thứ tự tăng dần.
    - Nếu nút lá đã đầy (số lượng khóa = 2t-1), thực hiện tách nút (split):

- Chia nút lá thành hai nút, mỗi nút chứa một nửa số khóa (t-1 khóa).
- Khóa ở giữa (median key) được đưa lên nút cha.
- Nếu nút cha cũng đầy, tiếp tục thực hiện tách nút lên trên cho đến khi gặp nút cha không đầy hoặc đến nút gốc.
- Nếu nút gốc bị tách, tạo một nút gốc mới chứa khóa giữa, và chiều cao của cây tăng lên 1.

Độ phức tạp: O(t log(t) n)



Hình 3.3 - Ghi giá trị vào B-Tree

2.3.3. Xóa (Deletion)

Thao tác xóa một khóa khỏi B-Tree cũng cần đảm bảo tính cân bằng của cây. Quá trình xóa phức tạp hơn chèn và tìm kiếm, bao gồm các bước sau:

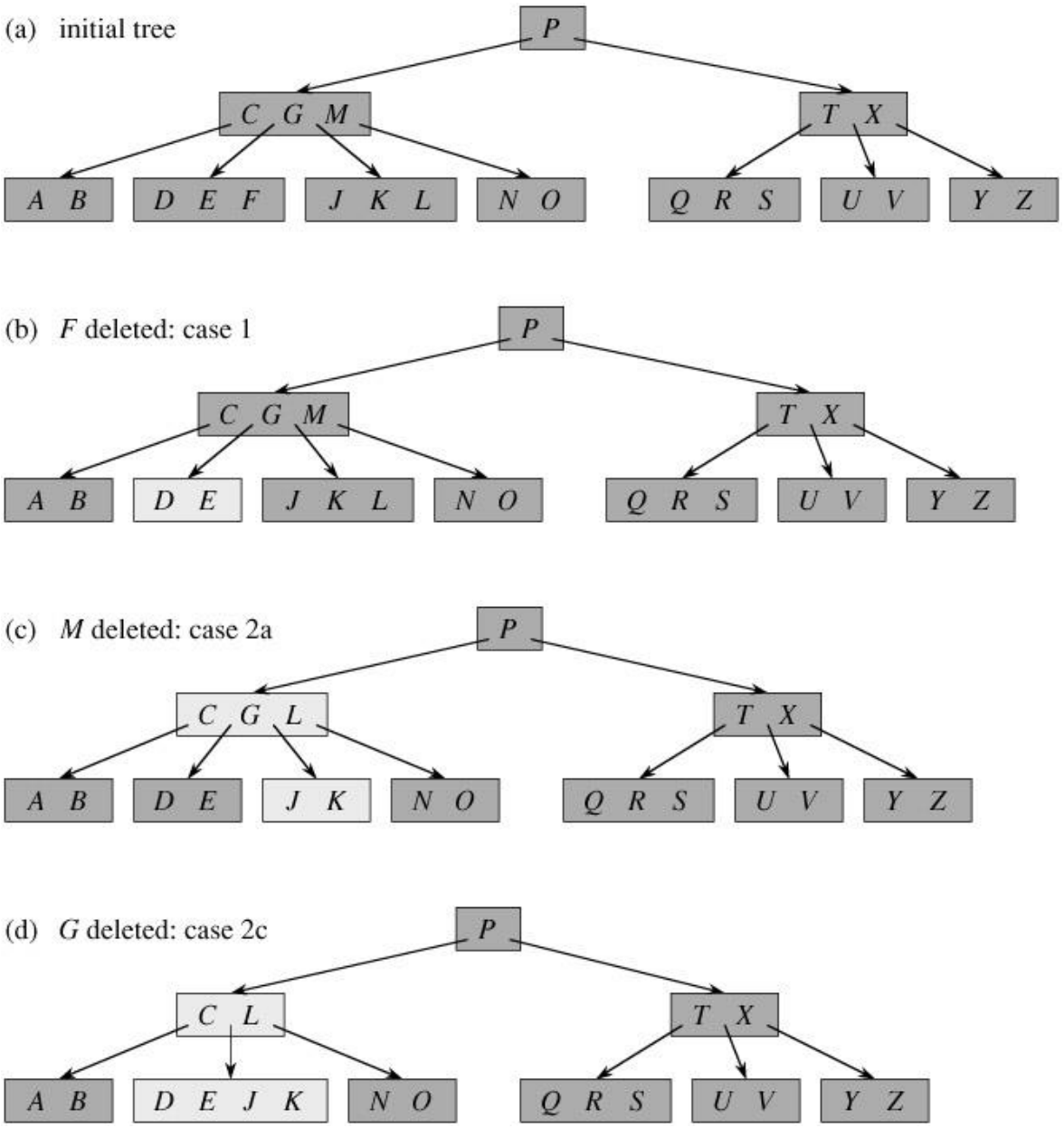
1. Tìm khóa cần xóa: Sử dụng thuật toán tìm kiếm để xác định nút chứa khóa cần xóa.
2. Xóa khóa:

- Trường hợp 1: Khóa cần xóa nằm ở nút lá:
    - Nếu nút lá có nhiều hơn t-1 khóa, xóa khóa trực tiếp khỏi nút.

- Nếu nút lá chỉ có t-1 khóa (khóa tối thiểu), xóa khóa khỏi nút. Sau đó, cần kiểm tra và điều chỉnh (rebalance) để đảm bảo tính cân bằng:
- Mượn khóa (Borrow/Rotation): Nếu một trong hai nút anh em liền kề (có cùng nút cha) có nhiều hơn t-1 khóa, mượn một khóa từ nút anh em đó và di chuyển khóa thích hợp từ nút cha xuống nút lá hiện tại. Đồng thời cập nhật lại các con trỏ cho phù hợp
- Hợp nhất (Merge): Nếu cả hai nút anh em liền kề chỉ có t-1 khóa, hợp nhất nút hiện tại với một trong hai nút anh em và một khóa từ nút cha. Nếu nút cha còn lại ít hơn t-1 khóa, tiếp tục điều chỉnh lên trên (lặp lại các bước mượn khóa hoặc hợp nhất nếu cần thiết). Trường hợp đặc biệt, nếu nút cha là nút gốc và nút cha chỉ còn lại 0 khóa sau khi hợp nhất, thì ta sẽ xóa nút gốc, khi đó, nút mới được hợp nhất sẽ trở thành nút gốc mới, và chiều cao của cây giảm đi 1.

- Trường hợp 2: Khóa cần xóa nằm ở nút trong:
  - Tìm khóa lớn nhất trong cây con bên trái (predecessor) hoặc khóa nhỏ nhất trong cây con bên phải (successor) của khóa cần xóa.
  - Thay thế khóa cần xóa bằng predecessor/successor.
  - Xóa predecessor/successor khỏi nút lá (sử dụng Trường hợp 1).

Độ phức tạp: O(t log(t)n)



Hình 3.3 – Phương thức xóa khỏi B-tree

2.4. Biến thể của B-Tree: B+ Tree

B+ Tree là một biến thể phổ biến của B-Tree, được tối ưu hóa hơn nữa cho việc lưu trữ dữ liệu trên đĩa và thường được sử dụng trong các hệ quản trị cơ sở dữ liệu và hệ thống tập tin. Điểm khác biệt chính giữa B+ Tree và B-Tree là:

- Lưu trữ dữ liệu: Trong B+ Tree, dữ liệu thực tế (hoặc con trỏ đến dữ liệu) chỉ được lưu trữ ở các nút lá. Các nút trong chỉ chứa các khóa đóng vai trò dẫn đường (routing keys) cho việc tìm kiếm.

- Liên kết giữa các nút lá: Các nút lá trong B+ Tree được liên kết với nhau bằng các con trỏ, tạo thành một danh sách liên kết. Điều này cho phép duyệt qua toàn bộ dữ liệu một cách hiệu quả theo thứ tự đã sắp xếp, hỗ trợ các truy vấn theo phạm vi (range queries) tốt hơn.

Cấu trúc của B+ Tree:

- Nút gốc (Root): Có thể là nút lá hoặc nút có ít nhất hai con trỏ.
- Nút trong (Internal Node): Chứa các khóa và con trỏ đến các nút con. Các khóa đóng vai trò là các giá trị phân tách (separator values) để dẫn đường tìm kiếm.
- Nút lá (Leaf Node): Chứa các khóa và con trỏ dữ liệu (hoặc dữ liệu thực tế). Các nút lá được liên kết với nhau tạo thành một danh sách liên kết.

Ưu điểm của B+ Tree:

- Tối ưu cho truy cập tuần tự: Nhờ danh sách liên kết ở các nút lá, B+ Tree cho phép truy cập tuần tự dữ liệu nhanh hơn, phù hợp với các truy vấn theo phạm vi.
- Hiệu quả lưu trữ cao hơn: Do các nút trong không lưu trữ dữ liệu, B+ Tree có thể lưu trữ nhiều khóa hơn trong mỗi nút, dẫn đến cây thấp hơn và giảm số lần truy cập đĩa.

2.5. Ứng dụng của B-Tree

B-Tree, với những ưu điểm vượt trội về hiệu suất truy cập và quản lý dữ liệu, đã trở thành một trong những cấu trúc dữ liệu nền tảng trong các hệ thống lưu trữ và truy vấn dữ liệu, đặc biệt là:

- Cơ sở dữ liệu (Database): B-Tree (và đặc biệt là B+ Tree) là cấu trúc dữ liệu chính được sử dụng để xây dựng chỉ mục (index) trong hầu hết các hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) như MySQL, PostgreSQL, Oracle, SQL Server. Chỉ mục B-Tree giúp tăng tốc độ truy vấn dữ liệu, đặc biệt là các truy vấn tìm kiếm theo phạm vi (range queries) và các truy vấn yêu cầu sắp xếp dữ liệu.

- Hệ thống tập tin (File Systems): Nhiều hệ thống tập tin hiện đại sử dụng B-Tree (hoặc biến thể của nó) để quản lý cấu trúc thư mục và định vị tập tin trên đĩa. Ví dụ, NTFS (New Technology File System) của Windows, HFS+ (Hierarchical File System Plus) của macOS, và ext4 (Fourth Extended File System) của Linux đều sử dụng B-Tree.
- Các ứng dụng lưu trữ và truy vấn dữ liệu khác: B-Tree cũng được sử dụng trong các ứng dụng khác yêu cầu lưu trữ và truy vấn dữ liệu hiệu quả, chẳng hạn như các công cụ tìm kiếm, các hệ thống lưu trữ key-value, và các hệ thống quản lý dữ liệu chuỗi thời gian (time-series data).

3. Hash-based Indexing

3.1. Giới thiệu về Hash-based Indexing

Hash-based Indexing là một kỹ thuật lập chỉ mục sử dụng hàm băm (hash function) để ánh xạ trực tiếp các giá trị khóa (key values) đến các vị trí lưu trữ dữ liệu tương ứng. Khác với các phương pháp lập chỉ mục dựa trên cây (như B-Tree) yêu cầu duyệt qua cấu trúc cây để tìm kiếm, Hash-based Indexing cho phép truy cập dữ liệu gần như tức thì chỉ với một phép tính băm duy nhất. Điều này làm cho nó trở thành lựa chọn lý tưởng cho các ứng dụng yêu cầu tốc độ truy xuất dữ liệu cực nhanh, đặc biệt là cho các truy vấn so khớp chính xác (exact-match queries).

3.1.1. Khái niệm cơ bản

a. Lập chỉ mục (Indexing)

Lập chỉ mục là quá trình tạo ra một cấu trúc dữ liệu phụ trợ (index) giúp tăng tốc độ tìm kiếm dữ liệu trong một tập dữ liệu lớn. Chỉ mục hoạt động tương tự như mục lục của một cuốn sách, cho phép chúng ta nhanh chóng tìm đến trang chứa thông tin cần thiết thay vì phải đọc toàn bộ cuốn sách.

b. Hàm băm (Hash Function)

Hàm băm là trái tim của Hash-based Indexing, là một hàm toán học nhận đầu vào là một giá trị khóa có độ dài bất kỳ và trả về đầu ra là một giá trị băm (hash value) có

độ dài cố định, thường là một số nguyên. Giá trị băm này được sử dụng như một chỉ số (index) để truy cập vào bảng băm (hash table).

Một hàm băm tốt cần đảm bảo các tính chất sau:

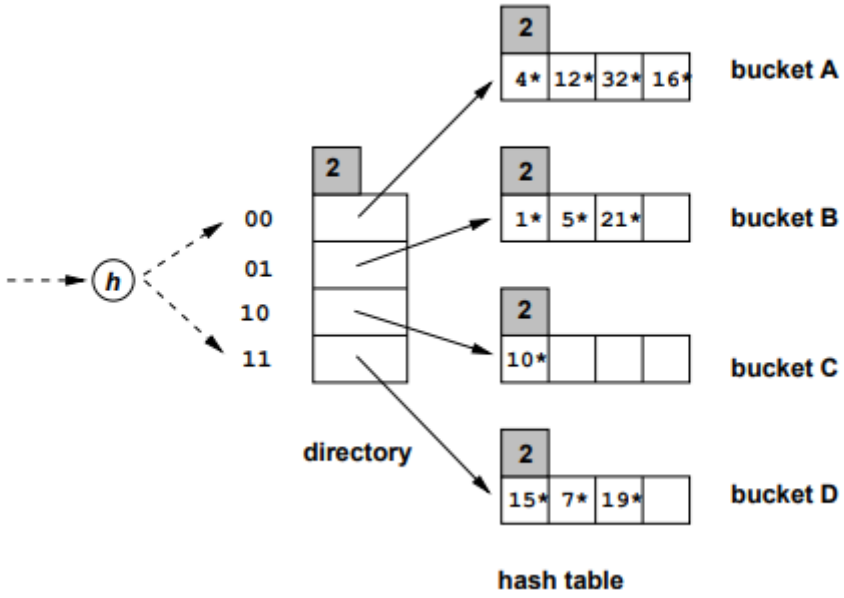
- **Tính đồng nhất (Uniformity):** Phân phối các giá trị băm một cách đồng đều trên toàn bộ không gian của bảng băm, giảm thiểu xác suất xảy ra xung đột.
- **Tính nhanh chóng (Fast Computation):** Có tốc độ tính toán nhanh để không làm ảnh hưởng đến hiệu suất tổng thể của hệ thống.
- **Tính nhạy cảm (Sensitivity):** Một thay đổi nhỏ trong giá trị khóa đầu vào cũng phải dẫn đến một thay đổi lớn trong giá trị băm đầu ra, giảm thiểu xung đột cho các khóa gần giống nhau.
- **Tính tất định (Deterministic):** Cùng một giá trị khóa đầu vào phải luôn tạo ra cùng một giá trị băm đầu ra.

Một số hàm băm phổ biến bao gồm:

- **Division Method:** h(key) = key mod m (với m là kích thước của bảng băm).
- **Multiplication Method:** h(key) = floor(m * (key * A mod 1)) (với A là một hằng số nằm trong khoảng 0 và 1, m là kích thước của bảng băm).
- **Universal Hashing:** Chọn ngẫu nhiên một hàm băm từ một tập hợp các hàm băm được thiết kế đặc biệt.

c. Bảng băm (Hash Table)

Bảng băm là cấu trúc dữ liệu chính được sử dụng trong Hash-based Indexing. Nó là một mảng các "thùng" (buckets) hoặc "khe" (slots), mỗi thùng có thể chứa một hoặc nhiều bản ghi dữ liệu. Vị trí của một bản ghi trong bảng băm được xác định bởi giá trị băm của khóa của bản ghi đó.



Hình 3.4 - Mô phỏng cấu trúc của Hash-based Indexing

3.2. Nguyên lý hoạt động

3.2.1. Ánh xạ khóa thành chỉ số

Quá trình lập chỉ mục bắt đầu bằng việc áp dụng hàm băm lên giá trị khóa của mỗi bản ghi dữ liệu. Hàm băm chuyển đổi khóa thành một giá trị băm, giá trị này sau đó được sử dụng để xác định vị trí (thùng) trong bảng băm mà bản ghi sẽ được lưu trữ.

3.2.2. Xử lý xung đột (Collision Handling)

Xung đột xảy ra khi hai hoặc nhiều khóa khác nhau được hàm băm ánh xạ đến cùng một thùng trong bảng băm. Đây là một vấn đề không thể tránh khỏi trong Hash-based Indexing do không gian khóa thường lớn hơn nhiều so với không gian của bảng băm. Do đó, các kỹ thuật xử lý xung đột hiệu quả là rất cần thiết để đảm bảo tính chính xác và hiệu suất của việc lập chỉ mục.

Có hai phương pháp chính để xử lý xung đột:

- **Chaining (Separate Chaining):** Mỗi thùng trong bảng băm được triển khai như một danh sách liên kết (linked list) hoặc một cấu trúc dữ liệu động khác. Khi xảy ra xung đột, bản ghi mới sẽ được thêm vào danh sách liên kết tương ứng với thùng đó.

    - **Ưu điểm:** Đơn giản, dễ triển khai.
    - **Nhược điểm:** Hiệu suất có thể giảm nếu danh sách liên kết quá dài, dẫn đến thời gian tìm kiếm tuyến tính trong danh sách.
- **Open Addressing:** Khi xảy ra xung đột, kỹ thuật này sẽ tìm kiếm một thùng trống khác trong bảng băm để lưu trữ bản ghi, thay vì sử dụng danh sách liên kết. Có một số phương pháp dò tìm (probing) khác nhau trong Open Addressing, bao gồm:
    - **Linear Probing:** Tìm kiếm tuần tự các thùng tiếp theo từ vị trí ban đầu cho đến khi tìm thấy thùng trống.
    - **Quadratic Probing:** Tìm kiếm thùng trống bằng cách tăng khoảng cách dò tìm theo cấp số nhân.
    - **Double Hashing:** Sử dụng một hàm băm thứ hai để xác định khoảng cách dò tìm.
    - **Ưu điểm:** Tiết kiệm không gian hơn so với Chaining vì không cần lưu trữ con trỏ cho danh sách liên kết.
    - **Nhược điểm:** Có thể xảy ra hiện tượng phân cụm (clustering), khi các bản ghi có xu hướng tập trung tại một số khu vực trong bảng băm, làm giảm hiệu suất

3.2.3. Truy xuất dữ liệu

Khi cần truy xuất dữ liệu dựa trên một giá trị khóa, hàm băm sẽ được áp dụng lên khóa đó để tính toán giá trị băm. Giá trị băm này sau đó được sử dụng để xác định thùng trong bảng băm mà bản ghi có thể được lưu trữ.

- **Trường hợp không có xung đột:** Nếu thùng chỉ chứa một bản ghi, bản ghi đó sẽ được trả về ngay lập tức.
- **Trường hợp có xung đột:**
    - **Chaining:** Duyệt qua danh sách liên kết trong thùng để tìm bản ghi có khóa trùng khớp.

- **Open Addressing:** Sử dụng cùng phương pháp dò tìm đã được sử dụng khi chèn để tìm kiếm bản ghi trong các thùng khác.

3.2.4. Chèn dữ liệu (Insertion)

Để chèn một bản ghi mới:

- Tính toán giá trị băm của khóa sử dụng hàm băm.
- Xác định thùng tương ứng trong bảng băm dựa trên giá trị băm.
- Nếu thùng trống, chèn bản ghi vào thùng.
- Nếu thùng đã có bản ghi (xung đột), áp dụng kỹ thuật xử lý xung đột (Chaining hoặc Open Addressing) để chèn bản ghi.

3.2.5. Xóa dữ liệu (Deletion)

Để xóa một bản ghi:

- Tính toán giá trị băm của khóa sử dụng hàm băm.
- Xác định thùng tương ứng trong bảng băm dựa trên giá trị băm.
- Tìm kiếm bản ghi trong thùng (sử dụng danh sách liên kết nếu áp dụng Chaining hoặc dò tìm nếu áp dụng Open Addressing).
- Nếu tìm thấy bản ghi, xóa bản ghi khỏi thùng.
    - **Chaining:** Xóa bản ghi khỏi danh sách liên kết.
    - **Open Addressing:** Cần đánh dấu thùng là đã bị xóa (deleted) thay vì để trống hoàn toàn để tránh ảnh hưởng đến quá trình tìm kiếm các bản ghi khác được chèn sau bản ghi bị xóa.

3.3. Các yếu tố ảnh hưởng đến hiệu suất

3.3.1. Hàm băm

Chất lượng của hàm băm là yếu tố quan trọng nhất ảnh hưởng đến hiệu suất của Hash-based Indexing. Một hàm băm tốt cần phân phối các giá trị băm một cách đồng đều trên toàn bộ không gian của bảng băm, giảm thiểu xác suất xảy ra xung đột.

3.3.2. Hệ số tải (Load Factor)

Hệ số tải được định nghĩa là tỷ lệ giữa số lượng bản ghi được lưu trữ trong bảng băm và kích thước của bảng băm (số lượng thùng). Hệ số tải cao có thể dẫn đến nhiều xung đột hơn, làm giảm hiệu suất. Thông thường, hệ số tải nên được giữ dưới 0,7.

