

TRƯỜNG ĐẠI HỌC MỎ-ĐỊA CHẤT
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO HỌC THUẬT
TỐI ƯU HÓA TRUY VẤN TRONG
HỆ CƠ SỞ DỮ LIỆU PHÂN TÁN

PHẠM AN CƯỜNG

12/2025

MỤC LỤC

MỞ ĐẦU	1
Lý do chọn đề tài.....	1
Mục tiêu nghiên cứu.....	2
Đối tượng và phạm vi nghiên cứu.....	2
Phương pháp thực hiện	3
Cấu trúc báo cáo.....	4
CHƯƠNG 1	6
TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU PHÂN TÁN.....	6
1.1. Khái niệm và kiến trúc hệ cơ sở dữ liệu phân tán.....	6
1.1.1. Định nghĩa.....	6
1.1.2. Các mục tiêu và tính trong suốt (Transparency).....	6
1.1.3. Kiến trúc DDBMS.....	7
1.2. Ưu điểm và thách thức.....	7
1.2.1. Ưu điểm	7
1.2.2. Thách thức.....	8
1.3. Các mô hình phân mảnh và phân phối dữ liệu	8
1.3.1. Phân mảnh ngang (Horizontal Fragmentation - HF).....	9
1.3.2. Phân mảnh dọc (Vertical Fragmentation - VF)	9
1.3.3. Phân mảnh hỗn hợp (Hybrid Fragmentation)	9
1.3.4. Phân phối và sao chép dữ liệu (Data Allocation and Replication)	9
1.4. Các hệ quản trị phổ biến	10
1.4.1. Google Spanner	10
1.4.2. Apache Cassandra	10
1.4.3. Amazon Aurora	11
CHƯƠNG 2	12
TRUY VẤN TRONG HỆ CƠ SỞ DỮ LIỆU PHÂN TÁN	12
2.1. Khái niệm truy vấn toàn cục và truy vấn con (mảnh).....	12
2.2. Biến đổi truy vấn toàn cục thành truy vấn mảnh	12
2.2.1. Phân tích và chuẩn hóa	13
2.2.2. Đơn giản hóa.....	13
2.3. Quá trình xử lý và tối ưu truy vấn trong môi trường phân tán	14
CHƯƠNG 3	17

CÁC KỸ THUẬT TỐI ƯU HÓA TRUY VẤN	17
3.1. Tối ưu hóa logic (Logical Optimization / Query Rewriting)	17
3.1.1. Các luật đại số quan hệ chính.....	17
3.1.2. Ví dụ về chuyển đổi biểu thức truy vấn	18
3.2. Tối ưu hóa vật lý (Physical Optimization)	18
3.2.1. Lựa chọn thuật toán thực hiện	18
3.2.2. Chiến lược truyền dữ liệu	19
3.2.3. Định vị truy vấn và dữ liệu	19
3.3. Lập kế hoạch truy vấn (Query Planning)	19
3.3.1. Biểu đồ truy vấn (Query Tree / QEP)	19
3.3.2. Cost-based vs. Rule-based Optimization	20
3.3.3. Quy hoạch động trong tối ưu hóa (Dynamic Programming)	20
3.4. Ví dụ minh họa và đánh giá hiệu năng.....	20
CHƯƠNG 4	23
CÁC CHIẾN LƯỢC HIỆN ĐẠI TRONG TỐI ƯU HÓA TRUY VẤN	23
4.1. Áp dụng Machine Learning (ML) trong tối ưu hóa truy vấn	23
4.1.1. Ước lượng lực lượng và chi phí	23
4.1.2. Lựa chọn kế hoạch (Plan Selection).....	24
4.1.3. Thách thức và hạn chế.....	24
4.2. Xử lý truy vấn thích nghi (Adaptive Query Processing - AQP).....	24
4.2.1. Các kỹ thuật AQP phổ biến.....	25
4.3. Hệ thống tối ưu hóa tự động và mở rộng.....	25
4.3.1. Apache Calcite.....	25
4.3.2. Presto / Trino	26
CHƯƠNG 5	27
ỨNG DỤNG VÀ THÁCH THỨC THỰC TẾ	27
5.1. Ứng dụng trong các hệ thống thực tế	27
5.1.1. Thương mại điện tử và bán lẻ (E-commerce and Retail)	27
5.1.2. Tài chính - Ngân hàng (Finance and Banking)	27
5.1.3. Phân tích dữ liệu lớn và khoa học dữ liệu	28
5.2. Các thách thức thực tế	28
5.2.1. Dữ liệu không đồng nhất (Data Heterogeneity)	28
5.2.2. Độ trễ mạng và sự không ổn định	29

5.2.3. <i>Tính chịu lỗi và xử lý lỗi (Fault Tolerance)</i>	29
5.2.4. <i>Dữ liệu bị xiên (Data Skew)</i>	29
5.2.5. <i>Quản lý tài nguyên và đa người dùng (Resource Management and Multi-tenancy)</i>	30
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....	31
TÀI LIỆU THAM KHẢO	34

MỞ ĐẦU

Lý do chọn đề tài

Thế kỷ 21 chứng kiến sự bùng nổ theo cấp số nhân của dữ liệu, thường được gọi là kỷ nguyên "Dữ liệu lớn" (Big Data). Khối lượng dữ liệu khổng lồ này, đến từ nhiều nguồn đa dạng như mạng xã hội, thiết bị IoT, giao dịch trực tuyến và các ứng dụng khoa học, đặt ra những thách thức to lớn cho việc lưu trữ, quản lý và xử lý hiệu quả. Trước bối cảnh đó, các hệ thống cơ sở dữ liệu (CSDL) truyền thống tập trung dần bộc lộ những hạn chế về khả năng mở rộng, tính sẵn sàng và hiệu năng khi đối mặt với quy mô và độ phức tạp của dữ liệu hiện đại.

Hệ cơ sở dữ liệu phân tán (CSDLPT - Distributed Database Systems, DDBMS) đã nổi lên như một giải pháp kiến trúc mạnh mẽ để giải quyết những thách thức này. Bằng cách phân chia dữ liệu và xử lý trên nhiều máy tính (nút hoặc trạm) được kết nối qua mạng, CSDLPT mang lại nhiều lợi ích vượt trội như khả năng mở rộng linh hoạt (scalability) cho phép hệ thống phát triển cùng với sự gia tăng của dữ liệu, tính sẵn sàng cao (high availability) đảm bảo hoạt động liên tục ngay cả khi một số nút gặp sự cố, và hiệu năng cải thiện thông qua việc xử lý song song và giảm thiểu độ trễ truy cập dữ liệu cục bộ. Những ưu điểm này làm cho CSDLPT trở thành nền tảng không thể thiếu cho vô số ứng dụng quan trọng trong các lĩnh vực như thương mại điện tử, tài chính ngân hàng, viễn thông, và phân tích dữ liệu lớn.

Tuy nhiên, sự phân tán dữ liệu và xử lý trong CSDLPT cũng kéo theo những phức tạp đáng kể, đặc biệt là trong quá trình xử lý truy vấn. Một truy vấn do người dùng đưa ra trên lược đồ toàn cục có thể yêu cầu truy cập và tổng hợp dữ liệu từ nhiều mảnh (fragments) được lưu trữ tại các trạm khác nhau. Quá trình này liên quan đến việc phân rã truy vấn gốc, định vị dữ liệu, lựa chọn chiến lược truyền dữ liệu giữa các trạm, và phối hợp thực thi các truy vấn con. Chi phí truyền thông dữ liệu qua mạng thường là yếu tố tốn kém nhất và có thể trở thành "nút cổ chai" của hệ thống nếu không được quản lý hiệu quả. Một kế hoạch thực thi truy vấn không tối ưu có thể dẫn đến thời gian phản hồi rất chậm, tiêu tốn tài nguyên hệ thống một cách không cần thiết (CPU, I/O, băng thông mạng) và làm giảm trải nghiệm người dùng cũng như hiệu quả hoạt động chung.

Trong bối cảnh đó, "Tối ưu hóa truy vấn" (Query Optimization) đóng vai trò then chốt, là một trong những thành phần cốt lõi và thách thức nhất trong thiết kế và vận hành CSDLPT. Mục tiêu của tối ưu hóa truy vấn là tìm ra một kế hoạch thực thi truy vấn có chi phí thấp nhất (thường là thời gian thực thi ngắn nhất hoặc lượng tài nguyên sử dụng ít nhất) trong số rất nhiều kế hoạch khả thi. Hiệu quả của bộ tối ưu hóa truy vấn ảnh hưởng trực tiếp đến hiệu suất tổng thể, khả năng đáp ứng và tính kinh tế của toàn bộ hệ thống CSDL phân tán. Với sự phát triển không ngừng của các ứng dụng đòi hỏi phân tích dữ liệu thời gian thực, các hệ thống đám mây (cloud databases) và sự đa dạng hóa của các nguồn dữ liệu, việc nghiên cứu và áp dụng các kỹ thuật tối ưu hóa truy vấn tiên tiến ngày càng trở nên cấp thiết. Do đó, việc tìm hiểu sâu sắc, hệ thống hóa các kiến thức và khám phá

những xu hướng mới trong lĩnh vực tối ưu hóa truy vấn trong CSDLPT là một nhiệm vụ khoa học có ý nghĩa lý luận và thực tiễn cao.