3.3.3. Kỹ thuật xử lý xung đột

Lựa chọn kỹ thuật xử lý xung đột phù hợp có thể ảnh hưởng đáng kể đến hiệu suất. Chaining thường đơn giản và dễ triển khai hơn, nhưng Open Addressing có thể tiết kiệm không gian hơn. Việc lựa chọn kỹ thuật nào phụ thuộc vào đặc điểm cụ thể của ứng dụng và dữ liệu.

3.3.4. Kích thước bảng băm

Kích thước bảng băm cần đủ lớn để chứa tất cả các bản ghi mà không làm cho hệ số tải quá cao. Tuy nhiên, bảng băm quá lớn cũng có thể gây lãng phí không gian lưu trữ. Việc lựa chọn kích thước bảng băm phù hợp là một bài toán cân bằng giữa hiệu suất và sử dụng tài nguyên.

3.4. Ưu điểm và nhược điểm

3.4.1. Ưu điểm

- **Tốc độ truy xuất dữ liệu cực nhanh:** Thời gian truy cập trung bình là O(1) trong trường hợp lý tưởng (không có hoặc ít xung đột). Đây là ưu điểm lớn nhất của Hash-based Indexing.
- **Hiệu quả cho các truy vấn so khớp chính xác (exact-match queries):** Hash-based Indexing đặc biệt hiệu quả cho các truy vấn tìm kiếm bản ghi dựa trên giá trị khóa chính xác.

3.4.2. Nhược điểm

- **Không hỗ trợ các truy vấn theo phạm vi (range queries):** Do các khóa không được sắp xếp theo thứ tự trong bảng băm, Hash-based Indexing không thể được sử dụng để thực hiện các truy vấn tìm kiếm bản ghi trong một phạm vi giá trị khóa nhất định.

- **Hiệu suất phụ thuộc vào hàm băm và kỹ thuật xử lý xung đột:** Hiệu suất có thể giảm đáng kể nếu hàm băm không tốt hoặc xảy ra quá nhiều xung đột.
- **Khó khăn trong việc thay đổi kích thước bảng băm:** Khi số lượng bản ghi tăng lên, việc thay đổi kích thước bảng băm (rehashing) có thể tốn kém về mặt thời gian và tài nguyên.

3.5. Ứng dụng

Hash-based Indexing được sử dụng trong nhiều ứng dụng khác nhau, bao gồm:

- **Bảng ký hiệu trong trình biên dịch (Symbol Tables in Compilers):** Trình biên dịch sử dụng bảng băm để lưu trữ thông tin về các biến, hàm và các định danh khác trong mã nguồn, cho phép truy cập nhanh chóng thông tin về các định danh này trong quá trình biên dịch.
- **Bộ nhớ đệm (Caching):** Hash-based Indexing được sử dụng để triển khai các cơ chế bộ nhớ đệm, cho phép truy cập nhanh đến các dữ liệu thường xuyên được sử dụng.
- **Cơ sở dữ liệu:** Một số hệ quản trị cơ sở dữ liệu sử dụng Hash-based Indexing cho các cột không yêu cầu truy vấn theo phạm vi, chẳng hạn như các cột chứa mã định danh duy nhất (unique identifiers).
- **Kiểm tra chính tả (Spell Checkers):** Hash-based Indexing có thể được sử dụng để xây dựng từ điển kiểm tra chính tả, cho phép kiểm tra nhanh một từ có tồn tại trong từ điển hay không.

3. Elasticsearch

3.1. Tổng quan về Elasticsearch

3.1.1. Khái niệm và vai trò

Elasticsearch là một công cụ tìm kiếm và phân tích dữ liệu phân tán, mã nguồn mở, được xây dựng trên nền tảng của thư viện tìm kiếm Apache Lucene. Nổi bật với khả năng tìm kiếm toàn văn (full-text search) mạnh mẽ, linh hoạt, hỗ trợ tìm kiếm gần thời gian thực (near real-time), khả năng mở rộng theo chiều ngang (horizontally scalable) và phân tích dữ liệu lớn (big data analytics),

Elasticsearch đã trở thành một giải pháp hàng đầu cho nhiều ứng dụng, từ tìm kiếm nội dung website, ứng dụng, cho đến phân tích log, giám sát hệ thống và phân tích dữ liệu kinh doanh.

Elasticsearch cho phép bạn lưu trữ, tìm kiếm và phân tích một lượng lớn dữ liệu một cách nhanh chóng và hiệu quả. Dữ liệu trong Elasticsearch được lưu trữ dưới dạng tài liệu (documents) định dạng JSON, được tổ chức thành các chỉ mục (indices).

3.1.2. Kiến trúc phân tán

Elasticsearch được thiết kế để hoạt động như một hệ thống phân tán, có thể mở rộng dễ dàng bằng cách thêm các nút (nodes) vào một cụm (cluster). Dữ liệu được tự động phân chia và phân phối trên các nút khác nhau trong cụm, đảm bảo tính sẵn sàng cao (high availability) và khả năng chịu lỗi (fault tolerance).

3.1.3. RESTful API

Elasticsearch cung cấp một giao diện lập trình ứng dụng (API) dựa trên RESTful, cho phép tương tác với cụm Elasticsearch thông qua các yêu cầu HTTP đơn giản. Điều này giúp dễ dàng tích hợp Elasticsearch với các ứng dụng và hệ thống khác.

3.1.4. Lịch sử phát triển

Elasticsearch được phát triển bởi Shay Banon và được phát hành lần đầu tiên vào năm 2010. Ban đầu, nó được phát triển như một giải pháp tìm kiếm cho ứng dụng công thức nấu ăn của Banon. Sau đó, nó đã nhanh chóng trở nên phổ biến và được áp dụng rộng rãi trong nhiều lĩnh vực khác nhau. Hiện nay, Elasticsearch là một phần của bộ công cụ Elastic Stack (trước đây gọi là ELK Stack), bao gồm Elasticsearch, Logstash (thu thập và xử lý log), Kibana (trực quan hóa dữ liệu) và Beats (thu thập dữ liệu nhẹ).

3.2. Các thành phần chính của Elasticsearch

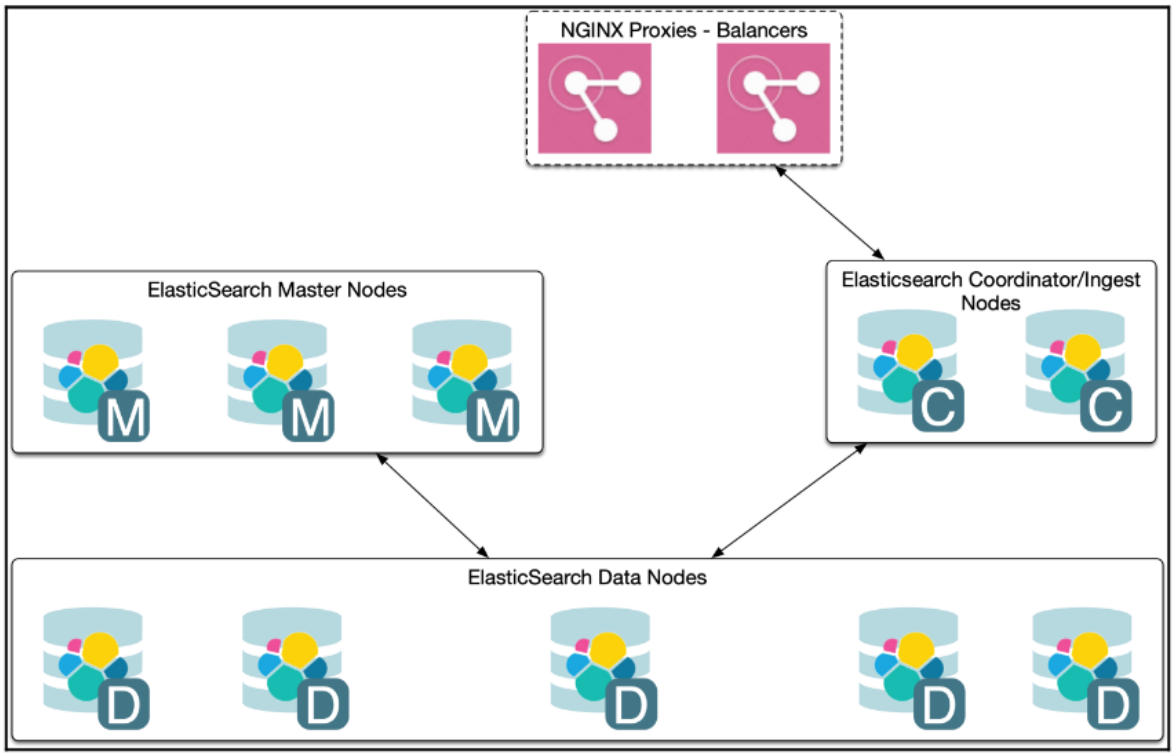
3.2.1. Cluster (Cụm)

Cụm (Cluster) trong Elasticsearch là một tập hợp các node (máy chủ) hoạt động cùng nhau để lưu trữ và quản lý dữ liệu, cung cấp khả năng tìm kiếm và phân tích. Mỗi cụm có một tên duy nhất, và các node trong cùng một cụm chia sẻ tên này, giúp chúng phối hợp và làm việc như một hệ thống thống nhất. Một cụm thường bao gồm một hoặc nhiều node, với một node chính (master node) điều phối hoạt động và các node dữ liệu (data nodes) lưu trữ và xử lý dữ liệu.

3.2.2. Node (Nút)

Một node là một instance (thể hiện) của Elasticsearch, là một server (máy chủ) riêng lẻ là một phần của cluster. Mỗi node tham gia vào các hoạt động lập chỉ mục và tìm kiếm của cluster, lưu trữ một phần dữ liệu của cluster. Có nhiều loại node khác nhau trong Elasticsearch, bao gồm:

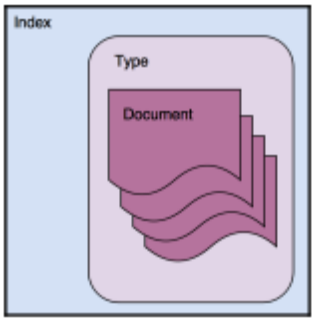
- **Master Node:** Chịu trách nhiệm quản lý các hoạt động cấp cluster như tạo, xóa index, thêm, xóa node, phân bổ shards cho các nodes. Trong một cluster, có một master node được bầu chọn (elected) từ các node đủ điều kiện làm master (master-eligible nodes).
- **Data Node:** Lưu trữ dữ liệu (các shards của các indices) và thực hiện các hoạt động liên quan đến dữ liệu như lập chỉ mục, tìm kiếm, tổng hợp (aggregations).
- **Ingest Node:** Tiền xử lý (pre-process) documents trước khi chúng được lập chỉ mục. Ingest Node có thể thực hiện các tác vụ như chuyển đổi cấu trúc dữ liệu, trích xuất thông tin, làm giàu dữ liệu (enrich data) trước khi nó được lưu trữ vào index.
- **Client Node/Coordinating Node:** Đóng vai trò như một bộ cân bằng tải (load balancer), định tuyến các yêu cầu tìm kiếm và lập chỉ mục đến các data nodes thích hợp. Nó không lưu trữ dữ liệu và không phải là master node. Client node tập hợp kết quả tìm kiếm từ các data nodes và trả về kết quả cuối cùng cho client.



Hình 3.5 – Mô phỏng các node trong Elasticsearch

3.2.3. Index (Chỉ mục)

Một index trong Elasticsearch tương tự như một database trong các hệ quản trị cơ sở dữ liệu quan hệ. Nó là một tập hợp các documents có các đặc điểm tương tự nhau. Mỗi index được định danh bằng một tên (viết thường) và được lưu trữ phân tán trên các nodes trong cluster.



Hình 3.6 – Mô phỏng index trong Elasticsearch

3.2.4. Document (Tài liệu)

Một document là đơn vị cơ bản của thông tin có thể được lập chỉ mục trong Elasticsearch. Nó được biểu diễn dưới dạng một đối tượng JSON, bao gồm một tập hợp các fields (trường), mỗi field có một tên và một giá trị tương ứng.

Ví dụ: Một document mô tả thông tin về một sản phẩm có thể có các fields như name, description, price, category.

3.2.5. Type (Loại)

Trước phiên bản 7.x, một index có thể chứa nhiều types, mỗi type có thể coi như một table trong database, giúp nhóm các documents có cấu trúc tương tự nhau trong cùng một index. Tuy nhiên, từ phiên bản 7.x trở đi, khái niệm type đã bị loại bỏ, và mỗi index chỉ nên chứa một type duy nhất, thường được đặt tên là _doc.

3.2.6. Field (Trường)

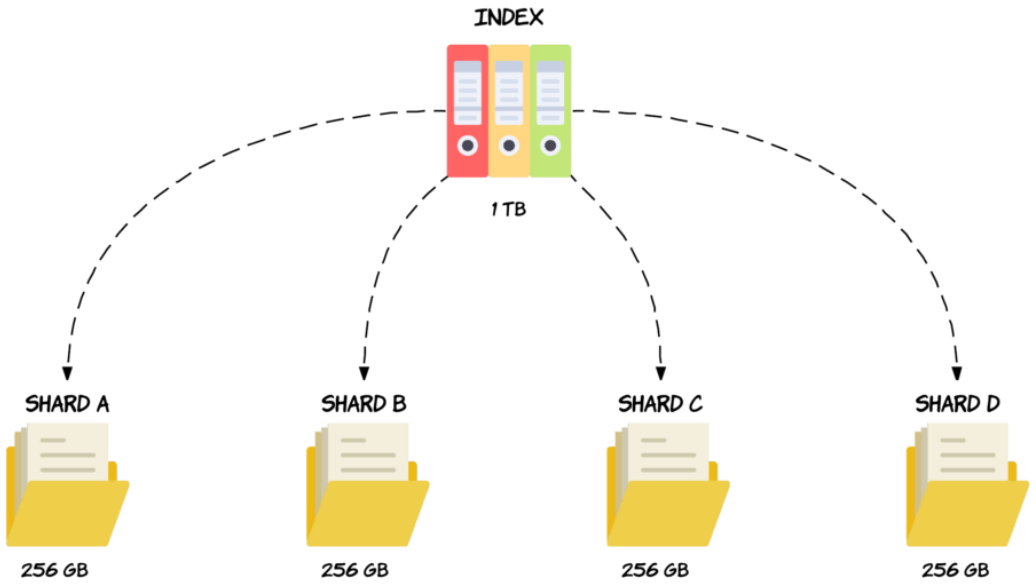
Một field là một cặp tên-giá trị (name-value pair) trong một document. Ví dụ, trong document về sản phẩm ở trên, name, description, price, và category là các fields. Mỗi field có một kiểu dữ liệu (data type) xác định, chẳng hạn như text, keyword, integer, date, boolean, geo_point, v.v.

3.2.7. Shard (Phân mảnh)

Để phân tán dữ liệu và tăng khả năng mở rộng, mỗi index trong Elasticsearch được chia thành nhiều phần nhỏ hơn gọi là shards. Mỗi shard là một index Lucene độc lập, có thể được lưu trữ trên bất kỳ node nào trong cluster. Khi một document được lập chỉ mục, nó sẽ được định tuyến đến một shard cụ thể dựa trên một hàm băm của ID document (mặc định) hoặc một routing value tùy chỉnh.

- **Primary Shard:** Mỗi shard có một bản chính (primary shard) chịu trách nhiệm cho việc lập chỉ mục dữ liệu. Số lượng primary shards được xác định khi tạo index và không thể thay đổi sau đó (trừ khi tạo lại index).
- **Replica Shard:** Mỗi primary shard có thể có một hoặc nhiều bản sao (replica shards) được lưu trữ trên các data nodes khác. Replica shards cung cấp khả năng chịu lỗi (nếu một node chứa primary shard bị lỗi, một replica shard sẽ được thăng cấp thành primary shard) và tăng hiệu suất đọc (các yêu cầu tìm

kiếm có thể được xử lý song song bởi cả primary và replica shards). Số lượng replica shards có thể thay đổi linh hoạt sau khi tạo index.



Hình 3.7 – Mô phỏng chia dữ liệu thành các phần nhỏ hơn có tên là Shard

3.2.8. Mapping (Ánh xạ)

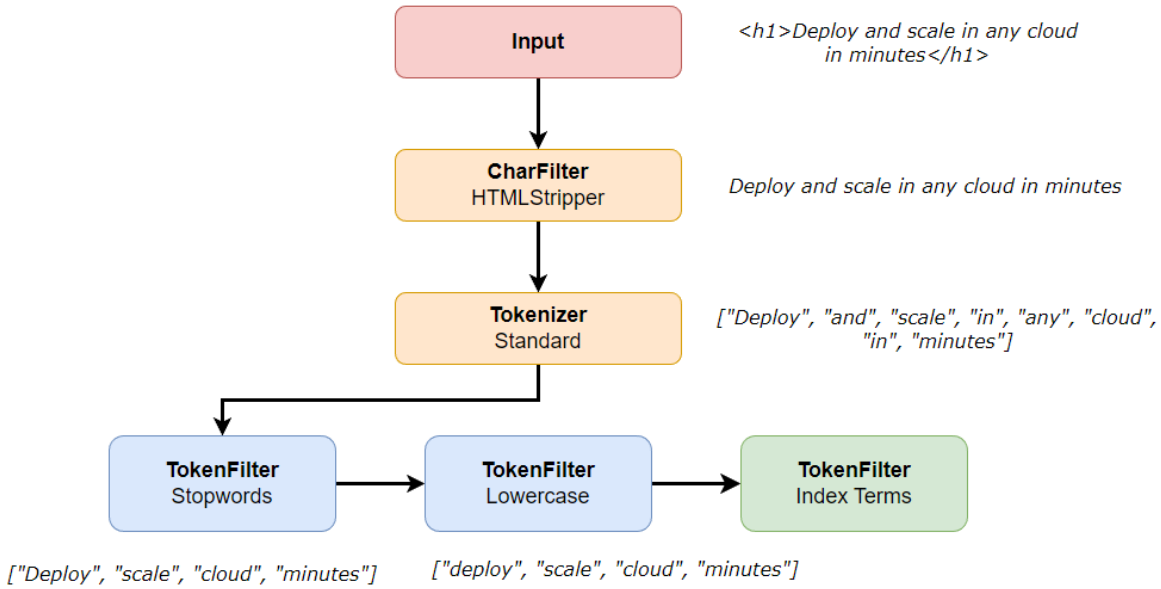
Mapping định nghĩa cấu trúc của các documents trong một index, bao gồm các fields và kiểu dữ liệu của chúng. Nó cũng định nghĩa cách các fields được phân tích (analyzed) và lập chỉ mục. Mapping có thể được định nghĩa tường minh (explicitly) hoặc được suy ra tự động (dynamically) bởi Elasticsearch khi các documents mới được lập chỉ mục.

Mapping đóng vai trò quan trọng trong việc tối ưu hóa tìm kiếm và lưu trữ. Ví dụ, một field được định nghĩa là text sẽ được phân tích thành các tokens để hỗ trợ tìm kiếm toàn văn, trong khi một field được định nghĩa là keyword sẽ được lưu trữ nguyên vẹn và chỉ hỗ trợ tìm kiếm chính xác.

3.2.9. Analyzer (Bộ phân tích)

Bộ phân tích chịu trách nhiệm xử lý văn bản (text fields) trong quá trình lập chỉ mục và tìm kiếm. Một analyzer bao gồm một bộ lọc ký tự (character filters), một tokenizer, và một hoặc nhiều bộ lọc token (token filters).

- **Character Filters:** Xử lý văn bản thô trước khi nó được đưa vào tokenizer. Ví dụ: loại bỏ các thẻ HTML, thay thế các ký tự đặc biệt.
- **Tokenizer:** Chia văn bản thành các tokens (thường là các từ đơn lẻ). Ví dụ, tokenizer tiêu chuẩn (standard tokenizer) sẽ chia văn bản dựa trên khoảng trắng và dấu câu.
- **Token Filters:** Xử lý các tokens sau khi chúng được tạo ra bởi tokenizer. Ví dụ: chuyển đổi chữ hoa thành chữ thường, loại bỏ các stop words (như "the", "a", "is"), stemming (rút gọn từ về dạng gốc), thêm từ đồng nghĩa (synonyms).



Hình ảnh 3.8 - Quy trình Analyzer

3.3. Quá trình lập chỉ mục (Indexing)

Khi một document được gửi đến Elasticsearch để lập chỉ mục, nó sẽ trải qua các bước sau:

- **Routing:** Elasticsearch xác định shard nào sẽ lưu trữ document dựa trên giá trị routing (mặc định là ID của document). Giá trị routing được băm để xác định primary shard tương ứng.
- **Ingestion (Nếu có Ingest Node):** Nếu cluster có cấu hình Ingest Node, document sẽ được gửi đến Ingest Node để tiền xử lý. Ingest Node có thể thực

hiện các tác vụ như chuyển đổi cấu trúc dữ liệu, trích xuất thông tin từ các fields, làm giàu dữ liệu trước khi nó được lưu trữ vào index.