Mục tiêu nghiên cứu

Báo cáo này được thực hiện nhằm đạt được các mục tiêu nghiên cứu chính sau đây:

- **Hệ thống hóa kiến thức nền tảng:** Trình bày một cách tổng quan và có hệ thống các khái niệm cơ bản về cơ sở dữ liệu phân tán, bao gồm kiến trúc, ưu nhược điểm, các mô hình phân mảnh và phân phối dữ liệu. Đồng thời, làm rõ những thách thức đặc thù trong quá trình xử lý truy vấn trong môi trường phân tán.
- **Nghiên cứu sâu các kỹ thuật tối ưu hóa truy vấn:** Tập trung phân tích chi tiết các giai đoạn và kỹ thuật tối ưu hóa truy vấn truyền thống trong CSDLPT, bao gồm tối ưu hóa logic (sử dụng các luật đại số quan hệ để biến đổi cây truy vấn) và tối ưu hóa vật lý (lựa chọn thuật toán thực hiện, chiến lược truyền dữ liệu, và xây dựng kế hoạch truy vấn dựa trên mô hình chi phí).
- **Khảo sát các chiến lược tối ưu hóa hiện đại:** Tìm hiểu và trình bày các phương pháp tiếp cận mới và tiên tiến trong tối ưu hóa truy vấn, như ứng dụng Học máy (Machine Learning) để cải thiện ước lượng chi phí và lựa chọn kế hoạch, hay Xử lý truy vấn thích nghi (Adaptive Query Processing) cho phép điều chỉnh kế hoạch trong quá trình thực thi.
- **Đánh giá và nhận diện xu hướng:** Phân tích ưu nhược điểm của các kỹ thuật và chiến lược khác nhau, xem xét bối cảnh ứng dụng phù hợp của chúng. Đồng thời, nhận diện các thách thức còn tồn tại trong thực tế và các định hướng nghiên cứu tiềm năng trong tương lai cho lĩnh vực tối ưu hóa truy vấn phân tán.

Đối tượng và phạm vi nghiên cứu

- **Đối tượng nghiên cứu:** Đối tượng nghiên cứu chính của báo cáo là các thuật toán, kỹ thuật, phương pháp và chiến lược được sử dụng để tối ưu hóa hiệu năng thực thi các câu truy vấn trong môi trường cơ sở dữ liệu phân tán. Cụ thể, báo cáo sẽ tập trung vào:
 - Các giai đoạn của quá trình xử lý và tối ưu hóa truy vấn phân tán.
 - Các kỹ thuật tối ưu hóa logic dựa trên đại số quan hệ và biến đổi tương đương biểu thức truy vấn.
 - Các kỹ thuật tối ưu hóa vật lý, bao gồm lựa chọn thuật toán thực hiện (đặc biệt là các thuật toán nổi như nested-loop, sort-merge, hash-join, và semi-join), chiến lược truyền dữ liệu (data shipping, query shipping).
 - Phương pháp lập kế hoạch truy vấn (query planning), bao gồm các cách tiếp cận dựa trên luật (rule-based) và dựa trên chi phí (cost-

based), cũng như vai trò của quy hoạch động (dynamic programming).

- Các chiến lược tối ưu hóa hiện đại như ứng dụng Học máy và xử lý truy vấn thích nghi.
- Các khía cạnh kiến trúc của CSDLPT có ảnh hưởng trực tiếp đến tối ưu hóa truy vấn như phân mảnh dữ liệu, sao chép dữ liệu và quản lý danh mục.

- **Phạm vi nghiên cứu:**

- Báo cáo chủ yếu tập trung vào tối ưu hóa truy vấn trong các hệ CSDL quan hệ phân tán, nơi các nguyên tắc tối ưu hóa cổ điển đã được phát triển và nghiên cứu sâu rộng. Một số đề cập có thể được thực hiện đối với các hệ NoSQL phân tán phổ biến (ví dụ Apache Cassandra) để minh họa sự đa dạng của các giải pháp, nhưng không phải là trọng tâm chính.
- Nhấn mạnh vào tối ưu hóa ở cấp độ toàn cục (global optimization), tức là các quyết định liên quan đến việc phối hợp thực thi giữa các trạm, đặc biệt là chi phí truyền thông mạng. Các khía cạnh tối ưu hóa cục bộ (local optimization) tại từng trạm sẽ được đề cập như một phần của quy trình tổng thể nhưng không đi vào chi tiết sâu như tối ưu hóa toàn cục.
- Báo cáo sẽ không đi sâu vào các vấn đề như: cơ chế kiểm soát đồng thời phân tán chi tiết, quản lý giao dịch phân tán phức tạp (ngoại trừ những ảnh hưởng của chúng đến việc lựa chọn kế hoạch truy vấn), các giao thức bảo mật chuyên sâu cho CSDLPT, hay thiết kế vật lý chi tiết (như chiến lược đánh chỉ mục tại các trạm cục bộ, trừ khi có liên quan trực tiếp đến một kỹ thuật tối ưu hóa phân tán cụ thể).
- Báo cáo mang tính chất tổng quan, phân tích và hệ thống hóa kiến thức dựa trên tài liệu hiện có, không bao gồm việc triển khai một bộ tối ưu hóa mới hay thực hiện các thử nghiệm so sánh hiệu năng (benchmark) trên các hệ thống thực tế, mặc dù các ví dụ minh họa sẽ được sử dụng để làm rõ các khái niệm.

Phương pháp thực hiện

Để đạt được các mục tiêu đề ra, báo cáo sẽ sử dụng kết hợp các phương pháp nghiên cứu sau:

- **Nghiên cứu tài liệu:** Đây là phương pháp chủ đạo, bao gồm việc thu thập, đọc hiểu, và tổng hợp thông tin từ các nguồn tài liệu học thuật và kỹ thuật uy tín. Các nguồn này bao gồm:
 - Sách giáo trình chuyên khảo về hệ cơ sở dữ liệu, cơ sở dữ liệu phân tán và tối ưu hóa truy vấn.
 - Các bài báo khoa học đăng trên các tạp chí ACM Transactions on Database Systems, VLDB Journal, IEEE Transactions on Knowledge and Data Engineering.

- Kỹ yếu của các hội nghị khoa học hàng đầu trong lĩnh vực cơ sở dữ liệu (SIGMOD, VLDB, ICDE).
- Tài liệu kỹ thuật, white papers từ các nhà cung cấp hệ quản trị CSDL phân tán và các dự án mã nguồn mở (như Google Spanner, Apache Cassandra, Apache Calcite).
- **Phương pháp phân tích:** Phân tích các khái niệm, kỹ thuật, thuật toán và chiến lược tối ưu hóa thành các thành phần nhỏ hơn để hiểu rõ bản chất, cơ chế hoạt động, ưu điểm và nhược điểm của từng yếu tố.
- **Phương pháp so sánh:** Đối chiếu, so sánh các kỹ thuật và chiến lược tối ưu hóa khác nhau dựa trên các tiêu chí như mô hình chi phí, độ phức tạp, giả định về dữ liệu và hệ thống, chi phí truyền thông, khả năng thích ứng và phạm vi ứng dụng.
- **Phương pháp tổng hợp:** Kết nối và hệ thống hóa các kiến thức đã phân tích từ nhiều nguồn khác nhau để xây dựng một bức tranh tổng thể, mạch lạc về lĩnh vực tối ưu hóa truy vấn trong CSDLPT, từ đó rút ra các nhận xét và kết luận có giá trị.
- **Phương pháp minh họa bằng ví dụ:** Sử dụng các ví dụ cụ thể về lược đồ CSDL, các dạng phân mảnh dữ liệu và các câu truy vấn SQL để làm rõ các khái niệm trừu tượng và các bước thực hiện của những kỹ thuật tối ưu hóa phức tạp.

Cấu trúc báo cáo

Ngoài phần Mở đầu, báo cáo được tổ chức thành các chương chính như sau:

- **Chương 1: Tổng quan về cơ sở dữ liệu phân tán:** Trình bày các khái niệm nền tảng về CSDLPT, bao gồm định nghĩa, kiến trúc, các ưu điểm và thách thức chính. Chương này cũng đi sâu vào các mô hình phân mảnh và phân phối dữ liệu (phân mảnh ngang, dọc, hỗn hợp) và giới thiệu một số hệ quản trị CSDLPT phổ biến hiện nay.
- **Chương 2: Truy vấn trong hệ cơ sở dữ liệu phân tán:** Mô tả quá trình xử lý một truy vấn trong môi trường phân tán, từ việc tiếp nhận truy vấn toàn cục, biến đổi nó thành các truy vấn con trên các mảnh dữ liệu, cho đến các bước tổng quan của quá trình xử lý và tối ưu hóa.
- **Chương 3: Các kỹ thuật tối ưu hóa truy vấn:** Nội dung bao gồm tối ưu hóa logic (các luật đại số quan hệ, chuyển đổi biểu thức truy vấn), tối ưu hóa vật lý (lựa chọn thuật toán thực hiện phép nối, phép bán nối; chiến lược truyền dữ liệu; định vị dữ liệu), và các phương pháp lập kế hoạch truy vấn (cây truy vấn, tối ưu hóa dựa trên chi phí và dựa trên luật, ứng dụng quy hoạch động).
- **Chương 4: Các chiến lược hiện đại trong tối ưu hóa truy vấn:** Khám phá các xu hướng và kỹ thuật mới nổi như việc ứng dụng Học máy (Machine Learning) để cải thiện các khía cạnh của tối ưu hóa, xử lý truy vấn thích nghi (Adaptive Query Processing), và giới thiệu các hệ thống tối ưu hóa tự động như Apache Calcite và Presto.