- **Analysis:** Các text fields trong document được phân tích bởi analyzer tương ứng (được định nghĩa trong mapping) để tạo ra các terms (tokens) được sử dụng để xây dựng inverted index.
- **Indexing:** Document được lưu trữ trong primary shard tương ứng. Các terms được thêm vào inverted index, một cấu trúc dữ liệu ánh xạ các terms đến danh sách các documents chứa chúng.
- **Replication:** Document được sao chép đến các replica shards tương ứng để đảm bảo tính sẵn sàng cao và khả năng chịu lỗi.

3.4. Quá trình tìm kiếm (Searching)

Khi một yêu cầu tìm kiếm được gửi đến Elasticsearch, nó sẽ trải qua các bước sau:

- **Parsing:** Elasticsearch phân tích cú pháp (parse) yêu cầu tìm kiếm và chuyển đổi nó thành một truy vấn (query) dạng cây (abstract syntax tree - AST).
- **Routing:** Elasticsearch xác định các shards nào có thể chứa các documents khớp với truy vấn.
- **Execution:**
    - **Scatter Phase:** Yêu cầu tìm kiếm được gửi đến tất cả các shards (primary hoặc replica) liên quan.
    - **Gather Phase:** Mỗi shard thực hiện truy vấn cục bộ trên inverted index của nó và trả về một tập hợp các documents khớp, được sắp xếp theo điểm số (relevance score). Điểm số này được tính toán dựa trên các yếu tố như tần suất xuất hiện của term trong document (term frequency), tần suất nghịch đảo của document (inverse document frequency - IDF), và độ dài của document.

- **Merge Phase:** Node điều phối (coordinating node) nhận kết quả từ các shards, hợp nhất chúng lại, sắp xếp lại theo điểm số và trả về kết quả cuối cùng cho client.

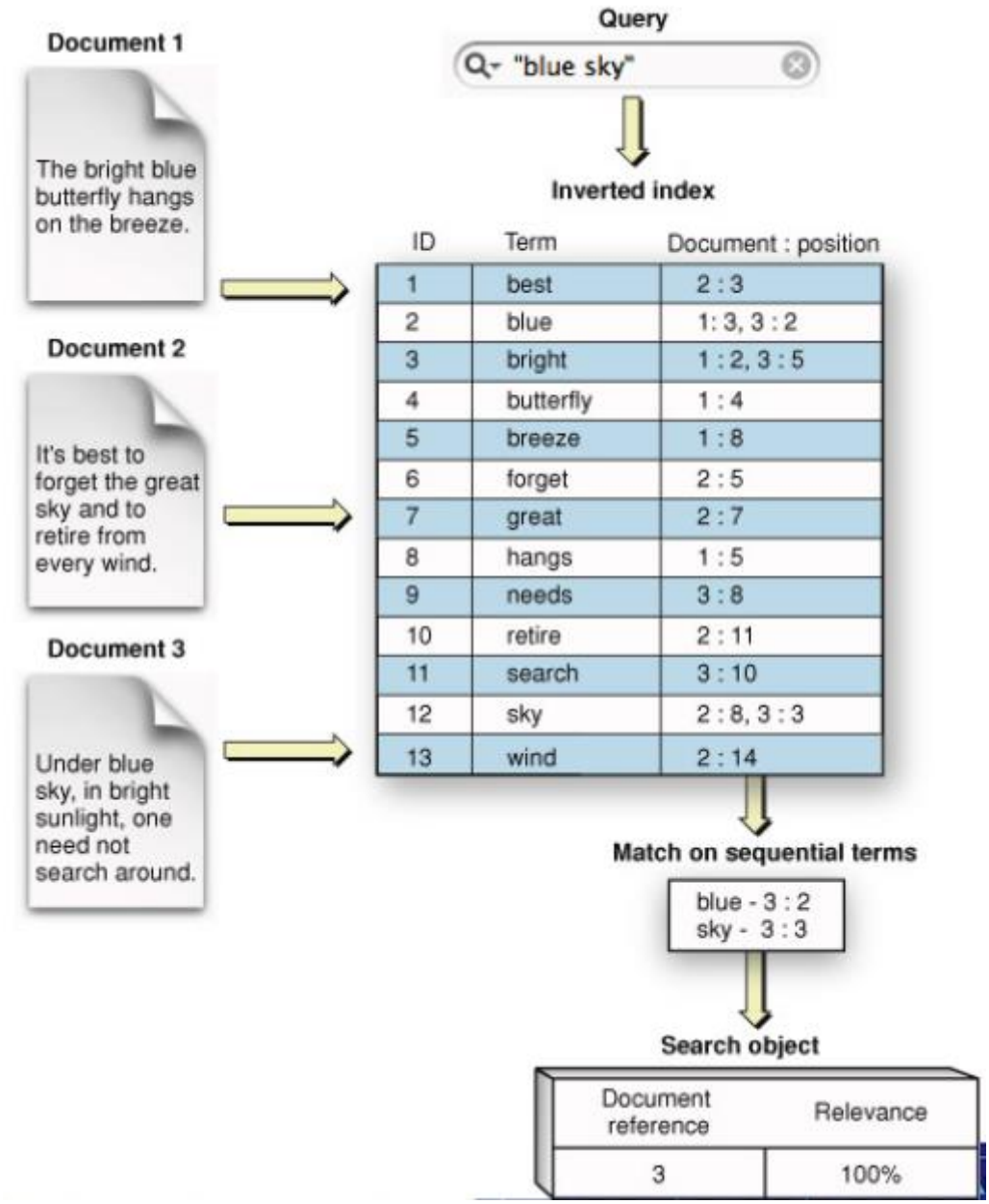
Inverted Index:

Trái tim của khả năng tìm kiếm nhanh chóng của Elasticsearch là inverted index. Inverted index là một cấu trúc dữ liệu ánh xạ các terms (tokens) đến danh sách các documents chứa chúng. Ví dụ:

Term	Document IDs
the	1, 2, 3, 5
lord	1, 3
rings	1, 2
tolkien	1, 5
hobbit	2

Bảng 3.1 - Cấu trúc dữ liệu ánh xạ các terms (tokens) đến danh sách các documents chứa chúng.

Khi một truy vấn tìm kiếm được thực hiện, Elasticsearch sẽ tìm kiếm các terms trong truy vấn trong inverted index và trả về các documents chứa các terms đó.



Hình 3.9 – Mô phỏng xử lý dữ liệu trong Elasticsearch

3.5. Ưu điểm của Elasticsearch

Tìm kiếm toàn văn mạnh mẽ: Dựa trên nền tảng Lucene, Elasticsearch cung cấp khả năng tìm kiếm toàn văn mạnh mẽ, hỗ trợ nhiều loại truy vấn phức tạp, bao gồm tìm kiếm mờ (fuzzy search), tìm kiếm gần đúng (proximity search), tìm kiếm ký tự đại diện (wildcard search), tìm kiếm biểu thức chính quy (regex search), v.v.

Tìm kiếm gần thời gian thực (Near Real-Time): Dữ liệu mới được lập chỉ mục gần như ngay lập tức có thể được tìm kiếm, thường chỉ với độ trễ khoảng 1 giây.

Khả năng mở rộng theo chiều ngang: Elasticsearch có thể dễ dàng mở rộng bằng cách thêm các nodes vào cluster, cho phép xử lý lượng dữ liệu lớn và lưu lượng truy cập cao.

Phân tích dữ liệu: Elasticsearch cung cấp các tính năng phân tích dữ liệu mạnh mẽ, bao gồm tổng hợp (aggregations) cho phép thực hiện các phép tính toán phức tạp trên dữ liệu, chẳng hạn như tính trung bình, tổng, độ lệch chuẩn, phân vị, v.v.

RESTful API: Giao diện RESTful API giúp dễ dàng tích hợp Elasticsearch với các ứng dụng và hệ thống khác.

Hệ sinh thái phong phú: Elasticsearch là một phần của Elastic Stack, cung cấp các công cụ mạnh mẽ cho việc thu thập, xử lý, trực quan hóa và phân tích dữ liệu.

3.6. Nhược điểm của Elasticsearch

Độ phức tạp: Việc cấu hình và quản trị một cluster Elasticsearch có thể phức tạp, đặc biệt là đối với các triển khai lớn.

Tài nguyên: Elasticsearch có thể tiêu tốn nhiều tài nguyên, đặc biệt là bộ nhớ.

ACID Properties: Elasticsearch không đảm bảo đầy đủ các thuộc tính ACID (Atomicity, Consistency, Isolation, Durability) như các hệ quản trị cơ sở dữ liệu quan hệ truyền thống. Nó cung cấp tính nhất quán cuối cùng (eventual consistency).

Security: Việc bảo mật Elasticsearch đòi hỏi cấu hình cẩn thận và sử dụng các tính năng bảo mật như mã hóa và xác thực.

3.7. Ứng dụng của Elasticsearch

Elasticsearch được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau, bao gồm:

- **Tìm kiếm ứng dụng (Application Search):** Cung cấp chức năng tìm kiếm cho các ứng dụng web, di động và doanh nghiệp.
- **Tìm kiếm website (Website Search):** Cải thiện trải nghiệm người dùng bằng cách cung cấp khả năng tìm kiếm nhanh chóng và chính xác trên các trang web.

- **Phân tích log (Log Analysis):** Thu thập, xử lý và phân tích log từ các hệ thống khác nhau để giám sát, khắc phục sự cố và phân tích bảo mật.
- **Giám sát hệ thống (System Monitoring):** Theo dõi hiệu suất và tình trạng của các hệ thống và ứng dụng trong thời gian thực.
- **Phân tích dữ liệu kinh doanh (Business Analytics):** Phân tích dữ liệu kinh doanh để tìm ra các xu hướng, mô hình và thông tin chi tiết có giá trị.
- **Security Information and Event Management (SIEM):** Thu thập và phân tích dữ liệu bảo mật từ nhiều nguồn khác nhau để phát hiện các mối đe dọa và lỗ hổng bảo mật.

Chương 4 Tối ưu hóa thuật toán

1. Lý thuyết tối ưu hóa

1.1. Khái niệm về tối ưu hóa

Tối ưu hóa là một lĩnh vực then chốt trong khoa học máy tính và toán học ứng dụng, tập trung vào việc tìm kiếm giải pháp tốt nhất (tối ưu) cho một vấn đề cụ thể. Quá trình này liên quan đến việc điều chỉnh các tham số, thay đổi chiến lược hoặc cải tiến cấu trúc của một hệ thống hoặc thuật toán để đạt được hiệu suất cao nhất, sử dụng tài nguyên hiệu quả nhất hoặc đạt được mục tiêu đề ra với chi phí thấp nhất, trong khi vẫn phải thỏa mãn các ràng buộc nhất định.[6]

Trong bối cảnh của đề tài này, tối ưu hóa đóng vai trò quan trọng trong việc nâng cao hiệu quả của các thuật toán tìm kiếm trên cơ sở dữ liệu lớn. Mục tiêu chính của việc tối ưu hóa trong trường hợp này là giảm thiểu thời gian truy vấn, tối ưu hóa việc sử dụng bộ nhớ, đồng thời nâng cao độ chính xác và độ tin cậy của kết quả tìm kiếm.

1.2. Các thành phần của tối ưu hóa

Một bài toán tối ưu hóa thường bao gồm ba thành phần chính:

a. Hàm mục tiêu (Objective Function):

Hàm mục tiêu, ký hiệu là f(x), là một hàm toán học mô tả mục tiêu cần tối ưu hóa. Biến đầu vào x đại diện cho các tham số hoặc biến quyết định (decision variables) của bài toán. Giá trị của hàm mục tiêu f(x) thể hiện chất lượng của một giải pháp cụ thể. Tùy thuộc vào bài toán cụ thể, chúng ta có thể muốn tối thiểu hóa (minimize) hoặc tối đa hóa (maximize) hàm mục tiêu.

Trong phạm vi của đề tài, hàm mục tiêu có thể được định nghĩa dựa trên các tiêu chí sau:

- Thời gian truy vấn (Latency): Mục tiêu là tối thiểu hóa thời gian trung bình cần thiết để thực hiện một truy vấn tìm kiếm trên cơ sở dữ liệu. Hàm mục tiêu có thể được xây dựng để phản ánh tổng thời gian truy vấn, thời gian truy vấn trung bình, hoặc thời gian truy vấn trong trường hợp xấu nhất.

- Sử dụng bộ nhớ (Memory Usage): Mục tiêu là tối thiểu hóa lượng bộ nhớ cần thiết để lưu trữ cấu trúc chỉ mục và thực hiện các thao tác tìm kiếm. Hàm mục tiêu có thể đo lường tổng dung lượng bộ nhớ sử dụng, hoặc mức sử dụng bộ nhớ cao nhất trong quá trình hoạt động.
- Độ chính xác của kết quả tìm kiếm (Precision and Recall): Mục tiêu là tối đa hóa độ chính xác của kết quả trả về, đảm bảo rằng các kết quả tìm kiếm liên quan nhất được trả về và hạn chế tối đa các kết quả không liên quan. Các độ đo như Precision (tỷ lệ kết quả tìm kiếm chính xác) và Recall (tỷ lệ kết quả liên quan được tìm thấy) có thể được sử dụng để xây dựng hàm mục tiêu.
- Kết hợp nhiều tiêu chí: Trong nhiều trường hợp, hàm mục tiêu có thể được xây dựng để tối ưu hóa đồng thời nhiều tiêu chí, ví dụ như tối thiểu hóa thời gian truy vấn đồng thời tối đa hóa độ chính xác. Các kỹ thuật như tổng trọng số (weighted sum) hoặc tối ưu hóa Pareto có thể được sử dụng để kết hợp các tiêu chí này thành một hàm mục tiêu duy nhất

b. Ràng buộc (Constraints):

Ràng buộc là các điều kiện, giới hạn, hoặc yêu cầu mà các giải pháp tối ưu phải thỏa mãn. Chúng định nghĩa không gian tìm kiếm (search space) hợp lệ cho bài toán tối ưu hóa. Ràng buộc có thể được biểu diễn dưới dạng các phương trình hoặc bất phương trình.

Các ràng buộc trong đề tài này có thể bao gồm:

- Giới hạn tài nguyên phần cứng: Giới hạn về CPU, RAM, dung lượng lưu trữ, băng thông mạng, v.v. Các giải pháp tối ưu phải đảm bảo rằng hệ thống hoạt động trong phạm vi tài nguyên phần cứng cho phép.
- Yêu cầu về bảo mật dữ liệu: Các ràng buộc về bảo mật dữ liệu, quyền truy cập, mã hóa dữ liệu, v.v. Các giải pháp tối ưu phải tuân thủ các quy định và chính sách bảo mật dữ liệu.
- Độ phức tạp thuật toán: Giới hạn về độ phức tạp tính toán của thuật toán, đảm bảo rằng thuật toán có thể thực thi trong thời gian chấp nhận được.

- Tính khả dụng (Availability) và độ tin cậy (Reliability): Hệ thống phải đảm bảo tính khả dụng cao và độ tin cậy, có khả năng phục hồi sau lỗi và tiếp tục hoạt động bình thường.

c. Phương pháp tối ưu hóa (Optimization Methods):

Phương pháp tối ưu hóa là các thuật toán, kỹ thuật được sử dụng để tìm kiếm giải pháp tối ưu cho bài toán. Lựa chọn phương pháp tối ưu hóa phù hợp phụ thuộc vào đặc điểm của hàm mục tiêu, các ràng buộc, và không gian tìm kiếm.

Các kỹ thuật tối ưu hóa:

- Tối ưu hóa tuyến tính (Linear Programming): Áp dụng cho các bài toán có hàm mục tiêu và các ràng buộc là tuyến tính.
- Tối ưu hóa phi tuyến (Nonlinear Programming): Áp dụng cho các bài toán có hàm mục tiêu hoặc các ràng buộc là phi tuyến.
- Tối ưu hóa nguyên (Integer Programming): Áp dụng cho các bài toán mà các biến quyết định phải nhận giá trị nguyên.
- Tối ưu hóa tổ hợp (Combinatorial Optimization): Áp dụng cho các bài toán mà không gian tìm kiếm là hữu hạn hoặc có thể đếm được.
- Thuật toán di truyền (Genetic Algorithms): Lấy cảm hứng từ quá trình tiến hóa tự nhiên, sử dụng các cơ chế chọn lọc, lai ghép và đột biến để tìm kiếm giải pháp tối ưu.
- Giải thuật tiến hóa (Evolutionary Algorithms): Bao gồm các thuật toán dựa trên nguyên lý tiến hóa như thuật toán di truyền, chiến lược tiến hóa (evolution strategies), lập trình gene (genetic programming),...
- Thuật toán đàn kiến (Ant Colony Optimization): Mô phỏng hành vi tìm kiếm thức ăn của đàn kiến để tìm kiếm giải pháp tối ưu.
- Tối ưu bầy đàn (Particle Swarm Optimization): Mô phỏng hành vi xã hội của các đàn chim hoặc đàn cá để tìm kiếm giải pháp tối ưu.
- Thuật toán mô phỏng luyện kim (Simulated Annealing): Lấy cảm hứng từ quá trình luyện kim, sử dụng kỹ thuật hạ nhiệt độ dần dần để tìm kiếm giải pháp tối ưu.

- Máy học (Machine Learning): Bao gồm các phương pháp học có giám sát (supervised learning), học không giám sát (unsupervised learning), và học tăng cường (reinforcement learning), có thể được sử dụng để xây dựng các mô hình dự đoán, tối ưu hóa các tham số, hoặc học các chiến lược tối ưu.
- Học sâu (Deep Learning): Một nhánh của máy học, sử dụng các mạng nơ-ron sâu (deep neural networks) với nhiều lớp ẩn để học các biểu diễn dữ liệu phức tạp và giải quyết các bài toán tối ưu hóa khó.

Trong đề tài này tập trung vào việc áp dụng các kỹ thuật máy học, đặc biệt là học sâu, để phát triển các mô hình dự đoán vị trí lưu trữ dữ liệu (Learned Hash Index). Các mô hình này được huấn luyện để ánh xạ các khóa dữ liệu đến các vị trí lưu trữ tối ưu trong bảng băm, giúp giảm thiểu xung đột và tăng tốc độ truy xuất.

1.3. Các kỹ thuật tối ưu hóa ứng dụng trong thuật toán tìm kiếm

a. Tối ưu hóa thuật toán (Algorithm Optimization):

Mục tiêu: Giảm độ phức tạp thời gian và không gian của thuật toán tìm kiếm, cải thiện hiệu suất thực thi của thuật toán.

Phương pháp:

- Cải tiến các bước xử lý dữ liệu: Phân tích và tối ưu hóa từng bước trong thuật toán tìm kiếm, loại bỏ các thao tác thừa, sử dụng các cấu trúc dữ liệu và giải thuật hiệu quả hơn.
- Sử dụng các cấu trúc dữ liệu hiệu quả: Lựa chọn các cấu trúc dữ liệu phù hợp cho từng bài toán cụ thể, ví dụ như sử dụng B-Tree cho các truy vấn theo phạm vi, sử dụng Hash Table cho các truy vấn so khớp chính xác, hoặc sử dụng các cấu trúc dữ liệu lai (hybrid data structures) kết hợp ưu điểm của nhiều cấu trúc dữ liệu khác nhau.
- Áp dụng các kỹ thuật chia để trị (divide and conquer): Chia nhỏ bài toán tìm kiếm thành các bài toán con độc lập, giải quyết từng bài toán con và kết hợp các kết quả lại để thu được kết quả cuối cùng.

- Tối ưu hóa mã nguồn (code optimization): Sử dụng các kỹ thuật lập trình hiệu quả, tối ưu hóa vòng lặp, giảm thiểu số lần gọi hàm, sử dụng các thư viện được tối ưu hóa, v.v.
- Áp dụng các kỹ thuật phân tán (distributed computing): Phân chia dữ liệu và các tác vụ tìm kiếm trên nhiều máy tính hoặc bộ xử lý để tăng tốc độ xử lý.

b. Tối ưu hóa bộ nhớ (Memory Optimization):

Mục tiêu: Giảm thiểu lượng bộ nhớ cần thiết để lưu trữ dữ liệu và thực hiện các thao tác tìm kiếm mà không làm giảm hiệu suất.