- **Chương 5: Ứng dụng và thách thức thực tế:** Nêu bật các ứng dụng điển hình của CSDLPT và tối ưu hóa truy vấn trong các lĩnh vực thực tiễn như thương mại điện tử, tài chính, phân tích dữ liệu lớn. Đồng thời, thảo luận về các thách thức gặp phải khi triển khai và vận hành trong thực tế như dữ liệu không đồng nhất, độ trễ mạng, và yêu cầu về tính chịu lỗi.
- **Kết luận và hướng phát triển:** Tóm tắt những nội dung chính và kết quả nghiên cứu của báo cáo, chỉ ra những hạn chế còn tồn tại. Đề xuất các định hướng nghiên cứu và phát triển tiềm năng trong tương lai cho lĩnh vực tối ưu hóa truy vấn trong CSDLPT.

CHƯƠNG 1

TỔNG QUAN VỀ CƠ SỞ DỮ LIỆU PHÂN TÁN

1.1. Khái niệm và kiến trúc hệ cơ sở dữ liệu phân tán

1.1.1. Định nghĩa

Một cơ sở dữ liệu phân tán (Distributed Database - DDB) là một tập hợp các cơ sở dữ liệu có liên quan logic với nhau, được phân bố trên nhiều máy tính khác nhau trong một mạng truyền thông.

Một hệ quản trị cơ sở dữ liệu phân tán (Distributed Database Management System - DDBMS) là một hệ thống phần mềm cho phép quản lý CSDL phân tán và làm cho sự phân tán này trở nên trong suốt đối với người dùng. Người dùng tương tác với hệ thống như thể đó là một CSDL tập trung duy nhất.

Các thành phần chính của một DDBMS bao gồm:

- **Các trạm (Sites hoặc Nodes):** Các máy tính độc lập trong mạng, mỗi trạm có thể lưu trữ một phần của CSDL.
- **Mạng truyền thông (Communication Network):** Cơ sở hạ tầng kết nối các trạm, chịu trách nhiệm truyền tải dữ liệu và các thông điệp điều khiển. Đặc tính của mạng (băng thông, độ trễ) ảnh hưởng lớn đến hiệu năng hệ thống.
- **Danh mục dữ liệu phân tán (Distributed Data Catalog/Dictionary):** Là một siêu dữ liệu (metadata) chứa thông tin về toàn bộ CSDL phân tán, bao gồm lược đồ toàn cục, thông tin phân mảnh, vị trí lưu trữ và thông tin sao chép của các mảnh dữ liệu. Bản thân danh mục này cũng có thể được phân tán và sao chép.

1.1.2. Các mục tiêu và tính trong suốt (Transparency)

Mục tiêu hàng đầu của một DDBMS là cung cấp tính trong suốt phân tán (distribution transparency), tức là che giấu các chi tiết về sự phân tán dữ liệu khỏi người dùng và các ứng dụng. Điều này được thể hiện qua các cấp độ trong suốt sau:

- **Trong suốt phân mảnh (Fragmentation Transparency):** Người dùng làm việc với lược đồ toàn cục mà không cần biết rằng dữ liệu thực tế đã được chia thành các mảnh nhỏ. Hệ thống tự động ánh xạ các truy vấn trên quan hệ toàn cục thành các truy vấn trên các mảnh tương ứng.
- **Trong suốt vị trí (Location Transparency):** Người dùng không cần biết vị trí vật lý (trạm nào) lưu trữ dữ liệu. Họ có thể truy vấn dữ liệu từ bất kỳ trạm nào trong hệ thống.
- **Trong suốt sao chép (Replication Transparency):** Người dùng không nhận thức được rằng dữ liệu có thể được sao chép ở nhiều nơi. Hệ thống tự động quản lý việc truy cập vào bản sao gần nhất (để tăng tốc độ đọc) và đảm bảo tính nhất quán khi cập nhật dữ liệu trên tất cả các bản sao.

- **Trong suốt cục bộ (Local Transparency):** Mỗi trạm trong hệ thống phân tán là một hệ quản trị CSDL hoàn chỉnh, có khả năng tự quản lý dữ liệu cục bộ của mình.

Ngoài tính trong suốt, DDBMS còn hướng tới các mục tiêu quan trọng khác:

- **Tính tự trị cục bộ (Local Autonomy):** Mỗi trạm có quyền kiểm soát dữ liệu và các hoạt động cục bộ.
- **Không phụ thuộc vào vị trí trung tâm (No Reliance on a Central Site):** Thiết kế này giúp tránh được "điểm lỗi duy nhất" (single point of failure), tăng cường độ tin cậy của hệ thống.
- **Tính sẵn sàng và độ tin cậy cao (High Availability and Reliability):** Nhờ vào sao chép dữ liệu và phân tán xử lý, hệ thống có thể tiếp tục hoạt động ngay cả khi một hoặc nhiều trạm gặp sự cố.

1.1.3. Kiến trúc DDBMS

- **Kiến trúc Client-Server:** Mô hình phổ biến nơi các máy khách (client) gửi yêu cầu đến các máy chủ (server) để xử lý. Trong môi trường phân tán, một yêu cầu có thể được xử lý bởi một hoặc nhiều server phối hợp với nhau.
- **Kiến trúc Peer-to-Peer (P2P):** Không có sự phân biệt giữa client và server. Mọi nút trong mạng đều bình đẳng (peer), vừa có thể gửi yêu cầu vừa có thể xử lý yêu cầu. Kiến trúc này có khả năng chịu lỗi và mở rộng rất tốt, tiêu biểu như trong hệ thống Apache Cassandra. Các nút thường sử dụng các giao thức như "gossip" để trao đổi thông tin trạng thái và "consistent hashing" để xác định vị trí dữ liệu.
- **Hệ thống đa CSDL (Multidatabase Systems - MDBS):** Đây là một dạng đặc biệt của DDBMS, nhằm mục đích tích hợp các hệ CSDL đã có sẵn, thường là không đồng nhất (heterogeneous). MDBS cung cấp một giao diện duy nhất để người dùng truy vấn dữ liệu từ nhiều nguồn khác nhau mà không cần thay đổi các hệ CSDL gốc.

1.2. Ưu điểm và thách thức

1.2.1. Ưu điểm

- **Phản ánh cấu trúc tổ chức:** Dữ liệu có thể được lưu trữ tại chính nơi nó được tạo ra và sử dụng nhiều nhất (ví dụ: dữ liệu chi nhánh ngân hàng được lưu tại chi nhánh đó), giúp tăng tính tự chủ và hiệu quả quản lý.
- **Tính sẵn sàng và độ tin cậy cao:** Khi một trạm bị lỗi, các trạm khác vẫn có thể hoạt động. Nếu dữ liệu được sao chép, người dùng vẫn có thể truy cập dữ liệu từ các bản sao, đảm bảo hệ thống hoạt động gần như liên tục.
- **Khả năng mở rộng (Scalability):** DDBMS cho phép mở rộng theo chiều ngang (horizontal scaling) một cách dễ dàng bằng cách thêm các nút mới vào hệ thống, hiệu quả hơn nhiều so với việc nâng cấp một máy chủ trung tâm duy nhất (mở rộng theo chiều dọc - vertical scaling).
- **Hiệu năng truy cập được cải thiện:**

- **Tính cục bộ của dữ liệu (Data Locality):** Các truy vấn cục bộ có thể được thực thi nhanh chóng mà không cần truyền dữ liệu qua mạng.
- **Xử lý song song (Parallel Processing):** Các truy vấn phức tạp có thể được chia nhỏ và thực thi đồng thời trên nhiều trạm, giúp giảm đáng kể thời gian phản hồi.

1.2.2. Thách thức

- **Thiết kế phức tạp:** Việc quyết định cách phân mảnh dữ liệu (fragmentation schema) và phân bổ các mảnh (allocation schema) là một bài toán rất khó, ảnh hưởng trực tiếp đến hiệu năng hệ thống.
- **Tối ưu hóa truy vấn phức tạp:** Đây là thách thức trọng tâm của báo cáo này. Chi phí truyền thông dữ liệu giữa các trạm thường là yếu tố chi phối. Bộ tối ưu hóa phải xem xét không gian các kế hoạch thực thi lớn hơn nhiều so với hệ thống tập trung, bao gồm cả việc lựa chọn vị trí thực hiện các phép toán và chiến lược truyền dữ liệu.
- **Quản lý giao dịch phân tán:** Đảm bảo thuộc tính ACID (Atomicity, Consistency, Isolation, Durability) cho các giao dịch chạy trên nhiều trạm là rất phức tạp. Các giao thức như cam kết hai pha (Two-Phase Commit - 2PC) thường được sử dụng nhưng có thể làm giảm hiệu năng và có nguy cơ bị khóa (blocking).
- **Kiểm soát đồng thời phân tán:** Ngăn chặn các vấn đề như cập nhật mất mát (lost updates) hay đọc dữ liệu bẩn (dirty reads) trở nên khó khăn hơn. Việc phát hiện và giải quyết bế tắc phân tán (distributed deadlock) cũng là một bài toán phức tạp.
- **Phục hồi lỗi phân tán:** Khi một trạm hoặc mạng bị lỗi, quá trình phục hồi phải đảm bảo tính nhất quán của toàn bộ CSDL, đòi hỏi các giao thức phục hồi phức tạp.
- **Bảo mật:** Dữ liệu được truyền qua mạng và lưu trữ ở nhiều nơi, làm tăng nguy cơ bị tấn công và truy cập trái phép.
- **Quản lý danh mục:** Danh mục phân tán phải được quản lý cẩn thận, đảm bảo tính sẵn sàng, sao chép và nhất quán.

1.3. Các mô hình phân mảnh và phân phối dữ liệu

Phân mảnh dữ liệu là quá trình chia một quan hệ toàn cục thành các mảnh nhỏ hơn. Việc này nhằm mục đích tăng tính cục bộ, cho phép xử lý song song và cải thiện hiệu quả quản lý.

Các quy tắc của phân mảnh:

1. **Tính đầy đủ (Completeness):** Toàn bộ dữ liệu của quan hệ gốc phải được chứa trong các mảnh.
2. **Tính tái tạo (Reconstructability):** Phải có khả năng tái tạo lại quan hệ gốc từ các mảnh của nó.

3. **Tính tách biệt (Disjointness):** Các mảnh không được chứa dữ liệu trùng lặp (đối với phân mảnh ngang).

1.3.1. Phân mảnh ngang (Horizontal Fragmentation - HF)

Phân mảnh ngang chia một quan hệ thành các tập con của các bộ (rows) dựa trên một điều kiện logic.

- **Ví dụ:** Quan hệ NHANVIEN(MaNV, Ten, ChucVu, Luong, MaPB) có thể được phân mảnh ngang dựa trên phòng ban (MaPB).
 - $NV_PB01 = \sigma_{\{MaPB='PB01'\}}(NHANVIEN)$
 - $NV_PB02 = \sigma_{\{MaPB='PB02'\}}(NHANVIEN)$
- **Ưu điểm:** Rất hiệu quả cho các ứng dụng thường xuyên truy cập dữ liệu theo nhóm, ví dụ như "liệt kê tất cả nhân viên của phòng ban PB01". Việc xử lý trên mỗi mảnh có thể diễn ra song song.

1.3.2. Phân mảnh dọc (Vertical Fragmentation - VF)

Phân mảnh dọc chia một quan hệ thành các tập con của các thuộc tính (columns). Để đảm bảo tính tái tạo, khóa chính của quan hệ gốc phải được lặp lại trong mỗi mảnh dọc.

- **Ví dụ:** Quan hệ NHANVIEN có thể được phân mảnh dọc để tách thông tin cá nhân và thông tin lương nhạy cảm.
 - $NV_THONGTIN = \Pi_{\{MaNV, Ten, ChucVu, MaPB\}}(NHANVIEN)$
 - $NV_LUONG = \Pi_{\{MaNV, Luong\}}(NHANVIEN)$
- **Ưu điểm:** Tăng cường bảo mật (có thể cấp quyền khác nhau trên mỗi mảnh) và cải thiện hiệu năng cho các truy vấn chỉ cần truy cập một số thuộc tính nhất định (giảm I/O).

1.3.3. Phân mảnh hỗn hợp (Hybrid Fragmentation)

Đây là sự kết hợp của hai phương pháp trên: áp dụng phân mảnh ngang cho các mảnh dọc, hoặc ngược lại.

- **Ví dụ:**
 1. Phân mảnh dọc quan hệ NHANVIEN thành NV_THONGTIN và NV_LUONG.
 2. Tiếp tục phân mảnh ngang mảnh NV_THONGTIN theo phòng ban (MaPB).

1.3.4. Phân phối và sao chép dữ liệu (Data Allocation and Replication)

Sau khi phân mảnh, bước tiếp theo là quyết định nơi lưu trữ các mảnh (phân bổ) và có tạo bản sao hay không (sao chép).

- **Phân bổ dữ liệu (Data Allocation):**
 - **Phân chia (Partitioned):** Mỗi mảnh được lưu trữ tại một trạm duy nhất. Đơn giản nhưng không có tính sẵn sàng cao.

- **Sao chép (Replicated):** Các bản sao của mảnh dữ liệu được lưu trữ tại nhiều trạm.
- **Sao chép dữ liệu (Data Replication):**
 - **Sao chép toàn bộ (Fully Replicated):** Toàn bộ CSDL được sao chép tại mọi trạm. Tối ưu cho việc đọc nhưng rất tốn kém và phức tạp khi cập nhật.
 - **Sao chép một phần (Partially Replicated):** Chỉ một số mảnh quan trọng hoặc thường xuyên được truy cập mới được sao chép. Đây là phương án cân bằng và phổ biến nhất.
 - **Ưu điểm của sao chép:** Tăng tính sẵn sàng và độ tin cậy; Tăng tốc độ truy vấn đọc do có thể truy cập bản sao gần nhất.
 - **Nhược điểm của sao chép:** Tăng chi phí lưu trữ; Phức tạp hóa việc cập nhật, đòi hỏi các giao thức để đảm bảo tính nhất quán giữa các bản sao (ví dụ: Read-One-Write-All).

1.4. Các hệ quản trị phổ biến

Dưới đây là một số DDBMS tiêu biểu, thể hiện các cách tiếp cận khác nhau để giải quyết các thách thức của hệ thống phân tán.

1.4.1. Google Spanner

- **Loại hình:** NewSQL, CSDL quan hệ phân tán toàn cầu.
- **Đặc điểm nổi bật:**
 - **Tính nhất quán mạnh (Strong Consistency):** Spanner là hệ thống thương mại đầu tiên cung cấp tính nhất quán ngoài (external consistency) và khả năng tuần tự hóa (serializability) cho các giao dịch phân tán trên quy mô toàn cầu. Điều này đạt được nhờ vào công nghệ TrueTime API, một hệ thống đồng hồ được đồng bộ hóa chặt chẽ trên các trung tâm dữ liệu, cung cấp một khoảng thời gian tin cậy cho các dấu thời gian (timestamps).
 - **Khả năng mở rộng toàn cầu:** Được thiết kế từ đầu để mở rộng ra hàng triệu máy chủ trên khắp các châu lục. Dữ liệu được tự động phân mảnh và di chuyển giữa các máy chủ để cân bằng tải.
 - **Giao diện SQL:** Cung cấp giao diện truy vấn SQL quen thuộc với các lập trình viên, hỗ trợ các giao dịch ACID.

1.4.2. Apache Cassandra

- **Loại hình:** NoSQL, CSDL dạng cột rộng (Wide-Column Store).
- **Đặc điểm nổi bật:**
 - **Kiến trúc phi tập trung hoàn toàn (Masterless):** Mọi nút trong cụm Cassandra đều giống nhau, không có nút chủ (master node), do đó không có điểm lỗi duy nhất. Điều này mang lại khả năng chịu lỗi và tính sẵn sàng vượt trội.

- **Khả năng mở rộng tuyến tính:** Hiệu năng và dung lượng tăng một cách có thể dự đoán được khi thêm các nút mới vào cụm.
- **Tính nhất quán tùy chỉnh (Tunable Consistency):** Cassandra cho phép người dùng định nghĩa mức độ nhất quán cho mỗi thao tác đọc hoặc ghi (ví dụ: ONE, QUORUM, ALL). Điều này cho phép ứng dụng cân bằng linh hoạt giữa tính nhất quán, tính sẵn sàng và độ trễ theo nhu cầu (theo định lý CAP).

1.4.3. Amazon Aurora

- **Loại hình:** CSDL quan hệ tương thích MySQL và PostgreSQL, được xây dựng cho đám mây (Cloud-Native).
- **Đặc điểm nổi bật:**
 - **Tách biệt lớp tính toán và lưu trữ:** Kiến trúc độc đáo của Aurora tách rời phần xử lý truy vấn (compute) khỏi phần lưu trữ (storage). Lớp lưu trữ là một dịch vụ phân tán, chuyên dụng, được tối ưu hóa cho các tác vụ I/O.
 - **Kiến trúc lưu trữ phân tán, tự phục hồi:** Dữ liệu được sao chép 6 bản trên 3 vùng sẵn sàng (Availability Zones - AZs) khác nhau, giúp hệ thống có khả năng chịu lỗi cực cao. Aurora có thể chịu được sự cố của một AZ hoàn chỉnh mà không làm mất dữ liệu.
 - **Hiệu năng cao:** Bằng cách chuyển các công việc như ghi log (redo logging) xuống lớp lưu trữ, các nút tính toán được giải phóng để tập trung vào việc xử lý các truy vấn đọc/ghi, mang lại hiệu suất cao hơn nhiều so với các CSDL quan hệ truyền thống. Mặc dù người dùng thấy một CSDL duy nhất, hạ tầng bên dưới hoàn toàn là phân tán.