Phương pháp:

- Sử dụng các kỹ thuật nén dữ liệu (data compression): Nén dữ liệu trước khi lưu trữ để giảm dung lượng lưu trữ, đồng thời cần có cơ chế giải nén hiệu quả khi cần truy xuất dữ liệu. Các kỹ thuật nén phổ biến bao gồm Huffman coding, Lempel-Ziv, và các phương pháp nén chuyên dụng cho từng loại dữ liệu (ví dụ: nén ảnh, nén âm thanh, nén video).
- Tối ưu hóa cách lưu trữ chỉ mục: Sử dụng các cấu trúc chỉ mục nhỏ gọn, ví dụ như sử dụng các kỹ thuật mã hóa (encoding) để giảm kích thước của các khóa và con trỏ trong B-Tree hoặc sử dụng các cấu trúc chỉ mục dựa trên Bloom filter để kiểm tra nhanh sự tồn tại của một phần tử trong tập dữ liệu.
- Áp dụng các chiến lược quản lý bộ nhớ thông minh: Sử dụng các kỹ thuật như caching, swapping, và memory pooling để quản lý bộ nhớ hiệu quả, tái sử dụng bộ nhớ và giảm thiểu sự phân mảnh bộ nhớ.
- Sử dụng các cấu trúc dữ liệu tiết kiệm bộ nhớ: Ví dụ như sử dụng Succinct Data Structures, cho phép lưu trữ và truy vấn dữ liệu với lượng bộ nhớ gần với giới hạn lý thuyết thông tin (information-theoretic lower bound).

c. Tối ưu hóa phân tán (Distributed Optimization):

Mục tiêu: Tăng cường khả năng mở rộng (scalability) và hiệu suất (performance) của hệ thống tìm kiếm bằng cách phân tán dữ liệu và các tác vụ tìm kiếm trên nhiều máy tính hoặc bộ xử lý.

Phương pháp:

- Sử dụng các kiến trúc phân tán: Áp dụng các mô hình và kiến trúc phân tán như MapReduce, Hadoop, Spark để xử lý song song lượng dữ liệu lớn.
- Phân vùng dữ liệu (data partitioning): Chia nhỏ dữ liệu thành các phân vùng (partitions) và lưu trữ trên các nút khác nhau trong hệ thống phân tán. Các kỹ thuật phân vùng phổ biến bao gồm phân vùng theo phạm vi (range partitioning), phân vùng theo hàm băm (hash partitioning), và phân vùng nhất quán (consistent hashing).
- Sao chép dữ liệu (data replication): Sao chép dữ liệu trên nhiều nút để tăng tính sẵn sàng và khả năng chịu lỗi của hệ thống.
- Sử dụng các hệ thống tìm kiếm phân tán: Triển khai các hệ thống tìm kiếm phân tán như Elasticsearch, Solr, hoặc các hệ thống cơ sở dữ liệu NoSQL hỗ trợ phân tán dữ liệu và truy vấn song song.
- Tối ưu hóa giao tiếp mạng (network optimization): Giảm thiểu lưu lượng truyền thông giữa các nút trong hệ thống phân tán bằng cách sử dụng các kỹ thuật nén dữ liệu, truyền dữ liệu theo đợt (batching), và sử dụng các giao thức mạng hiệu quả.

1.4. Ứng dụng của lý thuyết tối ưu hóa trong nghiên cứu

Trong phạm vi của đề tài này, lý thuyết tối ưu hóa được áp dụng để phát triển và cải tiến các thuật toán tìm kiếm hiện có, đặc biệt là tập trung vào kỹ thuật **Learned Hash Index**. Ý tưởng chính của Learned Hash Index là sử dụng các mô hình học máy, cụ thể là mạng nơ-ron sâu (deep neural networks), để dự đoán vị trí lưu trữ của các khóa dữ liệu trong bảng băm.

Việc áp dụng lý thuyết tối ưu hóa trong nghiên cứu này được thể hiện qua các khía cạnh sau:

- Xây dựng hàm mục tiêu: Hàm mục tiêu được thiết kế để tối thiểu hóa xung đột trong bảng băm, đồng thời tối ưu hóa thời gian truy vấn và sử dụng bộ nhớ.
- Lựa chọn mô hình học máy: Các mô hình học máy, đặc biệt là mạng nơ-ron, được lựa chọn và huấn luyện để ánh xạ các khóa dữ liệu đến các vị trí lưu trữ

tối ưu. Quá trình huấn luyện tương đương với việc giải quyết một bài toán tối ưu hóa, trong đó các tham số của mô hình được điều chỉnh để tối thiểu hóa hàm mất mát (loss function), thường liên quan đến tỷ lệ xung đột và độ chính xác của dự đoán.

- Xử lý các ràng buộc: Các ràng buộc về tài nguyên phần cứng, yêu cầu bảo mật, và độ phức tạp thuật toán được xem xét và tích hợp vào quá trình thiết kế và triển khai giải pháp.
- Đánh giá và so sánh: Các phương pháp tối ưu hóa khác nhau được đánh giá và so sánh dựa trên các tiêu chí đã định nghĩa trong hàm mục tiêu, sử dụng các tập dữ liệu thử nghiệm và các kịch bản truy vấn khác nhau.

2. Giải pháp đề xuất: Learned Hash Index

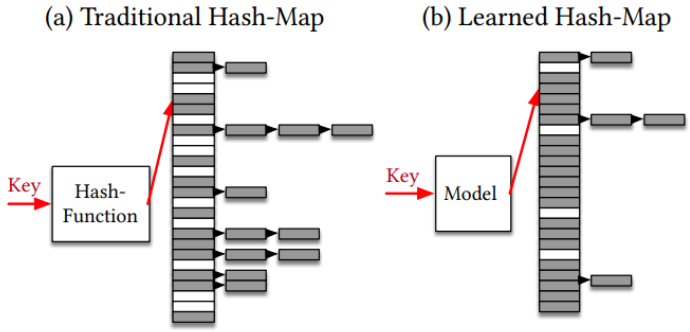
Trong bối cảnh dữ liệu lớn ngày càng phát triển, việc tối ưu hóa các thuật toán tìm kiếm trở nên vô cùng quan trọng để đảm bảo hiệu suất và khả năng mở rộng của hệ thống. Các phương pháp lập chỉ mục truyền thống như B-Tree và Hash Table, mặc dù đã được sử dụng rộng rãi và chứng minh hiệu quả trong nhiều ứng dụng, nhưng vẫn có những hạn chế nhất định khi đối mặt với sự gia tăng về khối lượng và độ phức tạp của dữ liệu.

Một trong những giải pháp tiên tiến hiện nay là **Learned Hash Index**, kết hợp giữa các kỹ thuật học máy và các cấu trúc dữ liệu truyền thống để cải thiện quá trình lập chỉ mục và truy xuất dữ liệu. Learned Hash Index sử dụng các mô hình học máy để học cách ánh xạ các khóa dữ liệu đến các vị trí lưu trữ trong bảng băm, thay vì sử dụng các hàm băm truyền thống.

2.1. Định nghĩa Learned Hash Index

Learned Hash Index là một phương pháp lập chỉ mục mới sử dụng các mô hình học máy, thường là các mạng nơ-ron (neural networks), để dự đoán vị trí lưu trữ của các khóa dữ liệu trong bảng băm (hash table). Thay vì sử dụng các hàm băm truyền thống (ví dụ: modulo, SHA) để ánh xạ khóa thành các vị trí lưu trữ một cách ngẫu nhiên, Learned Hash Index học cách ánh xạ dựa trên phân bố thực tế của dữ liệu.

Phương pháp này kết hợp khả năng khái quát hóa và học từ dữ liệu của các mô hình học máy với cấu trúc dữ liệu bảng băm để tối ưu hóa quá trình tìm kiếm, giảm thiểu xung đột và tăng tốc độ truy xuất dữ liệu[7].



Hình 4.1 - Traditional Hash-map và Learned Hash-map

2.2. Cách thức hoạt động của Learned Hash Index

a. Thu thập Dữ liệu Đào tạo:

Quy trình: Bước đầu tiên trong việc xây dựng Learned Hash Index là thu thập một tập hợp các khóa dữ liệu (key-value pairs) từ cơ sở dữ liệu lớn để sử dụng làm dữ liệu đào tạo (training data) cho mô hình học máy. Tập dữ liệu này cần đủ lớn và đại diện cho phân bố thực tế của dữ liệu trong hệ thống.

Mục tiêu: Đảm bảo rằng dữ liệu đào tạo phản ánh đúng phân bố và đặc điểm của dữ liệu thực tế, bao gồm các thông tin như phân bố tần suất xuất hiện của các khóa, mối quan hệ giữa các khóa, và các thuộc tính khác có thể ảnh hưởng đến vị trí lưu trữ tối ưu.

b. Tiền xử lý Dữ liệu:

Loại bỏ nhiễu: Loại bỏ các dữ liệu không cần thiết, dữ liệu bị lỗi hoặc nhiễu để đảm bảo chất lượng dữ liệu cao.

Chuẩn hóa văn bản (nếu có): Nếu sử dụng thêm thông tin từ các trường khác như text, thực hiện các bước tiền xử lý như loại bỏ stop words, stemming/lemmatization để chuẩn hóa từ.

Biểu diễn dữ liệu: Biểu diễn văn bản dưới dạng vector bằng các kỹ thuật như TF-IDF, word2vec, GloVe, hoặc sử dụng các embeddings từ các mô hình ngôn ngữ tiên tiến như BERT.

Chuẩn hóa dữ liệu (Normalization/Standardization): Chuẩn hóa các giá trị khóa để đảm bảo rằng chúng nằm trong một phạm vi nhất định, giúp mô hình học máy học hiệu quả hơn.

Mã hóa dữ liệu (Encoding): Nếu khóa dữ liệu không phải là số, cần mã hóa chúng thành dạng số để có thể sử dụng làm đầu vào cho mô hình học máy.

c. Huấn luyện Mô hình:

Lựa chọn mô hình: Lựa chọn mô hình học máy phù hợp để học ánh xạ từ khóa sang vị trí lưu trữ. Mặc dù các mô hình học sâu (deep learning) như Multi-Layer Perceptron

(MLP), Recurrent Neural Networks (RNN) hoặc Convolutional Neural Networks (CNN) có thể mang lại độ chính xác cao, nhưng cũng cần xem xét các mô hình đơn giản hơn như Linear Regression, Decision Tree để cân bằng giữa độ chính xác và độ phức tạp. Bắt đầu với các kiến trúc đơn giản như Multi-Layer Perceptron (MLP) để ánh xạ từ key word sang mã băm là một lựa chọn hợp lý.

Quá trình huấn luyện: Sử dụng dữ liệu đào tạo để tối ưu hóa các tham số của mô hình nhằm giảm thiểu sai số trong dự đoán vị trí lưu trữ. Quá trình huấn luyện thường sử dụng các thuật toán tối ưu hóa dựa trên gradient descent như Stochastic Gradient Descent (SGD), Adam, RMSprop.

- **Thiết kế hàm mất mát (Loss Function):**
  - Hàm mất mát (loss function) là một hàm số đo lường mức độ sai lệch giữa giá trị dự đoán của mô hình và giá trị thực tế. Việc lựa chọn hàm mất mát phù hợp là rất quan trọng để đảm bảo mô hình học được ánh xạ chính xác.
  - Lựa chọn hàm mất mát: Chọn hàm mất mát phù hợp tùy thuộc vào cách thức ánh xạ băm. Có hai hướng tiếp cận chính:
  - Coi giá trị băm là giá trị liên tục: Trong trường hợp này, ta có thể sử dụng các hàm mất mát như Mean Squared Error (MSE) hoặc Mean Absolute Error (MAE). MSE sẽ phạt nặng hơn các sai số lớn, trong khi MAE sẽ phạt đều tất cả các sai số. h(key) được coi là một giá trị liên tục và h(key) mod m (với m là kích thước của bảng băm) được sử dụng để ánh xạ vào các thùng. Phương pháp này yêu cầu h(key) phải xấp xỉ phân phối đều (uniform distribution) để giảm thiểu xung đột.
  - Coi giá trị băm là giá trị rời rạc (phân loại): Trong trường hợp này, ta có thể sử dụng hàm mất mát Cross-Entropy. Mỗi giá trị băm (bucket) được coi là một lớp (class) và mô hình sẽ dự đoán xác suất để một khóa rơi vào mỗi lớp. Phương pháp này coi bảng băm như một tập hợp các thùng (buckets) và h(key) là một hàm ánh xạ từ khóa vào các thùng. h(key) không cần phải xấp xỉ phân phối đều, nhưng cần phải nhất quán (consistent), tức là cùng một khóa phải luôn được ánh xạ vào cùng một thùng.

d. Áp dụng Mô hình:

Dự đoán vị trí lưu trữ: Khi một khóa dữ liệu mới cần được lưu trữ hoặc truy xuất, mô hình học máy sẽ dự đoán vị trí lưu trữ tối ưu trong bảng băm dựa trên các đặc trưng đã học được từ dữ liệu huấn luyện.

Giảm thiểu xung đột: Bằng cách dự đoán chính xác hơn vị trí lưu trữ, Learned Hash Index giảm thiểu khả năng xảy ra xung đột trong bảng băm so với các hàm băm truyền thống. Điều này dẫn đến việc tăng tốc độ truy xuất dữ liệu, đặc biệt là trong các trường hợp truy vấn so khớp chính xác.

e. Tích hợp và Áp dụng Mô hình vào Hệ thống Bảng Băm:

Triển khai mô hình: Đưa mô hình học máy (đã được huấn luyện) vào hệ thống tìm kiếm, đảm bảo rằng mỗi truy vấn tìm kiếm sẽ sử dụng mô hình để dự đoán vị trí lưu trữ của khóa dữ liệu. Việc triển khai có thể được thực hiện dưới dạng một dịch vụ (service) độc lập hoặc tích hợp trực tiếp vào hệ thống cơ sở dữ liệu.

Tối ưu hóa luồng dữ liệu: Thiết kế luồng dữ liệu để đảm bảo rằng dữ liệu được xử lý và lưu trữ một cách hiệu quả, từ quá trình dự đoán vị trí lưu trữ đến truy xuất dữ liệu. Cần xem xét các yếu tố như độ trễ (latency) của mô hình, băng thông mạng, và khả năng xử lý song song.

f. Kiểm thử và Đánh giá Hiệu suất:

Chạy các truy vấn tìm kiếm: Thực hiện các truy vấn tìm kiếm trên hệ thống có tích hợp Learned Hash Index và hệ thống sử dụng hàm băm truyền thống để so sánh hiệu suất.

Đo lường hiệu suất: Đo lường các chỉ số hiệu suất như thời gian truy vấn trung bình (average query time), độ trễ (latency), mức sử dụng bộ nhớ (memory usage), tỷ lệ xung đột (collision rate), và độ chính xác của kết quả tìm kiếm (precision, recall, F1-score).

So sánh kết quả: So sánh các chỉ số hiệu suất giữa hệ thống có Learned Hash Index và hệ thống truyền thống để xác định mức độ cải thiện.

g. Cập nhật và Điều chỉnh:

Liên tục cập nhật: Mô hình học máy cần được cập nhật định kỳ hoặc liên tục dựa trên dữ liệu mới và các truy vấn thực tế để duy trì và cải thiện hiệu suất theo thời gian.

Phản hồi từ hệ thống: Sử dụng phản hồi từ hệ thống (ví dụ: các truy vấn thường xuyên, các thay đổi trong phân bố dữ liệu) để tinh chỉnh và tối ưu hóa mô hình, đảm bảo rằng nó luôn thích ứng với sự thay đổi trong phân bố dữ liệu và yêu cầu của ứng dụng.

Các kỹ thuật học liên tục (Continual Learning/Lifelong Learning): Áp dụng các kỹ thuật Continual Learning để mô hình có thể học hỏi từ dữ liệu mới mà không quên đi kiến thức đã học được từ dữ liệu cũ (catastrophic forgetting).

2.3 Lợi ích của Learned Hash Index

a. Độ chính xác cao hơn:

Khả năng dự đoán: Mô hình học máy có khả năng học từ dữ liệu lịch sử và dự đoán chính xác hơn vị trí lưu trữ tối ưu cho các khóa dữ liệu, từ đó giảm thiểu số lần xảy ra xung đột so với các hàm băm truyền thống.

Giảm thiểu xung đột: Việc giảm thiểu xung đột trực tiếp dẫn đến việc tăng tốc độ truy xuất dữ liệu, đặc biệt là trong các truy vấn so khớp chính xác.

b. Khả năng mở rộng tốt:

Khả năng dự đoán: Mô hình học máy có khả năng học từ dữ liệu lịch sử và dự đoán chính xác hơn vị trí lưu trữ tối ưu cho các khóa dữ liệu, từ đó giảm thiểu số lần xảy ra xung đột so với các hàm băm truyền thống.

Giảm thiểu xung đột: Việc giảm thiểu xung đột trực tiếp dẫn đến việc tăng tốc độ truy xuất dữ liệu, đặc biệt là trong các truy vấn so khớp chính xác.

c. Tối ưu hóa tài nguyên:

Sử dụng bộ nhớ hiệu quả: Bằng cách giảm thiểu xung đột, Learned Hash Index gián tiếp giúp giảm kích thước trung bình của các chuỗi liên kết (chaining) hoặc số lần dò tìm (probing) trong bảng băm, từ đó giảm lượng bộ nhớ cần thiết để lưu trữ chỉ mục.

Giảm chi phí vận hành: Tối ưu hóa việc sử dụng tài nguyên phần cứng (CPU, bộ nhớ) dẫn đến giảm chi phí vận hành hệ thống, đặc biệt là trong các hệ thống dữ liệu lớn.

2.4 Các thách thức và giải pháp

a. Độ phức tạp của mô hình học máy:

- **Thách thức:** Việc xây dựng, huấn luyện và triển khai các mô hình học máy, đặc biệt là các mô hình học sâu, có thể phức tạp và đòi hỏi nhiều tài nguyên tính toán.
- **Giải pháp:**
  - Sử dụng các mô hình học máy đơn giản hơn: Bắt đầu với các mô hình đơn giản như Linear Regression, Decision Tree, hoặc Multi-Layer Perceptron (MLP) với ít lớp ẩn, và tăng dần độ phức tạp nếu cần thiết.
  - Tối ưu hóa quá trình huấn luyện: Sử dụng các kỹ thuật như giảm kích thước mô hình (model pruning), lượng tử hóa (quantization), và tri thức chưng cất (knowledge distillation) để giảm độ phức tạp của mô hình và tăng tốc độ huấn luyện.
  - Phân tán tính toán: Sử dụng các framework tính toán phân tán như TensorFlow, PyTorch (có hỗ trợ distributed training) để tăng tốc quá trình huấn luyện trên các tập dữ liệu lớn.

b. Dữ liệu không đồng đều:

- **Thách thức:** Phân bố dữ liệu không đồng đều (non-uniform data distribution) có thể làm giảm hiệu suất của mô hình học máy, đặc biệt là khi mô hình được huấn luyện trên một tập dữ liệu không đại diện cho toàn bộ phân bố dữ liệu.
- **Giải pháp:**
  - Sử dụng các kỹ thuật cân bằng dữ liệu (data balancing): Áp dụng các kỹ thuật như oversampling (nhân bản dữ liệu từ các lớp thiểu số), undersampling (loại bỏ bớt dữ liệu từ các lớp đa số), hoặc SMOTE (Synthetic Minority Over-sampling Technique) để cân bằng dữ liệu huấn luyện.
  - Phân cụm dữ liệu (data clustering): Phân cụm dữ liệu thành các nhóm có đặc điểm tương tự nhau và huấn luyện các mô hình riêng biệt cho từng nhóm.

- Sử dụng các mô hình chuyên biệt: Sử dụng các mô hình được thiết kế để xử lý dữ liệu không đồng đều, chẳng hạn như các mô hình dựa trên cây (tree-based models) hoặc các mô hình sử dụng hàm mất mát có trọng số (weighted loss function).

c. Cập nhật mô hình liên tục:

- **Thách thức:** Đảm bảo rằng mô hình học máy luôn được cập nhật với dữ liệu mới mà không làm gián đoạn dịch vụ hoặc gây ra sự không nhất quán trong quá trình truy vấn.
- **Giải pháp:**
  - Sử dụng các kỹ thuật học liên tục (continuous learning/lifelong learning): Áp dụng các kỹ thuật Continual Learning để mô hình có thể học hỏi từ dữ liệu mới mà không quên đi kiến thức đã học được từ dữ liệu cũ.
  - Triển khai mô hình theo hướng microservices: Đóng gói mô hình học máy thành các dịch vụ độc lập (microservices) để có thể cập nhật mô hình mà không ảnh hưởng đến các thành phần khác của hệ thống.
  - Sử dụng các kỹ thuật chuyển giao tri thức (transfer learning): Sử dụng kiến thức đã học được từ dữ liệu cũ để tăng tốc quá trình học trên dữ liệu mới.
  - A/B Testing: Thử nghiệm các phiên bản mô hình mới song song với mô hình hiện tại trên một phần nhỏ lưu lượng truy vấn trước khi triển khai chính thức.