CHƯƠNG 2

TRUY VẤN TRONG HỆ CƠ SỞ DỮ LIỆU PHÂN TÁN

2.1. Khái niệm truy vấn toàn cục và truy vấn con (mảnh)

Trong môi trường CSDLPT, chúng ta phân biệt hai cấp độ của một câu truy vấn:

- **Truy vấn toàn cục (Global Query):** Đây là câu truy vấn được người dùng hoặc ứng dụng đưa ra trên lược đồ toàn cục (global schema). Người dùng tương tác với hệ thống như thể nó là một CSDL tập trung duy nhất, nhờ vào tính trong suốt phân tán đã được đề cập. Truy vấn toàn cục được viết bằng một ngôn ngữ truy vấn cấp cao như SQL và không chứa bất kỳ thông tin nào về việc dữ liệu được phân mảnh hay phân bố ở đâu.
 - **Ví dụ về truy vấn toàn cục:** Giả sử chúng ta có một CSDL phân tán của một công ty với hai quan hệ:
 - NHANVIEN(MaNV, Ten, ChucVu, MaPB)
 - PHONGBAN(MaPB, TenPB, DiaDiem)
 - Người dùng muốn tìm tên và chức vụ của tất cả nhân viên làm việc tại phòng ban ở 'Hà Nội'. Câu truy vấn toàn cục sẽ là:

```
SELECT T.Ten, T.ChucVu
```

```
FROM NHANVIEN T, PHONGBAN D
```

```
WHERE T.MaPB = D.MaPB AND D.DiaDiem = 'Hà Nội';
```

- **Truy vấn con/truy vấn mảnh (Fragment Query/ Sub-query):** Đây là các truy vấn được suy ra từ truy vấn toàn cục và được thực thi trực tiếp trên các mảnh dữ liệu (data fragments) tại các trạm cục bộ. Hệ quản trị CSDL phân tán phải dịch truy vấn toàn cục thành một hoặc nhiều truy vấn con, xác định dữ liệu cần di chuyển giữa các trạm (nếu có), và tổng hợp kết quả từ các truy vấn con này để trả về cho người dùng.
 - **Ví dụ về truy vấn con:** Giả sử quan hệ NHANVIEN được phân mảnh ngang theo MaPB thành NV1 (MaPB < 'PB03') và NV2 (MaPB >= 'PB03'). Quan hệ PHONGBAN được phân mảnh ngang theo DiaDiem thành PB_HN (ở Hà Nội) và PB_HCM (ở TP.HCM). Truy vấn toàn cục trên có thể được biến đổi thành các truy vấn con, chẳng hạn như một truy vấn nối NV1 với PB_HN và một truy vấn khác nối NV2 với PB_HN.

Quá trình chuyển đổi từ một truy vấn toàn cục duy nhất thành một tập hợp các truy vấn con hiệu quả chính là trọng tâm của việc xử lý và tối ưu hóa truy vấn phân tán.

2.2. Biến đổi truy vấn toàn cục thành truy vấn mảnh

Đây là giai đoạn đầu tiên và mang tính logic của quá trình xử lý truy vấn, thường được gọi là phân rã truy vấn (query decomposition). Mục tiêu của giai đoạn này là biến đổi một truy vấn cấp cao trên các quan hệ toàn cục thành một

biểu thức đại số quan hệ trên các mảnh dữ liệu. Quá trình này bao gồm các bước chính sau:

2.2.1. Phân tích và chuẩn hóa

1. **Phân tích cú pháp (Parsing):** Hệ thống phân tích câu truy vấn SQL để kiểm tra tính đúng đắn về cú pháp và ngữ nghĩa. Các thông tin về quan hệ, thuộc tính và các điều kiện được trích xuất.
2. **Chuẩn hóa (Normalization):** Câu truy vấn được biến đổi về một dạng chuẩn (canonical form) để thuận tiện cho các bước xử lý sau. Ví dụ, các điều kiện trong mệnh đề WHERE có thể được chuyển thành dạng hội chuẩn (conjunctive normal form - CNF) hoặc tuyển chuẩn (disjunctive normal form - DNF). Biểu thức truy vấn thường được biểu diễn dưới dạng một cây truy vấn (query tree) hoặc đồ thị truy vấn (query graph), với các lá là các quan hệ và các nút trong là các toán tử đại số quan hệ.
 - Ví dụ: Câu truy vấn SQL ở mục 2.1 có thể được biểu diễn bằng biểu thức đại số quan hệ: $\Pi_{Ten,ChucVu}(\sigma_{DiaDiem='Ha\`N\grave{o}i'}(NHANVIEN \bowtie_{NHANVIEN.MaPB=PHONGBAN.MaPB} PHONGBAN))$

2.2.2. Đơn giản hóa

Trước khi phân rã, hệ thống sẽ cố gắng đơn giản hóa truy vấn để loại bỏ các phần dư thừa. Giai đoạn này sử dụng các quy tắc suy luận logic và ngữ nghĩa:

- **Loại bỏ các mệnh đề dư thừa:** Ví dụ, $(MaPB = 'PB01') \text{ AND } (MaPB = 'PB01')$ được rút gọn thành $(MaPB = 'PB01')$. Hoặc $(Luong > 500) \text{ AND } (Luong > 1000)$ được rút gọn thành $(Luong > 1000)$.
- **Kiểm tra mâu thuẫn:** Nếu một truy vấn chứa các điều kiện mâu thuẫn, ví dụ $(MaPB = 'PB01') \text{ AND } (MaPB = 'PB02')$, hệ thống có thể kết luận ngay rằng kết quả là rỗng và không cần thực thi tiếp.
- **Áp dụng các luật đại số:** Sử dụng các luật giao hoán, kết hợp để sắp xếp lại các toán tử, chuẩn bị cho bước tối ưu hóa sau.

2.2.3. Tái cấu trúc và bản địa hóa (Restructuring and Localization)

Đây là bước cốt lõi của quá trình phân rã. Hệ thống sử dụng thông tin từ danh mục phân mảnh để thay thế các quan hệ toàn cục trong cây truy vấn bằng các biểu thức tái tạo chúng từ các mảnh.

- **Thay thế bằng định nghĩa phân mảnh:** Mỗi quan hệ toàn cục R trong cây truy vấn được thay thế bằng biểu thức đại số quan hệ tương ứng với cách nó được phân mảnh.
 - Nếu R được phân mảnh ngang thành $R_1, R_2, \dots, R_n$, thì R sẽ được thay thế bằng $R_1 \cup R_2 \cup \dots \cup R_n$.
 - Nếu R được phân mảnh dọc thành $R_1, R_2, \dots, R_m$, thì R sẽ được thay thế bằng $R_1 \bowtie R_2 \bowtie \dots \bowtie R_m$.
 - Sau bước này, chúng ta có một cây truy vấn tổng quát (generic query tree) hoạt động trên các mảnh.

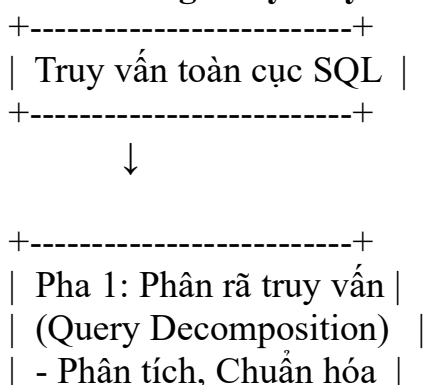
- **Tái cấu trúc và rút gọn (Pruning):** Cây truy vấn tổng quát thường rất phức tạp và dư thừa. Hệ thống sẽ áp dụng các luật đại số quan hệ để biến đổi và rút gọn nó.
 - Đẩy các phép toán SELECT (σ) và PROJECT (Π) xuống càng sâu càng tốt trong cây để giảm kích thước của các kết quả trung gian.
 - Phân phối phép JOIN ($\bowtie$) qua phép UNION ($\cup$): $(A \cup B) \bowtie C = (A \bowtie C) \cup (B \bowtie C)$.
 - **Loại bỏ các nhánh không cần thiết (Pruning):** Đây là bước quan trọng nhất. Hệ thống sẽ kiểm tra các điều kiện trong truy vấn với các điều kiện định nghĩa phân mảnh. Nếu một mảnh không thể thỏa mãn điều kiện của truy vấn, toàn bộ nhánh cây liên quan đến mảnh đó sẽ bị loại bỏ.
 - **Ví dụ về Pruning:**
 - Giả sử PHONGBAN được phân mảnh thành PB_HN ($\sigma_{\text{DiaDiem}='Ha'N\grave{o}i'}(\text{PHONGBAN})$) và PB_HCM ($\sigma_{\text{DiaDiem}='TP.HCM'}(\text{PHONGBAN})$).
 - Truy vấn của chúng ta có điều kiện $\text{DiaDiem} = 'H\grave{a} N\grave{o}i'$.
 - Khi áp điều kiện này vào mảnh PB_HCM, ta có $\sigma_{\text{DiaDiem}='Ha'N\grave{o}i'}(\text{PB_HCM}) = \sigma_{\text{DiaDiem}='Ha'N\grave{o}i'}(\sigma_{\text{DiaDiem}='TP.HCM'}(\text{PHONGBAN}))$. Biểu thức này luôn cho kết quả rỗng.
 - Do đó, mọi nhánh truy vấn liên quan đến mảnh PB_HCM sẽ được cắt tỉa (pruned), giúp giảm đáng kể không gian tính toán.

Kết quả của giai đoạn này là một truy vấn trên các mảnh (query on fragments), tương đương về mặt logic với truy vấn toàn cục ban đầu nhưng đã được tối ưu hóa sơ bộ bằng cách chỉ giữ lại các thao tác trên những mảnh dữ liệu thực sự có liên quan.

2.3. Quá trình xử lý và tối ưu truy vấn trong môi trường phân tán

Quá trình biến đổi ở mục 2.2 chỉ là pha đầu tiên trong một quy trình xử lý truy vấn phân tán tổng thể. Toàn bộ vòng đời của một truy vấn có thể được tóm tắt qua các pha chính sau đây. Sơ đồ này cho thấy một cái nhìn toàn cảnh và đóng vai trò là cầu nối đến các kỹ thuật tối ưu hóa chi phí.