Chương 5: Thực nghiệm

1. Bộ dữ liệu được sử dụng

1.1. Tiêu chí lựa chọn bộ dữ liệu

Bộ dữ liệu được lựa chọn dựa trên các tiêu chí sau:

- Tính đại diện (Representativeness): Bộ dữ liệu cần phản ánh được các đặc trưng của dữ liệu thực tế trong các ứng dụng mà các thuật toán tìm kiếm hướng tới, bao gồm kích thước, kiểu dữ liệu, phân bố của các giá trị, và tần suất xuất hiện của các khóa.
- Kích thước (Size): Bộ dữ liệu cần đủ lớn để đánh giá được hiệu suất và khả năng mở rộng của các thuật toán, đồng thời cũng cần có các bộ dữ liệu với kích thước khác nhau để đánh giá sự thay đổi hiệu suất theo kích thước dữ liệu.
- Độ phức tạp (Complexity): Bộ dữ liệu cần bao gồm các trường hợp dữ liệu đơn giản và phức tạp, với các phân bố dữ liệu khác nhau (ví dụ: đều, lệch chuẩn, tập trung) để đánh giá được khả năng xử lý của các thuật toán trong các tình huống khác nhau.
- Tính sẵn có (Availability): Ưu tiên sử dụng các bộ dữ liệu được công bố rộng rãi và được sử dụng phổ biến trong cộng đồng nghiên cứu để đảm bảo tính khách quan và khả năng so sánh kết quả.

1.2. Mô tả bộ dữ liệu Yelp Academic Dataset - Review

Bộ dữ liệu được sử dụng trong nghiên cứu này là **Yelp Academic Dataset**, cụ thể là tập dữ liệu **yelp_academic_dataset_review**. Đây là một tập dữ liệu lớn và phổ biến, thường được sử dụng trong các nghiên cứu về xử lý ngôn ngữ tự nhiên, hệ thống khuyến nghị, và khai phá dữ liệu.

- **Nguồn:** Yelp (https://www.yelp.com/dataset)
- **Mô tả:** Tập dữ liệu yelp_academic_dataset_review chứa thông tin về các đánh giá (reviews) của người dùng trên Yelp, bao gồm các trường chính sau:

- **review_id:** (String) ID duy nhất cho mỗi đánh giá.
- **user_id:** (String) ID duy nhất cho mỗi người dùng.
- **business_id:** (String) ID duy nhất cho mỗi doanh nghiệp.
- **stars:** (Integer) Điểm đánh giá (rating) từ 1 đến 5 sao.
- **text:** (String) Nội dung đánh giá bằng văn bản.
- **date:** (String) Ngày đánh giá được tạo.
- **useful:** (Integer) Số lượng người dùng thấy đánh giá này hữu ích.
- **funny:** (Integer) Số lượng người dùng thấy đánh giá này hài hước.
- **cool:** (Integer) Số lượng người dùng thấy đánh giá này thú vị.

- **Kích thước:** Tập dữ liệu yelp_academic_dataset_review thường có kích thước rất lớn, với **hàng triệu bản ghi đánh giá**. Phiên bản mới nhất (tính đến tháng 10 năm 2023) chứa **hơn 8 triệu đánh giá**.
- **Phân bố:**
  - Phân bố của trường stars thường lệch về phía các giá trị cao (nhiều đánh giá 4 và 5 sao hơn).
  - Độ dài của trường text (nội dung đánh giá) biến động lớn.
  - Tần suất của các từ trong trường text tuân theo phân bố Zipfian (một số từ xuất hiện rất thường xuyên, trong khi phần lớn các từ khác xuất hiện ít hơn).
- **Tính phù hợp:** Bộ dữ liệu yelp_academic_dataset_review rất phù hợp cho việc đánh giá các thuật toán tìm kiếm vì:
  - Kích thước lớn, cho phép đánh giá hiệu suất và khả năng mở rộng của các thuật toán.
  - Chứa cả dữ liệu cấu trúc (review_id, user_id, business_id, stars, date) và phi cấu trúc (text), cho phép thử nghiệm các loại truy vấn khác nhau.

- Phân bố dữ liệu phức tạp, phản ánh các thách thức trong các ứng dụng thực tế.

1.3. Xử lý và chuẩn bị dữ liệu

- **Trích xuất dữ liệu:**
  - Tải xuống tập dữ liệu yelp_academic_dataset_review từ trang web của Yelp.
  - Giải nén tập dữ liệu (thường được nén dưới định dạng .tgz hoặc .zip).
  - Sử dụng các thư viện như json (trong Python) để đọc và trích xuất dữ liệu từ các file JSON.
- **Làm sạch dữ liệu:**
  - Loại bỏ các bản ghi bị thiếu dữ liệu hoặc có dữ liệu không hợp lệ (ví dụ: giá trị stars nằm ngoài phạm vi 1-5).
  - Xử lý các ký tự đặc biệt, mã hóa ký tự (encoding) để đảm bảo tính nhất quán của dữ liệu.
- **Chọn trường sử dụng:**
  - **review_id:** Sử dụng làm khóa chính (primary key) cho các truy vấn tìm kiếm.
  - **business_id:** Sử dụng làm khóa ngoại (foreign key) và thử nghiệm các truy vấn theo business_id.
  - **text:** Sử dụng để thử nghiệm các truy vấn tìm kiếm toàn văn (full-text search).
  - **stars**: Sử dụng để thử nghiệm truy vấn theo phạm vi
- **Chia tách dữ liệu:**
  - Chia tập dữ liệu thành ba tập: tập huấn luyện (training set), tập validation (validation set), và tập kiểm thử (testing set) theo tỷ lệ 70:15:15.

- Tập huấn luyện được sử dụng để huấn luyện mô hình Learned Hash Index.
- Tập validation được sử dụng để tinh chỉnh các siêu tham số (hyperparameters) của mô hình.
- Tập kiểm thử được sử dụng để đánh giá hiệu suất cuối cùng của các thuật toán.

2. Triển khai các mô hình thuật toán B-tree, Hash-base Indexing, ElasticSearch

2.1. Môi trường triển khai

Phần cứng:

- Máy chủ: Sử dụng máy chủ có cấu hình:
  - CPU: 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz
  - RAM: 24 GB DDR4
  - Ổ cứng: 256 GB SSD NVMe
- Hệ điều hành: Windows

Phần mềm:

- Ngôn ngữ lập trình: Python 3.12
- Thư viện:
  - Pandas, NumPy: Xử lý dữ liệu
  - Scikit-learn: Triển khai các mô hình học máy
  - TensorFlow/PyTorch: Triển khai các mô hình học sâu (cho Learned Hash Index)
  - elasticsearch-py: Kết nối và thao tác với Elasticsearch
- Cơ sở dữ liệu:
  - Elasticsearch 7.17: Triển khai Elasticsearch
- Môi trường triển khai: Visual Studio Code

2.2 Thuật toán B-tree

2.2.1. Ngôn ngữ, thư viện và môi trường

- Ngôn ngữ lập trình: Python.
- Thư viện:
    - BTree: Thư viện tự định nghĩa, cung cấp các lớp BTree và BTreeNode để xây dựng và thao tác với cây B-Tree (code do người dùng cung cấp).
    - json: Thư viện tiêu chuẩn trong Python, dùng để đọc dữ liệu từ file JSON.
    - time: Thư viện tiêu chuẩn, dùng để đo thời gian thực thi.
    - pandas: Thư viện dùng để xử lý dữ liệu dạng bảng (DataFrame), hỗ trợ đọc dữ liệu theo chunk.
    - memory_profiler: Thư viện dùng để theo dõi mức sử dụng bộ nhớ.
- Môi trường: Visual studio code**.**

2.2.2. Các bước triển khai

a. Đọc và tiền xử lý dữ liệu:

- Dữ liệu được đọc từ file yelp_academic_dataset_review.json sử dụng hàm load_data. Hàm này đọc dữ liệu theo từng chunk (khối) với kích thước được chỉ định (mặc định là 10,000 bản ghi mỗi chunk), sử dụng yield để trả về một DataFrame mỗi lần đọc. Việc đọc dữ liệu theo chunk giúp xử lý hiệu quả các tập dữ liệu lớn, tránh tình trạng tràn bộ nhớ.
- Hàm load_data sử dụng thư viện json để phân tích cú pháp từng dòng trong file JSON và chuyển đổi thành các đối tượng Python (dictionary).
- Hàm load_data trả về các DataFrame (Pandas) chứa thông tin của các bản ghi reviews bao gồm: review_id, business_id, text, date, stars và các thông tin khác.
- Trường date được chuyển đổi sang định dạng ngày tháng %Y-%m-%d %H:%M:%S để nhất quán.

b. Xây dựng B-Tree:

- Hàm build_btree_index được sử dụng để xây dựng cây B-Tree.

- Tham số t (bậc của cây) được thiết lập mặc định là 3 (có thể điều chỉnh).
- Dữ liệu đầu vào data (dạng DataFrame hoặc danh sách các dictionary) được chuyển đổi thành danh sách các bản ghi records.
- Vòng lặp for duyệt qua từng bản ghi trong records. Với mỗi bản ghi, trường review_id được sử dụng làm khóa (key) và toàn bộ bản ghi được lưu trữ dưới dạng giá trị (value) trong cây B-Tree (thông qua hàm insert của lớp BTree).

c. Thực hiện truy vấn:

- Hàm search của lớp BTree được sử dụng để tìm kiếm các bản ghi dựa trên review_id.
- Hàm search trả về danh sách các giá trị (value) tương ứng với review_id được tìm thấy. Trong trường hợp này, mỗi giá trị là một dictionary chứa toàn bộ thông tin của bản ghi review.

2.2.3. Tham số quan trọng

t (bậc của cây): Tham số t quyết định số lượng khóa tối thiểu và tối đa trong mỗi nút của B-Tree (trừ nút gốc). t ảnh hưởng đến chiều cao của cây và hiệu suất của các thao tác chèn, xóa, tìm kiếm.

- **Lựa chọn t:** Giá trị t thường được chọn dựa trên kích thước bản ghi và kích thước trang đĩa (page size) của hệ thống lưu trữ. t lớn hơn có thể làm giảm chiều cao của cây, dẫn đến ít thao tác I/O đĩa hơn khi truy vấn, nhưng có thể làm tăng thời gian tìm kiếm trong mỗi nút.
- **Trong triển khai này:** t được mặc định là 3.

2.3. Thuật toán Hash-based Indexing

2.3.1. Ngôn ngữ, thư viện và môi trường

- **Ngôn ngữ lập trình:** Python.
- **Thư viện:**

- HashIndex: Thư viện tự định nghĩa, cung cấp lớp HashIndex để xây dựng và thao tác với Hash Index (code do người dùng cung cấp, giả định sử dụng dictionary để lưu trữ chỉ mục).
- json: Thư viện tiêu chuẩn trong Python, dùng để đọc dữ liệu từ file JSON.
- time: Thư viện tiêu chuẩn, dùng để đo thời gian thực thi.
- pandas: Thư viện dùng để xử lý dữ liệu dạng bảng (DataFrame), hỗ trợ đọc dữ liệu theo chunk.
- memory_profiler: Thư viện dùng để theo dõi mức sử dụng bộ nhớ.

- **Môi trường: Visual studio code**

2.3.2. Các bước triển khai

a. Đọc và tiền xử lý dữ liệu:

Tương tự như phần triển khai B-Tree, dữ liệu được đọc từ file yelp_academic_dataset_review.json sử dụng hàm load_data, chia thành các chunk và trả về dưới dạng các DataFrame.

b. Xây dựng Hash Index:

Hàm build_hash_index được sử dụng để xây dựng Hash Index.

Dữ liệu đầu vào data được chuyển đổi thành danh sách các bản ghi records.

Vòng lặp for duyệt qua từng bản ghi trong records. Với mỗi bản ghi, trường review_id được sử dụng làm khóa (key) và toàn bộ bản ghi được lưu trữ dưới dạng giá trị (value) trong Hash Index (thông qua hàm insert của lớp HashIndex).

c. Thực hiện truy vấn:

Hàm search của lớp HashIndex được sử dụng để tìm kiếm các bản ghi dựa trên review_id.

Hàm search trả về danh sách các giá trị (value) tương ứng với review_id được tìm thấy.

2.3.3. Tham số quan trọng

Hàm băm (Hash Function): Lựa chọn hàm băm phù hợp là rất quan trọng để đảm bảo hiệu suất của Hash Index. Hàm băm cần phân phối các khóa đều trên bảng băm để giảm thiểu xung đột.

Kích thước bảng băm (Hash Table Size): Kích thước bảng băm ảnh hưởng đến số lượng xung đột và hiệu suất truy vấn. Kích thước quá nhỏ dẫn đến nhiều xung đột, kích thước quá lớn gây lãng phí bộ nhớ.

Xử lý xung đột (Collision Handling): Cần có cơ chế xử lý xung đột hiệu quả khi hai hoặc nhiều khóa được băm đến cùng một vị trí trong bảng băm. Các kỹ thuật phổ biến bao gồm Separate Chaining (sử dụng danh sách liên kết) và Open Addressing (Linear Probing, Quadratic Probing, Double Hashing).

2.4. Thuật toán Elasticsearch

2.4.1. Ngôn ngữ, thư viện và môi trường

Ngôn ngữ lập trình: Python.

Thư viện:

- Elasticsearch, helpers: Thư viện client Elasticsearch cho Python, cung cấp các hàm để tương tác với Elasticsearch server (được sử dụng trong lớp ElasticSearchIndex).
- json: Thư viện tiêu chuẩn trong Python, dùng để đọc dữ liệu từ file JSON.
- time: Thư viện tiêu chuẩn, dùng để đo thời gian thực thi.
- pandas: Thư viện dùng để xử lý dữ liệu dạng bảng (DataFrame), hỗ trợ đọc dữ liệu theo chunk.
- memory_profiler: Thư viện dùng để theo dõi mức sử dụng bộ nhớ.

Môi trường: Yêu cầu cài đặt và cấu hình Elasticsearch server.

2.4.2. Các bước triển khai

a. Đọc và tiền xử lý dữ liệu:

Tương tự như phần triển khai B-Tree và Hash Index, dữ liệu được đọc từ file yelp_academic_dataset_review.json sử dụng hàm load_data, chia thành các chunk và trả về dưới dạng các DataFrame.

b. Xây dựng Elasticsearch Index:

Hàm build_elasticsearch_index được sử dụng để xây dựng Elasticsearch Index.

Đầu tiên, định nghĩa mapping cho index, chỉ định kiểu dữ liệu và analyzer cho các trường. Trường review_id được định nghĩa là keyword, trường text được định nghĩa là text với analyzer standard (hỗ trợ tìm kiếm toàn văn), và trường date được định nghĩa là date với format tương ứng.

Hàm create_index của lớp ElasticSearchIndex được gọi để tạo index reviews_index (có thể thay đổi tên index) với mapping đã định nghĩa.

Hàm index_documents của lớp ElasticSearchIndex được sử dụng để index các bản ghi từ DataFrame vào Elasticsearch. Hàm này sử dụng helpers.bulk để tối ưu hóa quá trình index, gửi nhiều bản ghi cùng lúc.

c. Thực hiện truy vấn:

Hàm search của lớp ElasticSearchIndex được sử dụng để thực hiện các truy vấn tìm kiếm trên Elasticsearch.

Hàm search nhận đầu vào là query_text và field (mặc định là text).

Hàm search sử dụng match query của Elasticsearch để tìm kiếm các bản ghi có trường field chứa query_text.

Kết quả trả về là danh sách các bản ghi khớp với truy vấn.

2.4.3. Tham số quan trọng

Mapping: Việc định nghĩa mapping chính xác cho các trường dữ liệu là rất quan trọng để đảm bảo hiệu suất và độ chính xác của các truy vấn tìm kiếm.

Analyzer: Lựa chọn analyzer phù hợp cho các trường text ảnh hưởng đến cách thức Elasticsearch xử lý và lập chỉ mục văn bản, từ đó ảnh hưởng đến kết quả tìm kiếm toàn văn.

Sharding và Replication: Elasticsearch cho phép chia nhỏ index thành các shard và sao chép các shard trên nhiều node để tăng khả năng mở rộng và độ tin cậy. Việc cấu hình sharding và replication cần được xem xét kỹ lưỡng dựa trên kích thước dữ liệu, yêu cầu về hiệu suất và khả năng chịu lỗi.

Refresh Interval: Elasticsearch lưu trữ dữ liệu mới trong bộ nhớ đệm (memory buffer) trước khi ghi xuống đĩa. Tham số refresh_interval quyết định tần suất làm mới (refresh) index, ảnh hưởng đến độ trễ giữa thời điểm dữ liệu được index và thời điểm dữ liệu có thể được tìm kiếm.

3. Triển khai mô hình Learned Hash Index

Phần này mô tả chi tiết các bước triển khai mô hình **Learned Hash Index** để lập chỉ mục và truy vấn trên bộ dữ liệu yelp_academic_dataset_review. Mô hình được xây dựng dựa trên ý tưởng ánh xạ các khóa (trong trường hợp này là review_id) thành các vị trí lưu trữ (bucket index) trong bảng băm thông qua một mô hình học máy, cụ thể là mạng nơ-ron MLP (Multi-Layer Perceptron).

3.1. Thu thập dữ liệu huấn luyện và tiền xử lý

a. Nguồn dữ liệu:

Dữ liệu huấn luyện được trích xuất từ file yelp_academic_dataset_review.json đã được làm sạch và tiền xử lý ở bước trước

b. Các trường sử dụng:

review_id: Trường review_id được sử dụng làm khóa chính (key) và là đầu vào chính cho mô hình..

c. Trích xuất review_id: Hàm load_review_ids (được sử dụng để đọc file JSON và trích xuất danh sách các review_id. Hàm này cũng có tham số limit để giới hạn số lượng review_id được trích xuất (mặc định limit=None nghĩa là trích xuất toàn bộ).

d. Tạo mapping cho review_id:

- Hàm create_id_mapping được sử dụng để tạo ánh xạ từ review_id sang index (dạng số nguyên). Việc này là cần thiết vì review_id trong file JSON là dạng chuỗi.
- Hàm trả về hai giá trị: unique_review_ids (danh sách các review_id duy nhất) và id_to_idx (một dictionary ánh xạ từ review_id sang index).
- Ví dụ: {'review_id_1': 0, 'review_id_2': 1, 'review_id_3': 2, ...}

e. Tạo targets:

- targets là danh sách các giá trị (dạng số nguyên) đại diện cho vị trí lưu trữ (bucket index) của mỗi review_id trong bảng băm.
- targets được tạo ra bằng cách áp dụng hàm bucket_function cho mỗi review_id duy nhất.
- Hàm bucket_function hiện tại đang thực hiện phép băm đơn giản: hash(key) % num_buckets.
- num_buckets là số lượng thùng (buckets) trong bảng băm, được thiết lập mặc định là 1024.

3.2. Phân chia dữ liệu

Sử dụng hàm **split_data** để phân chia dữ liệu thành tập huấn luyện và tập kiểm thử. Tỷ lệ phân chia được sử dụng là 80% cho tập huấn luyện và 20% cho tập kiểm thử.

3.3. Xây dựng mô hình

- **Kiến trúc:** Mô hình MLPHash là một mạng nơ-ron MLP (Multi-Layer Perceptron) với các tham số sau:
    - **Input layer:** Code hiện tại không xử lý đặc trưng bổ sung, nên số chiều input layer sẽ là 1 (tương ứng với review_id đã được mã hóa thành số nguyên).
    - **Hidden layers:** Số lượng hidden layers và số nơ-ron mỗi lớp chưa được định nghĩa rõ trong code. (Cần bổ sung code cho lớp MLPHash).
    - **Output layer:** Số lượng nơ-ron đầu ra bằng với num_buckets (số lượng thùng trong bảng băm), ở đây là 1024. Mô hình sử dụng

hàm softmax (thông qua nn.CrossEntropyLoss) để đưa ra phân phối xác suất, mỗi nơ-ron đại diện cho xác suất để review_id rơi vào thùng tương ứng.

- **Framework:** Mô hình được triển khai bằng PyTorch.

3.4. Huấn luyện mô hình

- **Dataset và DataLoader:**
    - ReviewHashDataset (định nghĩa trong dataset.py - cần được bổ sung code) là một lớp tùy chỉnh kế thừa từ torch.utils.data.Dataset, được sử dụng để tạo dataset cho việc huấn luyện và đánh giá mô hình.
    - DataLoader của PyTorch được sử dụng để tạo các batch dữ liệu từ dataset, hỗ trợ việc huấn luyện mô hình theo từng batch.
- **Thiết bị huấn luyện (Device):** Code tự động chọn thiết bị huấn luyện là "cuda" (GPU) nếu có, hoặc "cpu" nếu không.
- **Hàm mất mát (Loss Function):** nn.CrossEntropyLoss được sử dụng làm hàm mất mát, phù hợp cho bài toán phân loại nhiều lớp.
- **Trình tối ưu hóa (Optimizer):** optim.Adam được sử dụng với learning rate mặc định là 0.001.
- **Số epoch:** Mô hình được huấn luyện trong 5 epoch (có thể thay đổi).
- **Quá trình huấn luyện:**
    - Vòng lặp for duyệt qua từng epoch.
    - Trong mỗi epoch, vòng lặp for duyệt qua từng batch dữ liệu trong train_loader.
    - Với mỗi batch, thực hiện các bước:
        1. Chuyển dữ liệu sang thiết bị huấn luyện (CPU hoặc GPU).
        2. Xóa gradient tích lũy từ lần cập nhật trước (optimizer.zero_grad()).

3. Tính toán đầu ra của mô hình (outputs = model(x_batch)).
4. Tính toán độ lỗi (loss) giữa đầu ra dự đoán và nhãn thực tế (loss = criterion(outputs, y_batch)).
5. Tính toán gradient cho các tham số của mô hình (loss.backward()).
6. Cập nhật các tham số của mô hình (optimizer.step()).

- Tính toán và lưu trữ giá trị loss trung bình cho mỗi epoch.

3.5. Đánh giá mô hình

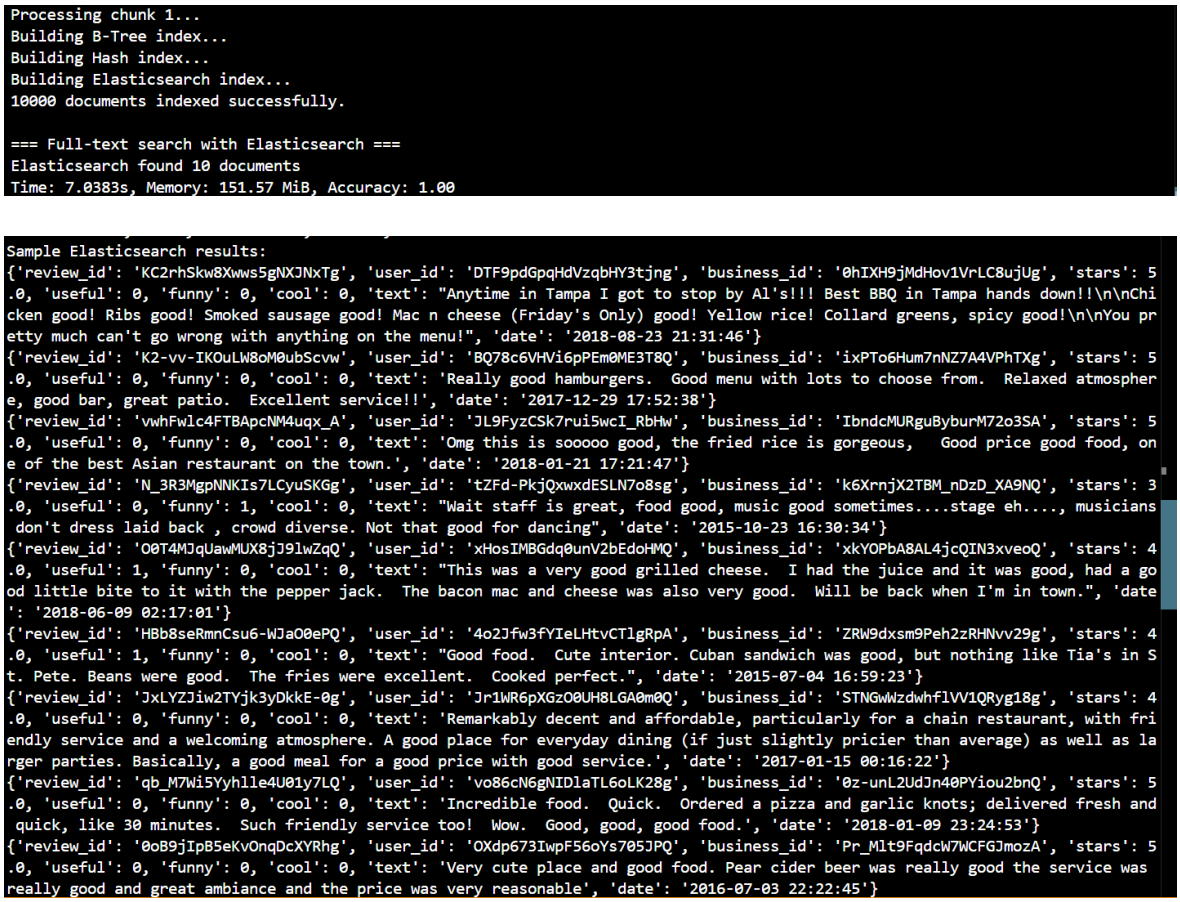
- **Quá trình đánh giá:**
  - Sau mỗi epoch huấn luyện, mô hình được đánh giá trên tập kiểm thử (test set) bằng cách sử dụng DataLoader.
  - Độ chính xác (accuracy) trên tập kiểm thử được tính toán và in ra.
- **Độ chính xác (Accuracy):** Code tính toán độ chính xác (accuracy) của mô hình trên tập kiểm thử bằng cách đếm số lượng dự đoán đúng (predicted == y_batch) chia cho tổng số lượng mẫu trong tập kiểm thử.

Chương 6: Kết quả và đánh giá

1. Kết quả của các thuật toán sau khi thực nghiệm

1.1 Mô hình thuật toán B-tree, Hash-base Indexing, ElasticSearch

Lần 1:



```
Processing chunk 1...
Building B-Tree index...
Building Hash index...
Building Elasticsearch index...
10000 documents indexed successfully.

=== Full-text search with Elasticsearch ===
Elasticsearch found 10 documents
Time: 7.0383s, Memory: 151.57 MiB, Accuracy: 1.00

Sample Elasticsearch results:
{'review_id': 'KC2rhSkw8Xwws5gNXJNxTg', 'user_id': 'DTF9pdGpqHdVzqbHY3tjng', 'business_id': '0hIXH9jMdHov1VrLC8ujUg', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "Anytime in Tampa I got to stop by Al's!!! Best BBQ in Tampa hands down!!\n\nChi
cken good! Ribs good! Smoked sausage good! Mac n cheese (Friday's Only) good! Yellow rice! Collard greens, spicy good!\n\nYou pr
etty much can't go wrong with anything on the menu!", 'date': '2018-08-23 21:31:46'}
{'review_id': 'K2-vv-IKOuLW8oM0ubScvw', 'user_id': 'BQ78c6VHVi6pPEm0ME3T8Q', 'business_id': 'ixPTo6Hum7nNZ7A4VPhTXg', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Really good hamburgers.  Good menu with lots to choose from.  Relaxed atmospher
e, good bar, great patio.  Excellent service!!', 'date': '2017-12-29 17:52:38'}
{'review_id': 'vwhFwlc4FTBApcNM4uqx_A', 'user_id': 'JL9FyzCSk7rui5wcI_RbHw', 'business_id': 'IbndcMURguByburM72o3SA', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Omg this is sooooo good, the fried rice is gorgeous,  Good price good food, on
e of the best Asian restaurant on the town.', 'date': '2018-01-21 17:21:47'}
{'review_id': 'N_3R3MgpNNKIs7LCyuSKGg', 'user_id': 'tZFd-PkjQxwxdESLN7o8sg', 'business_id': 'k6XrnjX2TBM_nDzD_XA9NQ', 'stars': 3
.0, 'useful': 0, 'funny': 1, 'cool': 0, 'text': "Wait staff is great, food good, music good sometimes....stage eh...., musicians
 don't dress laid back , crowd diverse. Not that good for dancing", 'date': '2015-10-23 16:30:34'}
{'review_id': 'O0T4MJqUawMUX8jJ9lwZqQ', 'user_id': 'xHosIMBGdq0unV2bEdoHMQ', 'business_id': 'xkYOPbA8AL4jcQIN3xveoQ', 'stars': 4
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': "This was a very good grilled cheese.  I had the juice and it was good, had a go
od little bite to it with the pepper jack.  The bacon mac and cheese was also very good.  Will be back when I'm in town.", 'date
': '2018-06-09 02:17:01'}
{'review_id': 'HBb8seRmnCsu6-WJaO0ePQ', 'user_id': '4o2Jfw3fYIeLHtvCTlgRpA', 'business_id': 'ZRW9dxsm9Peh2zRHNvv29g', 'stars': 4
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': "Good food.  Cute interior. Cuban sandwich was good, but nothing like Tia's in S
t. Pete. Beans were good.  The fries were excellent.  Cooked perfect.", 'date': '2015-07-04 16:59:23'}
{'review_id': 'JxLYZJiw2TYjk3yDkkE-0g', 'user_id': 'Jr1WR6pXGzO0UH8LGA0m0Q', 'business_id': 'STNGWWzdwhflVV1QRyg18g', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Remarkably decent and affordable, particularly for a chain restaurant, with fri
endly service and a welcoming atmosphere. A good place for everyday dining (if just slightly pricier than average) as well as la
rger parties. Basically, a good meal for a good price with good service.', 'date': '2017-01-15 00:16:22'}
{'review_id': 'qb_M7Wi5Yyhlle4U01y7LQ', 'user_id': 'vo86cN6gNIDlaTL6oLK28g', 'business_id': '0z-unL2UdJn40PYiou2bnQ', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Incredible food.  Quick.  Ordered a pizza and garlic knots; delivered fresh and
 quick, like 30 minutes.  Such friendly service too!  Wow.  Good, good, good food.', 'date': '2018-01-09 23:24:53'}
{'review_id': '0oB9jIpB5eKvOnqDcXYRhg', 'user_id': 'OXdp673IwpF56oYs705JPQ', 'business_id': 'Pr_Mlt9FqdcW7WCFGJmozA', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Very cute place and good food. Pear cider beer was really good the service was
really good and great ambiance and the price was very reasonable', 'date': '2016-07-03 22:22:45'}

=== Exact match search with B-Tree ===
B-Tree Search found 1 documents:
{'review_id': 'KU_O5udG6zpxOg-VcAEodg', 'user_id': 'mh_-eMZ6K5RLWhZyISBhwA', 'business_id': 'XQfwVwDr-v0ZS3_CbbE5Xw', 'stars': 3
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "If you decide to eat here, just be aware it is going to take about 2 hours from
 beginning to end. We have tried it multiple times, because I want to like it! I have been to it's other locations in NJ and nev
er had a bad experience. \n\nThe food is good, but it takes a very long time to come out. The waitstaff is very young, but usual
ly pleasant. We have just had too many experiences where we spent way too long waiting. We usually opt for another diner or rest
aurant on the weekends, in order to be done quicker.", 'date': '2018-07-07 22:09:11'}
B-Tree Time: 12.1068s, Memory: 151.58 MiB, Accuracy: 1.00

=== Exact match search with Hash Index ===
Hash Index Search found 1 documents:
{'review_id': 'KU_O5udG6zpxOg-VcAEodg', 'user_id': 'mh_-eMZ6K5RLWhZyISBhwA', 'business_id': 'XQfwVwDr-v0ZS3_CbbE5Xw', 'stars': 3
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "If you decide to eat here, just be aware it is going to take about 2 hours from
 beginning to end. We have tried it multiple times, because I want to like it! I have been to it's other locations in NJ and nev
er had a bad experience. \n\nThe food is good, but it takes a very long time to come out. The waitstaff is very young, but usual
ly pleasant. We have just had too many experiences where we spent way too long waiting. We usually opt for another diner or rest
aurant on the weekends, in order to be done quicker.", 'date': '2018-07-07 22:09:11'}
Hash Time: 11.8487s, Memory: 151.58 MiB, Accuracy: 1.00

Summary for this chunk:
- Elasticsearch: Count=10, Time=7.0383s, Mem=151.57MiB, Acc=1.00
- B-Tree: Count=1, Time=12.1068s, Mem=151.58MiB, Acc=1.00
- Hash: Count=1, Time=11.8487s, Mem=151.58MiB, Acc=1.00
```

Hình 6.1 – Kết quả mô phỏng lần một của các thuật toán

Lần 2:

```
Processing chunk 2...
Building B-Tree index...
Building Hash index...
Building Elasticsearch index...
10000 documents indexed successfully.

=== Full-text search with Elasticsearch ===
Elasticsearch found 10 documents
Time: 5.8094s, Memory: 161.97 MiB, Accuracy: 1.00
```

```
Sample Elasticsearch results:
{'review_id': 'kqDgcNIMlNrnTz4jV9XyHw', 'user_id': 'InERMUDHIu58hzdF54bVBA', 'business_id': 'TghRoAMx43V-9l7mH-SENg', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "Hadn't been here for several years.  We were pleasantly surprised.  Crab corn
 chowder is very good as is the seafood gumbo.  Very good quality seafood and very good preparation.  Good portions and reason
able prices.  Service was very good.  We'll be back.", 'date': '2012-05-28 23:54:45'}
{'review_id': 'yJnoXjg-am_Fsee6XdEq-g', 'user_id': 'St6m55Szn557A1TUabzFIg', 'business_id': 'YpWZGcSERoROrulARYyxTg', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Very good food, good service, I felt it was rather pricey. Atmosphere was very
good as well. Had the pancakes and eggs. Everyone in our party thought that their food was good.', 'date': '2017-08-18 01:06:47'
}
{'review_id': '3NSGm7ymNFA-jrL4fhCFzw', 'user_id': 'TuYjN4LIECwHKlAUPT-GSw', 'business_id': 'O9W0Qs32RL5kU2_9oqZCKQ', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Good pho.  Well prepared.  Spring rolls are skimpy. Only 3 given with order.  O
verall. Good price. Good food. Will be back again.', 'date': '2018-05-09 03:55:01'}
{'review_id': 'BId3PUVrQVk7IOsAbL_hvw', 'user_id': 'z9MOLWezOwviyS7i6f-K7g', 'business_id': '8cdOG1T2aaj8ahEkcA79iQ', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "Vaping made easy. Good flavors, good prices, good service, need I say more? Par
king is a little tricky but that's the only negative.", 'date': '2015-07-04 07:09:28'}
{'review_id': 'xC53bH30ya3d6jdCmMcSNg', 'user_id': 'uWiz6wwpw3BO9G9tWGArkQ', 'business_id': '5FDt7sy-70y4_37Dh2qcbw', 'stars': 3
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': "Pretty typical Mexican restaurant. Good chips & salsa. Margaritas are pretty de
cent. I have the chimichanga. I liked it. If you're in the area looking for good food and a good price, id go here. Prices are g
ood and the food comes quickly.", 'date': '2017-09-20 14:31:50'}
{'review_id': 'dA0pjyh0YQi-90efA6gvpA', 'user_id': '1Mg6dBktceR27KFx4zarKw', 'business_id': 'LQcGL4hfJAeK6bk2ZdhmXw', 'stars': 5
.0, 'useful': 2, 'funny': 0, 'cool': 1, 'text': 'Looks can be deceiving! I love the carne asada tacos! Carmello is good, Sonora
dogs are good, everything here is good!!\nYou must try it!', 'date': '2017-10-25 04:51:01'}
{'review_id': 'JZAca406y3pvJKFw6EXdFg', 'user_id': 'X7CLECaxz2eCtbprljjrxQ', 'business_id': 'aEEC4D-RylmiUyephE5t_g', 'stars': 4
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': 'Good solid food.  I enjoy lunch here at least once a week.  If you are in the
 Cummins Station area get lunch here.  Good food, good service.', 'date': '2013-09-04 01:10:27'}
{'review_id': 'P-2XaKsF_uaHE6ZWn1iSnQ', 'user_id': 'NYQze5Zr4QHVR-Fkf_qK_Q', 'business_id': 'g04aAvgol7IW8buqSbT4xA', 'stars': 5
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': 'Good place for a good breakfast and coffee. \nSimple wholesome and good service
 what else can one ask for breakfast?\nIf you are visiting New Orleans at the French quarter and looking for a good place to sta
rt your day, then that this is the place for you. \nRecommended...', 'date': '2016-05-07 14:27:20'}
{'review_id': '8moQ3ZKgJmYwenyRf2Z9FQ', 'user_id': '7Zx6f7Ffz0UptSV-CyObCw', 'business_id': 'QyMoP9r9KTht-r54EL2elQ', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Via Marconis is always a good time.  Lots of beers on tap and a robust menu.  T
```

```
=== Exact match search with B-Tree ===
B-Tree Search found 1 documents:
{'review_id': 'aUTP_Euzq0ZTtkN3xf3yPQ', 'user_id': 'w5iiF7drL4mKBBO8oAUQqg', 'business_id': '7lwe7n-Yc-V9E_HfLAeylg', 'stars': 4
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': "A nice neighborhood feel, but it isn't the first place to pop into mind when as
ked for recommendations. The interior and service are nice, but the food and drinks are pricey for what is served, unless you go
 for their outdoor summer happy hour!", 'date': '2015-01-18 17:56:43'}
B-Tree Time: 13.7875s, Memory: 161.97 MiB, Accuracy: 1.00

=== Exact match search with Hash Index ===
Hash Index Search found 1 documents:
{'review_id': 'aUTP_Euzq0ZTtkN3xf3yPQ', 'user_id': 'w5iiF7drL4mKBBO8oAUQqg', 'business_id': '7lwe7n-Yc-V9E_HfLAeylg', 'stars': 4
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': "A nice neighborhood feel, but it isn't the first place to pop into mind when as
ked for recommendations. The interior and service are nice, but the food and drinks are pricey for what is served, unless you go
 for their outdoor summer happy hour!", 'date': '2015-01-18 17:56:43'}
Hash Time: 12.7562s, Memory: 161.92 MiB, Accuracy: 1.00

Summary for this chunk:
- Elasticsearch: Count=10, Time=5.8094s, Mem=161.97MiB, Acc=1.00
- B-Tree: Count=1, Time=13.7875s, Mem=161.97MiB, Acc=1.00
- Hash: Count=1, Time=12.7562s, Mem=161.92MiB, Acc=1.00
```

Hình 6.2 – Kết quả mô phỏng lần hai của các thuật toán

Lần 3:

```
Processing chunk 3...
Building B-Tree index...
Building Hash index...
Building Elasticsearch index...
10000 documents indexed successfully.

=== Full-text search with Elasticsearch ===
Elasticsearch found 10 documents
Time: 7.7527s, Memory: 162.96 MiB, Accuracy: 1.00
```

```
Sample Elasticsearch results:
{'review_id': '6GJG8gwEmg1E3JnvCcWIqw', 'user_id': 'hPjXc__euN1LB1rXNy4KYw', 'business_id': 'f82dhKNiUXsDVPMLqKYiIQ', 'stars': 3
.0, 'useful': 2, 'funny': 1, 'cool': 0, 'text': "On a scale from good to Indian, this place scored a good.  It has a nice ambian
ce and the buffet has a good selection.  The food was heavy and the flavors are good for arizona, but not really good for Indian
 food in it's original form.  If you need a fix the buffet will get you in and out fast. Good for AZ.", 'date': '2011-02-18 20:2
8:22'}
{'review_id': 'BuaOGjyq9E_szhge2j5kjg', 'user_id': 'ySNgzQmce0rwDVPUT6tJDg', 'business_id': 'joeRm-_7T0XTkMdKo2nbaA', 'stars': 3
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "Bravo sits in a fun area of Wayne--right off Lancaster Ave, which can be pretty
 busy at the dinnertime hour, on a slightyl less busy street. Parking is still something to consider before 6:00 when you have t
o feed the meter, but all in all, it's still a good experience.\n\nThe pizza is good. It's a little on the greasy side, but over
all, good cheese, good sauce, good toppings, good crust. Good stuff, not excellent. The restaurant itself is very well maintaine
d.", 'date': '2013-07-15 16:50:56'}
{'review_id': 'waWa0dNoAR-kreXw_FQDhg', 'user_id': 'G5nBrEuXyVMYPwD9cVtN4g', 'business_id': 'M2ZrrqseHE5xssUrOL0Gtg', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Good food, good service. Never an issue. Never a bad meal. A+ on price too. Pan
cakes are very good.', 'date': '2017-07-30 02:07:28'}
{'review_id': '_Ai0b_wMij15gUdZ2yUV_A', 'user_id': 'w7CxrM25vRoxDpJOxoiZCw', 'business_id': '_E0WPHfJEYmwOUCAj-Jq1A', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'this place is awesome they have good food in restaurants and good gambling. pri
ces are fair on everything and good service.', 'date': '2012-08-17 20:05:53'}
{'review_id': 'KmMid4GRMGMuJW0l5SAxTQ', 'user_id': 'JYW5sdgLxUoGEU_zpGXx2w', 'business_id': 'JUlsvVAvZvGHWFfkKm0nlg', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 1, 'text': 'This place is so addicting, good vegan options, seitan or veggie tacos and guac
, good beers, good service. Love coming here!', 'date': '2018-03-24 23:37:01'}
{'review_id': 'OsTUenMc6ukeryD-gEE1uw', 'user_id': 'NgGnpTgk1ZeyzMhEj3rhIQ', 'business_id': 'SZU9c8V2GuREDN5KgyHFJw', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "This place is at the very end of Stearns Wharf ..... its not very big and it ta
kes awhile to get a table ....however the guys that work there are very efficient and take good care once you're seated..... cla
m chowder---pretty good ....lobster roll --- again pretty good ....Dungeness Crab also good ......overall its a good seafood res
taurant with nice Pacific Ocean views ....", 'date': '2018-08-06 01:37:04'}
{'review_id': 'MqG4OzM-BRhzmzQ2zi8jTA', 'user_id': 'WnP1UmehGKkCLmPlaJZ-9w', 'business_id': 'nQTJn9kdpU9Mns-b_qduVw', 'stars': 4
.0, 'useful': 1, 'funny': 1, 'cool': 1, 'text': 'always a good time.  good music and good dancing.  great bartenders.  agreed-hi
pster central....but once everyones drunk, images go out the window', 'date': '2008-09-07 19:57:57'}
{'review_id': 'Ujl9tdO4tjJgnKkEq1Uq0g', 'user_id': 'kbf2_FXGWJCBOrZFFf04vQ', 'business_id': 'gGyqnAlpFrka_qzpO7j4lQ', 'stars': 5
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "It was a really good experience. Things are expensive as you can probably guess
```

```
=== Exact match search with B-Tree ===
B-Tree Search found 1 documents:
{'review_id': 'y1ms55XoypPvBCAzJTTRxw', 'user_id': 'ORmds-VcRh70RWBFaDYVRw', 'business_id': '_RwlMTw9uFeOkfX9Ctf1HA', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "Blue Claws can be pricey but phenomenal. Stick with the larges i/o jumbo, sizes
 very close and save some $. Great with fries and beer. I've never had anything but those 3 things so couldn't tell you much els
e but worth the visit if you're looking for crabs in Philly.", 'date': '2013-08-17 01:37:19'}
B-Tree Time: 12.5327s, Memory: 162.96 MiB, Accuracy: 1.00

=== Exact match search with Hash Index ===
Hash Index Search found 1 documents:
{'review_id': 'y1ms55XoypPvBCAzJTTRxw', 'user_id': 'ORmds-VcRh70RWBFaDYVRw', 'business_id': '_RwlMTw9uFeOkfX9Ctf1HA', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "Blue Claws can be pricey but phenomenal. Stick with the larges i/o jumbo, sizes
 very close and save some $. Great with fries and beer. I've never had anything but those 3 things so couldn't tell you much els
e but worth the visit if you're looking for crabs in Philly.", 'date': '2013-08-17 01:37:19'}
Hash Time: 12.4740s, Memory: 162.96 MiB, Accuracy: 1.00

Summary for this chunk:
- Elasticsearch: Count=10, Time=7.7527s, Mem=162.96MiB, Acc=1.00
- B-Tree: Count=1, Time=12.5327s, Mem=162.96MiB, Acc=1.00
- Hash: Count=1, Time=12.4740s, Mem=162.96MiB, Acc=1.00
```