Sơ đồ luồng xử lý truy vấn phân tán



- Đơn giản hóa
- Tái cấu trúc (Pruning)



Biểu thức Đại số trên các Mảnh



Pha 2: Tối ưu hóa Toàn cục (Global Optimization)
- Lựa chọn thứ tự Join
- Chọn chiến lược truyền
- Ước lượng chi phí



Kế hoạch Thực thi Toàn cục (Global Exec. Plan)



Gửi Truy vấn con đến Trạm 1	Pha 3: Tối ưu hóa Cục bộ (Local Optimization)
	- Sử dụng chỉ mục
	- Chọn thuật toán cục bộ



--



Gửi Truy vấn con đến Trạm 2	Kế hoạch Thực thi Cục bộ
--------------------------------	--------------------------



--

Tổng hợp kết quả cuối

--

Pha 1: Phân rã truy vấn (Query Decomposition): Như đã mô tả chi tiết ở mục 2.2, pha này lấy đầu vào là một câu truy vấn SQL toàn cục và tạo ra một

biểu thức đại số quan hệ tương đương trên các mảnh dữ liệu. Pha này chủ yếu dựa trên các phép biến đổi logic và chưa quan tâm nhiều đến chi phí thực thi như chi phí I/O hay truyền thông mạng.

Pha 2: Tối ưu hóa toàn cục (Global Optimization): Đây là "bộ não" của bộ xử lý truy vấn phân tán.

- **Đầu vào:** Biểu thức đại số trên các mảnh từ Pha 1.
- **Nhiệm vụ:** Tìm ra một chiến lược thực thi (execution strategy) có chi phí thấp nhất để thực hiện biểu thức này. Việc này bao gồm các quyết định quan trọng:
 - **Thứ tự thực hiện các phép toán:** Đặc biệt là thứ tự các phép nối (JOIN), vì nó ảnh hưởng rất lớn đến kích thước của các kết quả trung gian cần được truyền qua mạng.
 - **Lựa chọn thuật toán:** Lựa chọn thuật toán cụ thể cho mỗi phép toán (ví dụ: dùng Nested-Loop Join, Hash Join, hay một phép Semi-Join để giảm thiểu truyền dữ liệu).
 - **Chiến lược truyền dữ liệu:** Quyết định di chuyển dữ liệu đến nơi tính toán (data shipping) hay di chuyển phép tính đến nơi có dữ liệu (query shipping).
- **Phương pháp:** Bộ tối ưu hóa toàn cục thường là dựa trên chi phí (cost-based). Nó sẽ xây dựng nhiều kế hoạch thực thi khả thi, ước tính chi phí (CPU, I/O, và đặc biệt là chi phí truyền thông) cho mỗi kế hoạch dựa trên các số liệu thống kê về dữ liệu, và cuối cùng chọn ra kế hoạch có chi phí ước tính thấp nhất.
- **Đầu ra:** Một kế hoạch thực thi toàn cục (Global Execution Plan - GEP), trong đó mô tả chi tiết các bước thực thi, các truy vấn con cần gửi đến từng trạm, và luồng dữ liệu di chuyển giữa chúng.

Pha 3: Tối ưu hóa cục bộ (Local Optimization): Sau khi GEP được tạo ra, các truy vấn con được gửi đến các trạm tương ứng để thực thi. Tại mỗi trạm, hệ quản trị CSDL cục bộ sẽ tiếp tục tối ưu hóa truy vấn con này.

- Bộ tối ưu hóa cục bộ có kiến thức chi tiết về dữ liệu tại trạm của mình, ví dụ như các chỉ mục (indexes) hiện có, phương pháp lưu trữ vật lý, và trạng thái bộ đệm (buffer).
- Nó sẽ tạo ra một kế hoạch thực thi cục bộ (Local Execution Plan - LEP) tối ưu nhất cho truy vấn con trong phạm vi tài nguyên của trạm đó.

Sự phân chia thành các pha này cho phép hệ thống giải quyết bài toán tối ưu hóa phức tạp bằng cách chia nó thành các bài toán con dễ quản lý hơn. Quá trình biến đổi và rút gọn logic đảm bảo rằng bộ tối ưu hóa chi phí sẽ bắt đầu với một điểm khởi đầu tốt hơn, tránh phải xem xét các kế hoạch rõ ràng là không hiệu quả.

CHƯƠNG 3

CÁC KỸ THUẬT TỐI ƯU HÓA TRUY VẤN

3.1. Tối ưu hóa logic (Logical Optimization / Query Rewriting)

Tối ưu hóa logic là quá trình biến đổi một biểu thức đại số quan hệ thành một biểu thức khác tương đương về mặt kết quả nhưng hiệu quả hơn để tính toán, mà không cần xem xét chi phí thực thi chi tiết hay các thuật toán cụ thể. Mục tiêu là giảm thiểu kích thước của các kết quả trung gian và số lượng các phép toán. Nền tảng của quá trình này là các luật đại số quan hệ.

3.1.1. Các luật đại số quan hệ chính

Một truy vấn được biểu diễn dưới dạng một cây đại số quan hệ. Bộ tối ưu hóa sẽ áp dụng các luật sau để tái cấu trúc cây này:

Tính giao hoán và kết hợp của các phép toán hai ngôi:

- **Phép nối (Join - $\bowtie$):**

- $A \bowtie B \equiv B \bowtie A$ (Giao hoán)
- $(A \bowtie B) \bowtie C \equiv A \bowtie (B \bowtie C)$ (Kết hợp)
- **Ý nghĩa:** Các luật này cho phép bộ tối ưu hóa tự do sắp xếp lại thứ tự các phép nối. Việc chọn thứ tự nối đúng đắn là một trong những quyết định quan trọng nhất, vì nó có thể làm giảm đáng kể kích thước của các kết quả trung gian.

Đẩy phép chọn (σ) xuống sớm nhất có thể: Đây là quy tắc heuristic quan trọng nhất trong tối ưu hóa truy vấn.

- $\sigma_p(A \bowtie B) \equiv \sigma_p(A) \bowtie B$ (nếu điều kiện p chỉ chứa các thuộc tính của A).
- $\sigma_p(A \bowtie B) \equiv \sigma_p(A) \bowtie \sigma_p(B)$ (nếu $p = p_A \wedge p_B$).
- $\sigma_p(A \cup B) \equiv \sigma_p(A) \cup \sigma_p(B)$.
- **Ý nghĩa:** Thực hiện phép chọn càng sớm (càng gần các lá của cây) càng tốt sẽ làm giảm đáng kể số bộ (rows) của các quan hệ trước khi chúng được gửi qua mạng hoặc được sử dụng trong các phép toán tốn kém như phép nối.

Đẩy phép chiếu (Π) xuống sớm nhất có thể:

- $\Pi_L(A \bowtie B) \equiv \Pi_L(\Pi_{L_A}(A) \bowtie \Pi_{L_B}(B))$ (với L_A, L_B là tập các thuộc tính của A và B cần thiết cho phép nối và cho kết quả cuối cùng L).
- **Ý nghĩa:** Phép chiếu làm giảm "chiều rộng" của các quan hệ bằng cách loại bỏ các cột không cần thiết. Việc này giúp giảm lượng dữ liệu cần đọc từ đĩa và đặc biệt là giảm khối lượng dữ liệu truyền qua mạng.

Kết hợp các phép toán một ngôi:

- $\sigma_{p1}(\sigma_{p2}(A)) \equiv \sigma_{p1 \wedge p2}(A)$. (Kết hợp chuỗi các phép chọn).
- $\Pi_{L1}(\Pi_{L2}(A)) \equiv \Pi_{L1}(A)$ (nếu $L1 \subseteq L2$). (Loại bỏ các phép chiếu dư thừa).
- **Ý nghĩa:** Giảm số lần phải duyệt qua dữ liệu.

3.1.2. Ví dụ về chuyển đổi biểu thức truy vấn

Xét lại truy vấn: `SELECT Ten FROM NHANVIEN, PHONGBAN WHERE NHANVIEN.MaPB = PHONGBAN.MaPB AND PHONGBAN.DiaDiem = 'Hà Nội' AND NHANVIEN.ChucVu = 'Manager';`

- **Cây ban đầu (không tối ưu):** $\Pi_{Ten}(\sigma_{DiaDiem='Hà\`N\grave{o}i' \wedge ChucVu='Manager'}(NHANVIEN \bowtie PHONGBAN))$
 - Kế hoạch này thực hiện phép nối không lọc giữa hai quan hệ NHANVIEN và PHONGBAN trước, sau đó mới lọc kết quả.
- **Cây tối ưu hóa (sau khi áp dụng các luật):** $\Pi_{Ten}((\sigma_{ChucVu='Manager'}(NHANVIEN)) \bowtie (\sigma_{DiaDiem='Hà\`N\grave{o}i'}(PHONGBAN)))$
 - Kế hoạch này thực hiện các phép chọn trên NHANVIEN và PHONGBAN trước, tạo ra hai tập dữ liệu trung gian nhỏ hơn nhiều. Sau đó, nó mới thực hiện phép nối trên hai tập dữ liệu đã được thu gọn này. Kết quả là chi phí tổng thể giảm đáng kể.

3.2. Tối ưu hóa vật lý (Physical Optimization)

Sau khi có được một cây truy vấn logic tối ưu, bộ tối ưu hóa vật lý sẽ quyết định cách thức thực thi cây đó. Giai đoạn này liên quan đến việc lựa chọn các thuật toán cụ thể cho mỗi toán tử và chiến lược di chuyển dữ liệu giữa các trạm, dựa trên mô hình chi phí.

3.2.1. Lựa chọn thuật toán thực hiện

Phép nối là phép toán tốn kém nhất, do đó việc lựa chọn thuật toán nối phù hợp là cực kỳ quan trọng.