Hình 6.3 – Kết quả mô phỏng lần ba của các thuật toán

Lần 4:

```
Processing chunk 4...
Building B-Tree index...
Building Hash index...
Building Elasticsearch index...
10000 documents indexed successfully.

=== Full-text search with Elasticsearch ===
Elasticsearch found 10 documents
Time: 7.2949s, Memory: 163.49 MiB, Accuracy: 1.00
```

```
Sample Elasticsearch results:
{'review_id': 'fecS8I22s-4xVUihK9rKgQ', 'user_id': 'OB6799oy9wgup38YD164ng', 'business_id': 'CS_GzUYlEPa6QHTRY224wQ', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Came here 7/2, @6pm, got seated, got drinks, got served, got dinner, all good,
no problems as with other reviews.Good meal, good service, good time. Good night and good bye!', 'date': '2018-07-29 01:42:47'}
{'review_id': '5YfHuE4hCmyM7dwexesPZg', 'user_id': '6OkLTeAykFIH64D0VXpBkw', 'business_id': '35GCFtezJ3upyqGf-zJ8MA', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Good ambiance (very good for an Indian restaurant!).  Service was attentive (al
so very good for an Indian restaurant :)). Food was well spiced (saag planner was very good Baingan Bharta was ok). Naan and oni
on kulcha were also good.', 'date': '2014-12-21 17:00:11'}
{'review_id': 'lRLbPpcW0ylWzuDPvnBIPQ', 'user_id': 'ZUdxJPc8qX07cEu2doXakw', 'business_id': 'wzIN0IqcNOnUjwuzRoG9AA', 'stars': 5
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': "Definite 5 stars...went with a meat eater...people are great...owners are reall
y nice.\nOkra: really good\nRed beans: really good\nSweet potato: really good\nFried cauliflower: only thing better is Popeyes..
.this was awesome\nPortobello mushrooms: I didn't love them, but I'm not a big mushroom person\nPotato salad: good\nIceccream: r
eally good\nSeating is a little tight, but all good.", 'date': '2018-08-06 19:19:43'}
{'review_id': 'OUFQM8TW_4toBoSY1F3mnQ', 'user_id': 'ww9cJV3z1EnlhWFxmO2X4w', 'business_id': 'bbEXAEFr4RYHLlZ-HFssTA', 'stars': 5
.0, 'useful': 0, 'funny': 2, 'cool': 1, 'text': "My girlfriend and I were so high, we decided to eat here. Wasn't our first time
, but we had the munchies bruh. Always a good experience eating here. Good weed, good fries, good burgers.", 'date': '2014-02-26
 21:19:55'}
{'review_id': 'DzLfYfEIzNtqXyYzqHWr0g', 'user_id': 'kFJHw8-IP1PbbL4T6QIivw', 'business_id': '1bJxvwuMTyXmQGu90WLPhA', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "Waitress was great. Food was good. I had the shrimp strawberry salad which was
good. We also ate steak, chicken and salmon. All were good. They're a little pricey in my opinion but overall good food.", 'date
': '2018-09-15 20:21:19'}
{'review_id': 'ty7JHfkahsEL4hbT8mB-JA', 'user_id': '_AoxFsS1XOIZtPuo2ygSyw', 'business_id': 'gfPDLZimZu1NtBIDbeXetg', 'stars': 3
.0, 'useful': 2, 'funny': 0, 'cool': 0, 'text': 'The food is good.  The gobi manchurian is particularly delicious.\n\nChicken vi
ndaloo is good and naan is very good.', 'date': '2015-11-22 02:37:43'}
{'review_id': 'cFU6iTYACaKH4wep_Pc2eg', 'user_id': '1ZwIGAUyqxURWf5P_3jCBQ', 'business_id': 'F2C5ENuY8CXfgoW-gAMdDA', 'stars': 3
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'The food is good. The  service is good but the salsas are really good if you sc
tually like heat. I love them', 'date': '2015-02-06 02:38:25'}
{'review_id': 'MJUQ2VzsHKx4NKUO71lhNg', 'user_id': 'KVSy3XbdBi4XDPILGuZYQQ', 'business_id': 'U3cudGMQsFqpAgpnonvYqA', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'Good food, good service, and good prices the only complaint I have is the place
 is small. I felt little cramped in there.', 'date': '2014-05-31 02:17:40'}
{'review_id': 'XcjSy9OosTw6dBJekKfmBA', 'user_id': 'hgE5yTf5sh-aXxK_9ffscw', 'business_id': 'mcr1lAEdvGLMJhuPwI3I2A', 'stars': 2
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "Totally overrated. \n\nFood doesn't taste good. Beef noodle was over-cooked; be
```

```
=== Exact match search with B-Tree ===
B-Tree Search found 1 documents:
{'review_id': '3g2H3grOcCEngesXkaZ6yA', 'user_id': 'vBO5OY159gfFpeZtSs4dlw', 'business_id': 'D22mY6Lmf_32hh5-ESXFPw', 'stars': 5
.0, 'useful': 2, 'funny': 0, 'cool': 1, 'text': "This place is fantastic. I was apprehensive (as I often am walking into bike sh
ops) when I walked in, but quickly realized that this place was different. There were some hard core road bikers in the back tal
king in their own language, a few weekend warriors getting their bikes worked on and getting gear, and me: an aspiring cyclist i
ntimidated by the prices, lingo, and other details of taking up this pastime.  I normally cringe when I walk into bike stores an
d feel that if I don't drop 4 grand quickly on a carbon fiber bike or show them the yellow jersey I wore when leading three stag
es of the Tour de France, that I'll quickly be shown the exit (or a brusque cold shoulder).\n\nAt this place, a guy came up and
asked what I was looking for and how he could help. I explained my situation: I have a road bike, I want to start using it as my
 primary means of transportation and for recreation, and I want to get the rest of the equipment that I need. He quickly explain
ed what equipment is NEEDED versus what equipment is NICE TO HAVE. We went through helmets, floor pumps, bike shorts, bike shoes
, flat kits, etc. I already have shoes and pedals as well as some tire repair kits. When I told him that, he quickly moved on -
never pressuring me to upgrade or to change anything. We talked tire options (tube liners/extra-thick tires/tube sealant), tune
up needs, maintenance suggestions, etc.\n\nIn the end, I only bought a helmet because I had all of the gear in the NEED category
 already. As I get going, I'll add the NICE TO HAVE stuff as I see fit. And I'll buy it from Reno Cycling & Fitness because they
 know how to treat customers. So it is amazing, for a fairly minor purchase, this guy invested nearly half an hour talking to me
 - never talking down to me. He understood my needs, listened when i talked and helped me immensely.\n\nIn a world of diminishin
g customer service, this place stands out. The depth of knowledge available will not be found in a big box store or at an online
 discounter. Keep honest, hardworking local companies in business and help yourself in the process.", 'date': '2013-08-18 03:13:
06'}
B-Tree Time: 13.3487s, Memory: 163.48 MiB, Accuracy: 1.00
```

```
=== Exact match search with Hash Index ===
Hash Index Search found 1 documents:
{'review_id': '3g2H3grOcCEngesXkaZ6yA', 'user_id': 'vBO5OY159gfFpeZtSs4dlw', 'business_id': 'D22mY6Lmf_32hh5-ESXFPw', 'stars': 5
.0, 'useful': 2, 'funny': 0, 'cool': 1, 'text': "This place is fantastic. I was apprehensive (as I often am walking into bike sh
ops) when I walked in, but quickly realized that this place was different. There were some hard core road bikers in the back tal
king in their own language, a few weekend warriors getting their bikes worked on and getting gear, and me: an aspiring cyclist i
ntimidated by the prices, lingo, and other details of taking up this pastime.  I normally cringe when I walk into bike stores an
d feel that if I don't drop 4 grand quickly on a carbon fiber bike or show them the yellow jersey I wore when leading three stag
es of the Tour de France, that I'll quickly be shown the exit (or a brusque cold shoulder).\n\nAt this place, a guy came up and
asked what I was looking for and how he could help. I explained my situation: I have a road bike, I want to start using it as my
 primary means of transportation and for recreation, and I want to get the rest of the equipment that I need. He quickly explain
ed what equipment is NEEDED versus what equipment is NICE TO HAVE. We went through helmets, floor pumps, bike shorts, bike shoes
, flat kits, etc. I already have shoes and pedals as well as some tire repair kits. When I told him that, he quickly moved on -
never pressuring me to upgrade or to change anything. We talked tire options (tube liners/extra-thick tires/tube sealant), tune
up needs, maintenance suggestions, etc.\n\nIn the end, I only bought a helmet because I had all of the gear in the NEED category
 already. As I get going, I'll add the NICE TO HAVE stuff as I see fit. And I'll buy it from Reno Cycling & Fitness because they
 know how to treat customers. So it is amazing, for a fairly minor purchase, this guy invested nearly half an hour talking to me
 - never talking down to me. He understood my needs, listened when i talked and helped me immensely.\n\nIn a world of diminishin
g customer service, this place stands out. The depth of knowledge available will not be found in a big box store or at an online
 discounter. Keep honest, hardworking local companies in business and help yourself in the process.", 'date': '2013-08-18 03:13:
06'}
Hash Time: 11.9040s, Memory: 163.45 MiB, Accuracy: 1.00
```

```
Summary for this chunk:
- Elasticsearch: Count=10, Time=7.2949s, Mem=163.49MiB, Acc=1.00
- B-Tree: Count=1, Time=13.3487s, Mem=163.48MiB, Acc=1.00
- Hash: Count=1, Time=11.9040s, Mem=163.45MiB, Acc=1.00

Processing chunk 5
```

Hình 6.4 – Kết quả mô phỏng lần bốn của các thuật toán

Lần 5:

```
Processing chunk 5...
Building B-Tree index...
Building Hash index...
Building Elasticsearch index...
10000 documents indexed successfully.

=== Full-text search with Elasticsearch ===
Elasticsearch found 10 documents
Time: 7.7243s, Memory: 164.45 MiB, Accuracy: 1.00
```

```
Sample Elasticsearch results:
{'review_id': 'sxLtAVAjND7Oi5SXzBwN8g', 'user_id': 'j3wE7vUShBR59powqwZ4fw', 'business_id': 'tUhPRgJ7mBx-1fGef7Nv3w', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'A good basic italian place. Good flavor meat sauces and quality pasta. Huge por
tions and a good value.', 'date': '2017-07-01 02:08:39'}
{'review_id': 'iYE1Hp0BygGsogxtxj0Udw', 'user_id': 'e_91LiZEXtZiBrTdeXZRTg', 'business_id': '4WdDY97x4GdMYtyk1KQMnw', 'stars': 5
.0, 'useful': 1, 'funny': 0, 'cool': 1, 'text': 'Ordered the spring roll and pho combo. \n\nSpring roll was pretty good. Ingredi
ents was fresh.\n\nPho combo was very good.\n\nVery good pho broth. Hard chewy noodles (this is a good thing) with rare steak on
 top.', 'date': '2017-01-03 00:21:51'}
{'review_id': 'Qkf4sCaX5OcYdY2MqdH8cQ', 'user_id': 'RCZ5M9o2-fxgFuurpmEs3w', 'business_id': 'W6S482dRQ_9nCcLjfv8VNg', 'stars': 3
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "(3.5) ~ good overall food & service.  Note: It's mainly for Take-Out, but there
 are a few tables inside.\n\nMENU:\n** Mango Lassi = (4) very good\n** Lamb Vindaloo = (4) very good & spicy\n* Onion Kulcha = (
3.5) good\n* Raita = (3.25) pretty good, but watery...\n* Water(tap) = (3) ok, but needed to be filtered, to remove the chlorine
 taste, etc.\n***Kulfi(dessert) = Next time....", 'date': '2018-02-26 01:45:29'}
{'review_id': 'PX1PC2Nj0bsDqs8HytIgww', 'user_id': 'QEmGH7moR3npbRBnvx2Bog', 'business_id': 'skN2XhKXlcjf53uIwzAedw', 'stars': 3
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': 'We ordered take out- it was good. The meat was good, but just good. It was pret
ty oily though. The fries were good, crispy which I liked. The shakes were very yummy! I do recommend those (vanilla and chocola
te).', 'date': '2016-01-31 15:03:41'}
{'review_id': 's2edZl4O0KdFI_XsP6-yEA', 'user_id': 'wfgD_wmxnDPaF3pCkKYwIQ', 'business_id': 'rG0UTjvbmVVsh9-kGzeliQ', 'stars': 3
.0, 'useful': 2, 'funny': 0, 'cool': 0, 'text': "It's a nice, local, neighborhood joint. Acts a kind of a neighborhood hub. Good
 cookies which is key, good sandwich's and I am told the soup is good too although I have never had it. Breakfast or lunch, you'
re good either way.", 'date': '2014-01-18 17:34:57'}
{'review_id': 'uHANBAIYAn9VHXWAi5uCTA', 'user_id': 'zfs04Xxc4JAJD0pcIUfAcQ', 'business_id': 'sGy67CpJctjeCWClWqonjA', 'stars': 5
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': "Alan was amazing. He's a great shift lead and deserves a raise. Good was also g
ood. Wish napkins were kept in the dining room for guests. Good attempt at customer service but not efficient. More customers ar
e upset they can't find any. Good experience otherwise.", 'date': '2016-04-23 23:07:59'}
{'review_id': 'P2fOANoyfVffM6Bog1wrag', 'user_id': 'TNMu-Y66XfLwJGUTcltX5w', 'business_id': 'N44roXfLNkBdpINQDjEFOQ', 'stars': 4
.0, 'useful': 1, 'funny': 0, 'cool': 0, 'text': 'This was good. Great salsa and guacamole, meats were fresh and had good flavor.
 Chicken fajitas were good but not very flavorful, tasted like a healthier version, as if something was missing. Steak fajita ta
co salad was really good. We were sat quickly and served right away.', 'date': '2015-07-05 18:28:09'}
{'review_id': 'gdKj53yJ1cdr9Jh0KMRVgg', 'user_id': 'HtewfIDJCsL3b4AY0Ukefg', 'business_id': 'ixPTo6Hum7nNZ7A4VPhTXg', 'stars': 4
.0, 'useful': 0, 'funny': 0, 'cool': 0, 'text': "We've been here twice and enjoyed it both times. Had the fish and chips once (g
ood) and an assortment of burgers (very good), also liked the salads and all beef chili. As an aside, the burgers are rectangula
```

```
=== Exact match search with B-Tree ===
B-Tree Search found 1 documents:
{'review_id': 'zwu3HkuLQW0udgNb43e-dw', 'user_id': 'CigyryBCd5GFc01FnXvgcQ', 'business_id': 'u7_3L1NBWgxhBM_B-cmmnA', 'stars': 4
.0, 'useful': 5, 'funny': 0, 'cool': 2, 'text': "I was unexpectedly surprised with how much I liked this place. I came here for
a late dinner, just wanting something small and light since I've been eating all day. It's a pretty small and cozy place, with a
 few tables outside. All the tables were taken, but we were able to sit at the high tops where they made the pizzas. Best seats
in the house!\n\nThey were super efficient in making the pizzas. All the ingredients looked fresh and plentiful on the pizzas th
emselves. The pies coming out of their wood burning oven looked amazing. I enjoyed watching them make them through the night. Lo
oking at that, as well as the menu, I didn't want anything small anymore but wanted to try a little bit of everything.\n\nGot th
e Italian Cesar salad- anchovies topped the salad so it had that saltiness in each and every bite. The pizza dough as the croute
ns were warm and doughy.\n\nWhile waiting for our pizza, they had an extra calzone and hooked it up for free. Thanks!\n\nGot the
 pepperoni and crudo pies. I love the crust of these pizzas- perfectly charred from the oven. The pepperoni was crunchy and not
oily. It was like eating tiny pepperoni chips! For the crudo pie, The prosciutto was great with the buffalo mozzarella and every
 bite was salty and creamy. \n\nEven though we were full, we wanted to try the nutella pie for dessert. So glad I did! The pie w
as again perfectly charred and the nutella and marshmallow was perfectly melted. I just wish I had milk to wash it all down.\n\n
Overall a great experience. The hip hop music playing in the background topped it all off. It wasn't too loud that you could sti
ll hear yourself carry a conversation. Will def go back next time I'm in the area.", 'date': '2017-08-20 00:54:11'}
B-Tree Time: 13.2041s, Memory: 164.45 MiB, Accuracy: 1.00
```

```
=== Exact match search with Hash Index ===
Hash Index Search found 1 documents:
{'review_id': 'zwu3HkuLQW0udgNb43e-dw', 'user_id': 'CigyryBCd5GFc01FnXvgcQ', 'business_id': 'u7_3L1NBWgxhBM_B-cmmnA', 'stars': 4
.0, 'useful': 5, 'funny': 0, 'cool': 2, 'text': "I was unexpectedly surprised with how much I liked this place. I came here for
a late dinner, just wanting something small and light since I've been eating all day. It's a pretty small and cozy place, with a
 few tables outside. All the tables were taken, but we were able to sit at the high tops where they made the pizzas. Best seats
in the house!\n\nThey were super efficient in making the pizzas. All the ingredients looked fresh and plentiful on the pizzas th
emselves. The pies coming out of their wood burning oven looked amazing. I enjoyed watching them make them through the night. Lo
oking at that, as well as the menu, I didn't want anything small anymore but wanted to try a little bit of everything.\n\nGot th
e Italian Cesar salad- anchovies topped the salad so it had that saltiness in each and every bite. The pizza dough as the crouto
ns were warm and doughy.\n\nWhile waiting for our pizza, they had an extra calzone and hooked it up for free. Thanks!\n\nGot the
 pepperoni and crudo pies. I love the crust of these pizzas- perfectly charred from the oven. The pepperoni was crunchy and not
oily. It was like eating tiny pepperoni chips! For the crudo pie, The prosciutto was great with the buffalo mozzarella and every
 bite was salty and creamy. \n\nEven though we were full, we wanted to try the nutella pie for dessert. So glad I did! The pie w
as again perfectly charred and the nutella and marshmallow was perfectly melted. I just wish I had milk to wash it all down.\n\n
Overall a great experience. The hip hop music playing in the background topped it all off. It wasn't too loud that you could sti
ll hear yourself carry a conversation. Will def go back next time I'm in the area.", 'date': '2017-08-20 00:54:11'}
Hash Time: 13.1843s, Memory: 164.45 MiB, Accuracy: 1.00

Summary for this chunk:
- Elasticsearch: Count=10, Time=7.7243s, Mem=164.45MiB, Acc=1.00
- B-Tree: Count=1, Time=13.2041s, Mem=164.45MiB, Acc=1.00
- Hash: Count=1, Time=13.1843s, Mem=164.45MiB, Acc=1.00
```

Hình 6.5 – Kết quả mô phỏng lần năm của các thuật toán

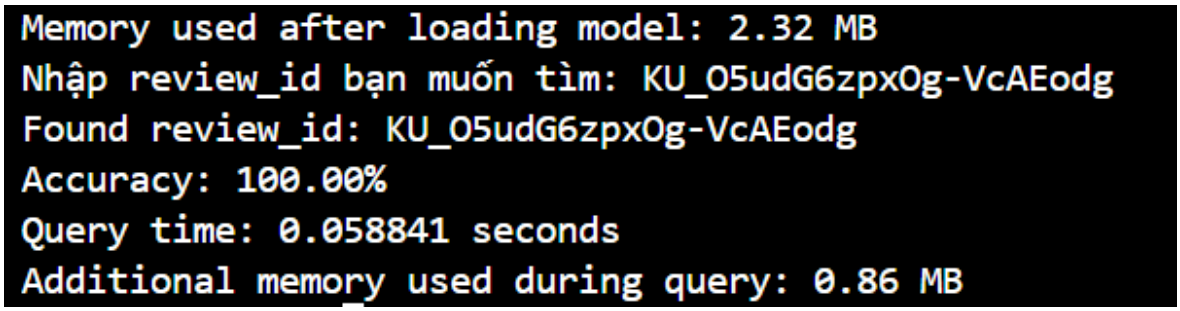
1.2 Mô hình Learned Hash Index

Train model:

```
PS D:\project> & C:/Users/HUY/AppData/Local/Programs/Python/Python312/python.exe d:/project/learned_hash_index/train.py
  Epoch [1/5] - Train Loss: 209.1923 - Test Acc: 0.0010
  Epoch [2/5] - Train Loss: 6.9325 - Test Acc: 0.0010
  Epoch [3/5] - Train Loss: 14.3661 - Test Acc: 0.0010
  Epoch [4/5] - Train Loss: 8.8154 - Test Acc: 0.0010
  Epoch [5/5] - Train Loss: 10.8593 - Test Acc: 0.0010
  Model saved at: models\saved_model.pth
```

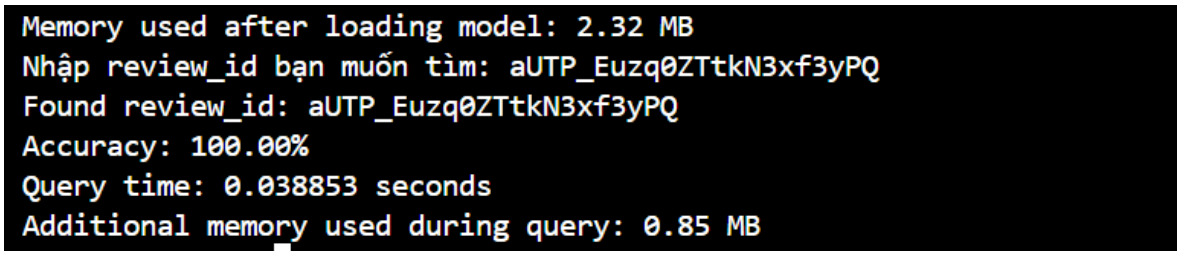
Hình 6.6 – Kết quả train model của mô hình Learned hash index

Test lần 1:



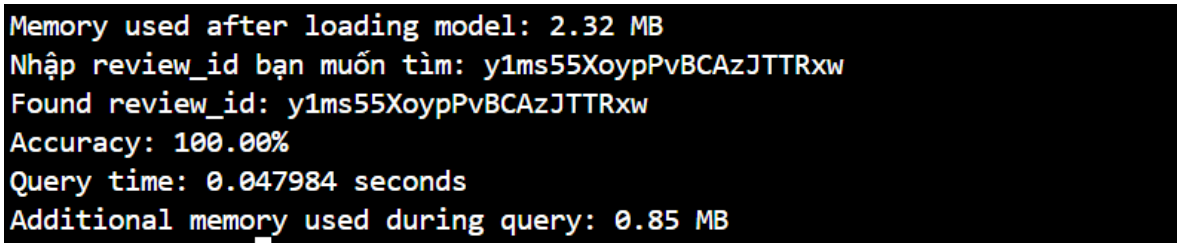
Hình 6.7 – Kết quả mô phỏng lần một của mô hình Learned hash index

Test lần 2:



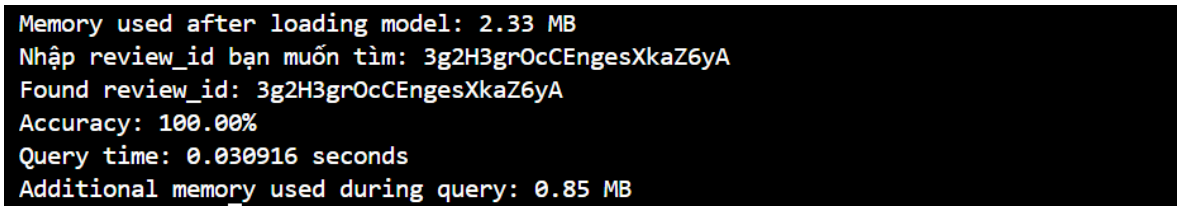
Hình 6.8 – Kết quả mô phỏng lần hai của mô hình Learned hash index

Test lần 3:



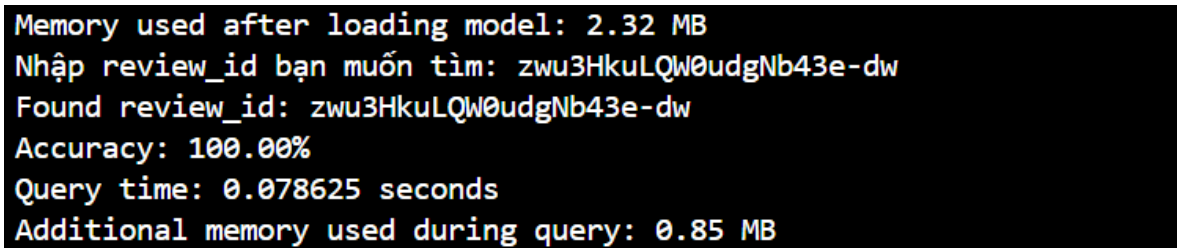
Hình 6.9 – Kết quả mô phỏng lần ba của mô hình Learned hash index

Test lần 4:



Hình 6.10 – Kết quả mô phỏng lần bốn của mô hình Learned hash index

Test lần 5:



Hình 6.11 – Kết quả mô phỏng lần năm của mô hình Learned hash index

2. So sánh kết quả giữa thuật toán gốc (Hash-base Indexing) và thuật toán tối ưu (Learned Hash Index)

Trong nghiên cứu này, chúng ta tiến hành so sánh hiệu suất của hai thuật toán lập chỉ mục dựa trên hàm băm: thuật toán gốc Hash-based Indexing và thuật toán tối ưu Learned Hash Index. Mục tiêu là đánh giá khả năng cải thiện hiệu suất, đặc biệt là về thời gian truy vấn và mức sử dụng bộ nhớ, của Learned Hash Index so với phương pháp truyền thống.

Phương pháp:

Để đánh giá, chúng ta thực hiện thử nghiệm trên cùng một tập dữ liệu với 5 truy vấn (được đại diện bởi các Review_id) cho cả hai thuật toán. Các chỉ số được đo lường bao gồm:

- Thời gian truy vấn (s): Thời gian cần thiết để tìm kiếm và trả về kết quả cho mỗi Review_id.
- Hiệu suất (%): Trong trường hợp này, được biểu thị bằng độ chính xác (Accuracy) của kết quả truy vấn.
- Bộ nhớ sử dụng (MiB): Lượng bộ nhớ RAM được sử dụng trong quá trình thực thi truy vấn.

	Review_id	Thời gian	Hiệu suất	Bộ nhớ
Lần 1	KU_O5udG6zpxOg-VcAEodg	11.8487s	100%	151.58Mib
Lần 2	aUTP_Euzq0ZTtkN3xf3yPQ	13.7875s	100%	161.97Mib
Lần 3	y1ms55XoypPVBCAzJTTRxw	12.4740s	100%	162.96Mib
Lần 4	3g2H3grOcCEngesXkaZ6yA	11.9040s	100%	163.45Mib

Lần 5	zwu3HkuLQW0udgNb43e-dw	13.1843s	100%	164.45Mib

Bảng 6.1 – Phân tích kết quả của thuật toán Hash-base Indexing

	Review_id	Thời gian	Hiệu suất	Bộ nhớ
Lần 1	KU_O5udG6zpxOg-VcAEodg	0.058841s	100%	0.86Mib
Lần 2	aUTP_Euzq0ZTtkN3xf3yPQ	0.038853s	100%	0.85Mib
Lần 3	y1ms55XoypPVBCAzJTTRxw	0.047984s	100%	0.85Mib
Lần 4	3g2H3grOcCEngesXkaZ6yA	0.030916s	100%	0.85Mib
Lần 5	zwu3HkuLQW0udgNb43e-dw	0.078625s	100%	0.85Mib

Bảng 6.2 – Phân tích kết quả của thuật toán Learned Hash Index

Từ bảng kết quả, ta có thể rút ra những nhận xét sau:

- Thời gian truy vấn: Learned Hash Index cho thấy sự cải thiện vượt trội về thời gian truy vấn. Trung bình, thời gian truy vấn của Learned Hash Index chỉ là 0.0505618 giây, nhanh hơn gấp khoảng 250 lần so với Hash-based Indexing (12.6397 giây).
- Hiệu suất: Cả hai thuật toán đều đạt hiệu suất 100% trong các lần thử nghiệm, cho thấy cả hai đều trả về kết quả chính xác cho các truy vấn.
- Bộ nhớ sử dụng: Learned Hash Index sử dụng bộ nhớ ít hơn đáng kể so với Hash-based Indexing. Trung bình, Learned Hash Index chỉ sử dụng 0.852 MiB, trong khi Hash-based Indexing sử dụng tới 160.882 MiB, tức là Learned Hash Index tiết kiệm hơn khoảng 188 lần về mức sử dụng bộ nhớ.

Kết quả thử nghiệm cho thấy Learned Hash Index vượt trội hơn hẳn so với Hash-based Indexing về cả thời gian truy vấn và mức sử dụng bộ nhớ trong khi vẫn duy trì được độ chính xác tương đương. Điều này cho thấy tiềm năng to lớn của Learned Hash Index trong việc tối ưu hóa các hệ thống lập chỉ mục, đặc biệt là trong các ứng dụng yêu cầu tốc độ truy vấn nhanh và hiệu quả sử dụng tài nguyên cao. Nghiên cứu sâu hơn có thể tập trung vào việc đánh giá Learned Hash Index trên các tập dữ liệu lớn hơn và phức tạp hơn, cũng như khám phá các phương pháp tối ưu hóa hơn nữa cho thuật toán này.

Chương 7: Kết luận và hướng phát triển

1. Kết luận

Nghiên cứu này đã tập trung phân tích và đánh giá hiệu suất của các thuật toán tìm kiếm phổ biến trên cơ sở dữ liệu lớn, bao gồm B-Tree, Hash-based Indexing, và Elasticsearch. Bên cạnh đó, nghiên cứu đã đề xuất và triển khai một giải pháp mới là Learned Hash Index, ứng dụng mô hình học máy để tối ưu hóa quá trình lập chỉ mục và truy vấn.

Các điểm chính của nghiên cứu bao gồm:

- Phân tích các thuật toán tìm kiếm truyền thống: Nghiên cứu đã đi sâu vào phân tích nguyên lý hoạt động, ưu điểm, nhược điểm của các thuật toán B-Tree, Hash-based Indexing và Elasticsearch. Qua đó, chỉ ra những thách thức khi áp dụng các thuật toán này trên dữ liệu lớn, bao gồm vấn đề về thời gian truy vấn, sử dụng bộ nhớ, và khả năng mở rộng.
- Giới thiệu Learned Hash Index: Learned Hash Index được đề xuất như một giải pháp tiềm năng để cải thiện hiệu suất tìm kiếm trên cơ sở dữ liệu lớn. Bằng cách sử dụng mô hình học máy, cụ thể là mạng nơ-ron, Learned Hash Index học cách ánh xạ khóa dữ liệu đến vị trí lưu trữ tối ưu trong bảng băm, từ đó giảm thiểu xung đột và tăng tốc độ truy vấn.
- Thực nghiệm và đánh giá: Nghiên cứu đã triển khai thử nghiệm các thuật toán B-Tree, Hash-based Indexing, Elasticsearch và Learned Hash Index trên tập dữ liệu Yelp Academic Dataset - Review. Kết quả thực nghiệm cho thấy Learned Hash Index vượt trội hơn hẳn so với Hash-based Indexing truyền thống về cả thời gian truy vấn (nhanh hơn khoảng 250 lần) và mức sử dụng bộ nhớ (tiết kiệm hơn khoảng 188 lần), trong khi vẫn đảm bảo độ chính xác tương đương (100%). Elasticsearch cũng cho thấy hiệu suất tốt về thời gian.
- Phân tích ưu, nhược điểm và thách thức: Nghiên cứu đã phân tích ưu điểm, nhược điểm của từng thuật toán, đồng thời chỉ ra những thách thức khi triển khai Learned Hash Index, bao gồm độ phức tạp của mô hình học máy, vấn đề dữ liệu không đồng đều, và nhu cầu cập nhật mô hình liên tục.

Tóm lại, Learned Hash Index cho thấy tiềm năng to lớn trong việc tối ưu hóa các hệ thống lập chỉ mục và tìm kiếm trên cơ sở dữ liệu lớn. Việc kết hợp giữa mô hình học máy và cấu trúc dữ liệu truyền thống mở ra một hướng đi mới đầy hứa hẹn cho lĩnh vực này.

2. Hướng phát triển

Dựa trên kết quả nghiên cứu, các hướng phát triển tiếp theo có thể bao gồm:

- Nghiên cứu sâu hơn về Learned Hash Index:
    - Thử nghiệm trên các tập dữ liệu lớn hơn và đa dạng hơn: Mở rộng phạm vi thử nghiệm trên các tập dữ liệu với kích thước lớn hơn, độ phức tạp cao hơn và nhiều loại dữ liệu khác nhau để đánh giá toàn diện hơn về hiệu suất và khả năng mở rộng của Learned Hash Index.
    - Tối ưu hóa mô hình học máy: Tiếp tục nghiên cứu và cải tiến mô hình học máy, bao gồm việc thử nghiệm các kiến trúc mạng nơ-ron khác nhau, tối ưu hóa siêu tham số, và áp dụng các kỹ thuật học sâu tiên tiến để nâng cao độ chính xác và giảm độ phức tạp của mô hình.
    - Giải quyết các thách thức: Tập trung giải quyết các thách thức đã được đề cập, bao gồm:
        - Phát triển các kỹ thuật để xử lý dữ liệu không đồng đều.
        - Xây dựng các cơ chế cập nhật mô hình liên tục và hiệu quả.
        - Giảm thiểu độ phức tạp của mô hình và tối ưu hóa quá trình huấn luyện.
    - Kết hợp với các cấu trúc dữ liệu khác: Nghiên cứu khả năng kết hợp Learned Hash Index với các cấu trúc dữ liệu khác như B-Tree để tạo ra các giải pháp lập chỉ mục lai (hybrid indexing), tận dụng ưu điểm của cả hai phương pháp.

- Mở rộng ứng dụng: Khám phá khả năng ứng dụng của Learned Hash Index trong các lĩnh vực khác ngoài tìm kiếm, chẳng hạn như cơ sở dữ liệu NoSQL, hệ thống lưu trữ key-value, và các ứng dụng phân tích dữ liệu lớn.
- Tích hợp với các công nghệ mới: Nghiên cứu khả năng tích hợp Learned Hash Index với các công nghệ mới như blockchain, điện toán biên (edge computing), và học liên kết (federated learning) để xây dựng các hệ thống tìm kiếm phân tán, bảo mật và hiệu quả hơn.

3. Đề xuất hướng nghiên cứu về Quantum Approximate Optimization Algorithm (QAOA) cho việc tối ưu hóa tài nguyên tìm kiếm

Một hướng nghiên cứu mới đầy tiềm năng là ứng dụng **Quantum Approximate Optimization Algorithm (QAOA)** để tối ưu hóa tài nguyên tìm kiếm, đặc biệt là trong bối cảnh Learned Hash Index.

Lý do đề xuất:

- Tiềm năng của tính toán lượng tử: Máy tính lượng tử, với khả năng tính toán song song vượt trội so với máy tính cổ điển, hứa hẹn mang lại những đột phá trong nhiều lĩnh vực, bao gồm tối ưu hóa.
- QAOA phù hợp cho bài toán tổ hợp: QAOA là một thuật toán lượng tử lai (hybrid quantum-classical algorithm) được thiết kế để giải quyết các bài toán tối ưu hóa tổ hợp, vốn là dạng bài toán phổ biến trong lĩnh vực khoa học máy tính, bao gồm cả việc tối ưu hóa tài nguyên.
- Tối ưu hóa bảng băm: Việc ánh xạ khóa dữ liệu đến các vị trí lưu trữ tối ưu trong bảng băm (như trong Learned Hash Index) có thể được xem như một bài toán tối ưu hóa tổ hợp, nơi QAOA có thể được áp dụng.

Hướng nghiên cứu cụ thể:

- Biểu diễn bài toán tối ưu hóa bảng băm dưới dạng thức phù hợp với QAOA: Cần định nghĩa hàm mục tiêu (ví dụ: tối thiểu hóa xung đột, tối ưu hóa thời gian truy vấn) và các ràng buộc (ví dụ: giới hạn bộ nhớ) dưới dạng

toán tử (operators) phù hợp với formalization của QAOA (thường là Hamiltonian).

- Thiết kế mạch lượng tử (quantum circuit) cho QAOA: Xây dựng mạch lượng tử phù hợp để giải quyết bài toán tối ưu hóa bảng băm, bao gồm việc lựa chọn các cổng lượng tử (quantum gates) và tham số hóa các góc quay (rotation angles).
- Tối ưu hóa các tham số của QAOA: Sử dụng các thuật toán tối ưu hóa cổ điển (ví dụ: gradient descent) để tìm ra các tham số tối ưu cho mạch lượng tử QAOA, nhằm đạt được nghiệm tối ưu (hoặc gần tối ưu) cho bài toán.
- Mô phỏng và thực nghiệm: Tiến hành mô phỏng QAOA trên các máy tính giả lập lượng tử (quantum simulators) và/hoặc thử nghiệm trên các máy tính lượng tử hiện có (với số lượng qubits hạn chế) để đánh giá hiệu suất của thuật toán.
- So sánh với các phương pháp truyền thống và Learned Hash Index: So sánh hiệu suất của QAOA với các phương pháp lập chỉ mục truyền thống (như Hash-based Indexing) và Learned Hash Index về các tiêu chí như thời gian truy vấn, mức sử dụng bộ nhớ, và độ chính xác.

Thách thức:

- Giới hạn của phần cứng lượng tử: Các máy tính lượng tử hiện tại vẫn còn hạn chế về số lượng qubits và độ trung thực (fidelity) của các cổng lượng tử, gây khó khăn cho việc triển khai QAOA trên các bài toán thực tế.
- Độ phức tạp của QAOA: Việc thiết kế mạch lượng tử và tối ưu hóa các tham số của QAOA có thể phức tạp và đòi hỏi chuyên môn sâu về tính toán lượng tử.

Mặc dù còn nhiều thách thức, việc ứng dụng QAOA để tối ưu hóa tài nguyên tìm kiếm, đặc biệt là trong bối cảnh Learned Hash Index, là một hướng nghiên cứu đầy tiềm năng và đáng để khám phá. Thành công trong lĩnh vực này có thể mở ra những cánh cửa mới cho việc xây dựng các hệ thống tìm kiếm hiệu suất cao, tận dụng sức mạnh của cả học máy và tính toán lượng tử.

Tài liệu tham khảo

[1] R. Elmasri, *Fundamentals of database systems*. Pearson Education India, 2008.

[2] V. Mayer-Schönberger and K. Cukier, *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013.

[3] G. Graefe, "Modern B-Tree Techniques," *Found. Trends databases,* vol. 3, no. 4, pp. 203–402, 2011, doi: 10.1561/1900000028.

[4] F. S. Patel and D. Kasat, "Hashing based indexing techniques for content based image retrieval: A survey," 2017.

[5] S. Kuppusamy, *Mastering Elasticsearch: A Comprehensive Guide*. Saravanan, 2024.

[6] D. Malyuta *et al.*, "Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently," *IEEE Control Systems Magazine,* vol. 42, no. 5, pp. 40-113, 2022.

[7] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 international conference on management of data*, 2018, pp. 489-504.

[8] R. Marcus, E. Zhang, and T. Kraska, "Cdfshop: Exploring and optimizing learned index structures," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2789-2792.