- **Nested-Loop Join:** Duyệt qua mọi bộ của quan hệ ngoài, và với mỗi bộ đó, duyệt qua toàn bộ quan hệ trong để tìm các bộ khớp. Đơn giản nhưng rất chậm, trừ khi một trong hai quan hệ rất nhỏ.
- **Sort-Merge Join:** Sắp xếp cả hai quan hệ theo thuộc tính nối, sau đó "trộn" chúng lại với nhau để tìm các cặp khớp. Hiệu quả nếu dữ liệu đã được sắp xếp hoặc cần được sắp xếp cho các bước sau.
- **Hash Join:** Băm (hash) quan hệ nhỏ hơn (build phase) vào một bảng băm trong bộ nhớ. Sau đó, duyệt qua quan hệ lớn hơn (probe phase), băm từng bộ và tìm kiếm trong bảng băm. Đây thường là thuật toán hiệu quả nhất cho các phép nối bằng (equi-joins).
- **Phép bán nối (Semi-Join - $A \ltimes B$):** Đây là một kỹ thuật cực kỳ quan trọng và đặc thù cho CSDLPT.
 - **Khái niệm:** Phép bán nối $A \ltimes B$ trả về tất cả các bộ của A có ít nhất một bộ tương ứng trong B theo điều kiện nối. Kết quả chỉ chứa các cột của A.
 - **Lợi ích cốt lõi: Giảm thiểu chi phí truyền thông.** Thay vì gửi toàn bộ quan hệ B (có thể rất lớn) đến trạm của A để thực hiện phép nối, ta có thể làm như sau:

Tại trạm của B, thực hiện phép chiếu để chỉ lấy các thuộc tính cần cho phép nối: $B' = \Pi_{\text{join_attrs}}(B)$.

Gửi B' (thường nhỏ hơn rất nhiều so với B) đến trạm của A.

Tại trạm của A, thực hiện phép bán nối: $A' = A \bowtie B'$. A' là tập hợp các bộ của A sẽ tham gia vào kết quả cuối cùng.

Gửi A' (nhỏ hơn hoặc bằng A) đến trạm của B.

Tại trạm của B, thực hiện phép nối cuối cùng: $A' \bowtie B$.

- Phép bán nối đặc biệt hiệu quả khi chỉ cần một số ít bộ của quan hệ lớn tham gia vào phép nối, và chi phí gửi các thuộc tính nối cộng với các bộ đã được lọc nhỏ hơn chi phí gửi toàn bộ một quan hệ.

3.2.2. Chiến lược truyền dữ liệu

- **Data Shipping (Gửi dữ liệu):** Di chuyển dữ liệu đến nơi tính toán.
 - **Ship-to-one:** Gửi tất cả các mảnh dữ liệu cần thiết từ các trạm khác nhau đến một trạm duy nhất để xử lý tập trung. Đơn giản nhưng có thể tạo ra điểm nóng (hot spot) tại trạm xử lý.
 - **Ship-to-all (Broadcast):** Gửi một quan hệ (thường là quan hệ nhỏ hơn) đến tất cả các trạm có chứa mảnh của quan hệ kia. Các phép nối được thực hiện song song tại nhiều trạm.
- **Query Shipping (Gửi truy vấn):** Di chuyển các phép toán đến nơi có dữ liệu. Kỹ thuật này tận dụng tối đa tính cục bộ của dữ liệu, giảm thiểu truyền thông. Phép bán nối là một ví dụ điển hình của chiến lược này.

3.2.3. Định vị truy vấn và dữ liệu

Bộ tối ưu hóa sử dụng thông tin từ danh mục phân mảnh không chỉ để cắt tỉa (pruning) mà còn để ra quyết định chiến lược. Nếu một phép nối diễn ra giữa hai mảnh dữ liệu cùng được lưu trữ tại một trạm (ví dụ NHANVIEN_HN và PHONGBAN_HN đều ở trạm Hà Nội), kế hoạch tối ưu gần như chắc chắn sẽ là thực hiện phép nối đó ngay tại trạm Hà Nội để triệt tiêu hoàn toàn chi phí mạng cho phép toán đó.

3.3. Lập kế hoạch truy vấn (Query Planning)

Giai đoạn này tổng hợp các kỹ thuật trên để xây dựng và chọn ra một Kế hoạch Thực thi Truy vấn (Query Execution Plan - QEP) hoàn chỉnh và tối ưu nhất.

3.3.1. Biểu đồ truy vấn (Query Tree / QEP)

QEP là một phiên bản chi tiết hóa của cây đại số quan hệ. Nó là một cây được chú thích, trong đó:

- Mỗi nút là một toán tử vật lý cụ thể (ví dụ: Hash Join, Table Scan, Index Scan, Sort).
- Các cạnh biểu diễn luồng dữ liệu di chuyển giữa các toán tử.
- Các chú thích có thể bao gồm thông tin về thuật toán được sử dụng, vị trí thực thi, và chi phí ước tính.

3.3.2. Cost-based vs. Rule-based Optimization

- **Tối ưu hóa dựa trên luật (Rule-based Optimization - RBO):** Sử dụng một tập các quy tắc heuristic cố định (như các luật ở mục 4.1) để biến đổi cây truy vấn. Cách tiếp cận này nhanh, dễ dự đoán nhưng thường không tìm ra được kế hoạch tốt nhất vì nó không xem xét đến các đặc điểm của dữ liệu như kích thước hay sự phân bố.
- **Tối ưu hóa dựa trên chi phí (Cost-based Optimization - CBO):** Đây là phương pháp hiện đại và chiếm ưu thế.
 - **Mô hình chi phí (Cost Model):** Trái tim của CBO. Chi phí của một kế hoạch được tính toán dựa trên một công thức kết hợp nhiều yếu tố: $Cost_{total} = w_1 \times Cost_{CPU} + w_2 \times Cost_{I/O} + w_3 \times Cost_{Network}$. Trong CSDLPT, trọng số w_3 của chi phí mạng thường là lớn nhất.
 - **Thống kê dữ liệu (Data Statistics):** Để mô hình chi phí hoạt động chính xác, DDBMS phải thu thập và duy trì các thông tin thống kê về dữ liệu trong danh mục của nó, ví dụ: số bộ trong mỗi quan hệ/mảnh, số trang đĩa, số lượng giá trị phân biệt của một thuộc tính (dùng để ước tính tính chọn lọc - selectivity), và biểu đồ phân bố giá trị (histograms).
 - **Ước lượng lực lượng (Cardinality Estimation):** Bộ tối ưu hóa sử dụng các thông tin thống kê để ước tính số bộ sẽ được tạo ra sau mỗi phép toán. Ước lượng càng chính xác, việc lựa chọn kế hoạch tối ưu càng đáng tin cậy.

3.3.3. Quy hoạch động trong tối ưu hóa (Dynamic Programming)

Để tránh phải duyệt qua toàn bộ không gian tìm kiếm khổng lồ của các kế hoạch khả thi (đặc biệt là thứ tự nối), các bộ tối ưu hóa CBO cổ điển (như trong System R) thường sử dụng thuật toán dựa trên quy hoạch động.

- **Nguyên lý:** Thuật toán xây dựng kế hoạch tối ưu một cách từ dưới lên.
 - Vòng 1:** Tìm kế hoạch truy cập tối ưu cho từng quan hệ đơn lẻ (ví dụ: dùng full table scan hay index scan).
 - Vòng 2:** Tìm kế hoạch tối ưu để nối từng cặp quan hệ, bằng cách xét tất cả các cách kết hợp kế hoạch từ Vòng 1.
 - Vòng k:** Tìm kế hoạch tối ưu cho các nhóm k quan hệ bằng cách kết hợp các kế hoạch tối ưu cho nhóm k-1 quan hệ với một quan hệ đơn lẻ. Tại mỗi bước, thuật toán chỉ giữ lại kế hoạch có chi phí thấp nhất cho mỗi tập hợp con các quan hệ.
- Quá trình này đảm bảo tìm ra kế hoạch nối tối ưu (dựa trên mô hình chi phí), nhưng độ phức tạp của nó tăng theo cấp số nhân với số lượng quan hệ, do đó chỉ phù hợp cho các truy vấn có số lượng phép nối vừa phải.

3.4. Ví dụ minh họa và đánh giá hiệu năng

Để tổng hợp các khái niệm trên, chúng ta sẽ phân tích một truy vấn cụ thể và so sánh chi phí của các kế hoạch thực thi khác nhau.

- **Bối cảnh:**

- **Lược đồ:** NHANVIEN(MaNV, TenNV, ChucVu, MaPB) và PHONGBAN(MaPB, TenPB, DiaDiem).
- **Phân mảnh:**
 - NHANVIEN: phân mảnh ngang thành NV_HN (MaPB < 'P03', lưu tại Trạm 1) và NV_HCM (MaPB >= 'P03', lưu tại Trạm 2).
 - PHONGBAN: phân mảnh ngang thành PB_HN (DiaDiem = 'Hà Nội', lưu tại Trạm 1) và PB_HCM (DiaDiem = 'TP.HCM', lưu tại Trạm 2).
- **Thông kê (giả định):**
 - NV_HN: 10,000 bộ; NV_HCM: 15,000 bộ.
 - PB_HN: 10 bộ; PB_HCM: 15 bộ.
 - sizeof(bộ NHANVIEN) = 100 bytes; sizeof(bộ PHONGBAN) = 50 bytes.
- **Truy vấn:** Tìm tên tất cả các nhân viên làm việc tại 'TP.HCM'.
SELECT TenNV FROM NHANVIEN, PHONGBAN WHERE NHANVIEN.MaPB = PHONGBAN.MaPB AND PHONGBAN.DiaDiem = 'TP.HCM'; Truy vấn này được gửi từ Trạm 1 (Hà Nội).

- **Phân tích và tối ưu hóa logic:**

- Đẩy phép chọn $\sigma_{\{DiaDiem='TP.HCM'\}}$ xuống.
- Áp dụng luật phân rã và cắt tia:
 - Toàn bộ nhánh liên quan đến PB_HN bị loại bỏ vì DiaDiem = 'TP.HCM' mâu thuẫn với điều kiện phân mảnh của nó.
 - Phép nối với NV_HN cũng có thể bị loại bỏ nếu hệ thống biết rằng không có phòng ban nào ở Hà Nội có mã MaPB >= 'P03'. Giả sử hệ thống không có thông tin này, truy vấn sau khi rút gọn là: $(NV_{HN} \cup NV_{HCM}) \bowtie PB_{HCM} \bowtie (NV_{HN} \bowtie PB_{HCM}) \cup (NV_{HCM} \bowtie PB_{HCM})$

- **So sánh các kế hoạch thực thi vật lý:**

- **Kế hoạch 1: Hướng tiếp cận "ngây thơ" (Ship-to-one)**

Chuyển toàn bộ NV_HCM (15,000 bộ) và PB_HCM (15 bộ) về Trạm 1 (nơi gửi truy vấn).

Thực hiện 2 phép nối tại Trạm 1: $NV_{HN} \bowtie PB_{HCM}$ và $NV_{HCM} \bowtie PB_{HCM}$.

Hợp kết quả.

Chi phí truyền thông: $Cost(ship\ NV_{HCM}) + Cost(ship\ PB_{HCM}) = (15,000 * 100\ bytes) + (15 * 50\ bytes) \approx 1.5\ MB$.

- **Kế hoạch 2: Tối ưu hóa bằng cách tận dụng tính cục bộ**

Tại Trạm 1, gửi PB_HCM (nhỏ) đến Trạm 2 để thực hiện phép nối tại đó. (Sai lầm, vì PB_HCM đã ở Trạm 2).

Kế hoạch đúng: Nhận thấy NV_HCM và PB_HCM cùng ở Trạm 2.

Bước 1: Tại Trạm 2, thực hiện phép nối cục bộ: $R2 = NV_HCM \Join PB_HCM$.

Bước 2: Tại Trạm 1, gửi PB_HCM đến để nối với NV_HN . (Sai lầm, nên gửi NV_HN đến trạm 2).

Kế hoạch tốt hơn:

- Tại Trạm 2, tính $R2 = NV_HCM \Join PB_HCM$. Giả sử có 15,000 nhân viên ở TP.HCM.
- Gửi PB_HCM đến Trạm 1.
- Tại Trạm 1, tính $R1 = NV_HN \Join PB_HCM$. Kết quả này sẽ rỗng (vì mã phòng ban không khớp).
- Gửi kết quả $R2$ từ Trạm 2 về Trạm 1.
- **Chi phí truyền thông:** $Cost(ship\ PB_HCM) + Cost(ship\ R2) \approx (15 * 50\ bytes) + (15,000 * 100\ bytes) \approx 1.5\ MB$.

○ **Kế hoạch 3: Tối ưu hóa bằng Semi-Join**

Nhận thấy phép nối $NV_HN \Join PB_HCM$ có thể không cần thiết.

Bước 1: Tại Trạm 2, tính $R2 = NV_HCM \Join PB_HCM$.

Bước 2: Gửi danh sách $MaPB$ từ PB_HCM (chỉ 15 giá trị) đến Trạm 1.

Bước 3: Tại Trạm 1, thực hiện phép bán nối: $R1' = NV_HN \ltimes \Pi_{\{MaPB\}}(PB_{\{HCM\}})$. Kết quả $R1'$ sẽ rỗng.

Bước 4: Chỉ cần gửi kết quả $R2$ từ Trạm 2 về Trạm 1.

Chi phí truyền thông: $Cost(ship\ \Pi_{\{MaPB\}}(PB_{\{HCM\}})) + Cost(ship\ R2) \approx (15 * 4\ bytes) + (15,000 * 100\ bytes) \approx 1.5\ MB$.

- **Phân tích lại ví dụ:** Ví dụ trên chưa thể hiện rõ ưu điểm. Hãy đổi truy vấn thành $SELECT\ TenNV\ FROM\ NHANVIEN, PHONGBAN\ WHERE\ NHANVIEN.MaPB = PHONGBAN.MaPB\ AND\ TenPB = 'Marketing'$; và giả sử phòng Marketing ở Trạm 2.

- **Kế hoạch Naïve:** Gửi toàn bộ NV_HN và NV_HCM đến Trạm 2. Chi phí = $Cost(ship\ NV_HN) = 10,000 * 100\ bytes = 1\ MB$.
- **Kế hoạch Semi-Join:**

Tại Trạm 2, tìm $MaPB$ của phòng Marketing. Giả sử là 'P04'.

Gửi $MaPB = 'P04'$ đến Trạm 1.

Tại Trạm 1, thực hiện $\sigma_{\{MaPB='P04'\}}(NV_HN)$. Giả sử có 50 nhân viên.

Gửi 50 nhân viên này đến Trạm 2.

Chi phí truyền thông: $Cost(ship\ MaPB) + Cost(ship\ 50\ NV) = (4\ bytes) + (50 * 100\ bytes) \approx 5\ KB$.

- **Đánh giá:** Bảng so sánh chi phí sẽ cho thấy kế hoạch Semi-Join giảm chi phí truyền thông từ 1 MB xuống chỉ còn 5 KB, một sự cải thiện đáng kể. Ví dụ này minh họa một cách trực quan sức mạnh của các kỹ thuật tối ưu hóa trong việc lựa chọn một chiến lược thực thi thông minh, giúp tiết kiệm tài nguyên và tăng tốc độ phản hồi của hệ thống.

CHƯƠNG 4

CÁC CHIẾN LƯỢC HIỆN ĐẠI TRONG TỐI ƯU HÓA TRUY VẤN

Các kỹ thuật tối ưu hóa dựa trên chi phí (CBO) cổ điển đã và đang là xương sống của các hệ quản trị CSDL trong nhiều thập kỷ. Tuy nhiên, khi các hệ thống dữ liệu ngày càng trở nên phức tạp, với khối lượng dữ liệu khổng lồ, workload đa dạng và các nguồn dữ liệu không đồng nhất, những hạn chế của phương pháp cổ điển ngày càng bộc lộ rõ rệt. Các hạn chế chính bao gồm:

- **Ước lượng sai:** Hiệu quả của CBO phụ thuộc hoàn toàn vào độ chính xác của mô hình chi phí và các số liệu thống kê. Các ước lượng về lực lượng (cardinality) có thể sai lệch nghiêm trọng do dữ liệu có sự tương quan phức tạp, thống kê lỗi thời, hoặc sự hiện diện của các hàm do người dùng định nghĩa (UDFs). Một ước lượng sai có thể dẫn đến việc lựa chọn một kế hoạch thực thi kém hiệu quả hàng chục, thậm chí hàng trăm lần.
- **Tính tĩnh:** Bộ tối ưu hóa cổ điển chọn ra kế hoạch "tốt nhất" *trước khi* thực thi và tuân thủ kế hoạch đó một cách cứng nhắc. Nó không có khả năng phản ứng lại những thay đổi hoặc những giả định sai lầm được phát hiện trong quá trình chạy.
- **Tính phức tạp và khó mở rộng:** Các mô hình chi phí và thuật toán tối ưu hóa thường được xây dựng thủ công, chứa nhiều quy tắc heuristic phức tạp, và rất khó để điều chỉnh hay mở rộng cho các kiến trúc phần cứng mới hoặc các loại dữ liệu mới.

Để giải quyết những thách thức này, cộng đồng nghiên cứu và công nghiệp đã phát triển các chiến lược tối ưu hóa hiện đại, hướng tới việc làm cho các bộ tối ưu hóa trở nên thông minh hơn, linh hoạt hơn và dễ mở rộng hơn.

4.1. Áp dụng Machine Learning (ML) trong tối ưu hóa truy vấn

Tầm nhìn của "bộ tối ưu hóa học được" (learned optimizer) là sử dụng các mô hình học máy để thay thế hoặc bổ sung cho các thành phần dựa trên heuristic, phức tạp của bộ tối ưu hóa truyền thống. Bằng cách học hỏi từ các lần thực thi truy vấn trong quá khứ, hệ thống có thể đưa ra các quyết định chính xác hơn.

4.1.1. Ước lượng lực lượng và chi phí

Đây là lĩnh vực ứng dụng ML phổ biến và thành công nhất trong tối ưu hóa truy vấn.

- **Vấn đề:** Các phương pháp truyền thống như dùng histogram để ước lượng lực lượng thường thất bại khi đối mặt với dữ liệu có nhiều chiều và các thuộc tính có sự tương quan phức tạp. Tương tự, các công thức tính chi phí thủ công không thể nắm bắt hết các sắc thái của phần cứng hiện đại (CPU cache, pipeline, SIMD).
- **Giải pháp ML:**

Thu thập dữ liệu: Hệ thống ghi lại nhật ký các truy vấn đã thực thi, bao gồm các đặc trưng của truy vấn (các toán tử, mệnh đề WHERE), kế

hoạch thực thi đã chọn, và kết quả thực tế (lực lượng thực tế, thời gian chạy thực tế).

Huấn luyện mô hình: Dữ liệu này được dùng để huấn luyện một mô hình ML. Ví dụ, một mô hình mạng nơ-ron sâu (Deep Neural Network) hoặc một mô hình cây (như XGBoost) có thể học được mối quan hệ giữa các mệnh đề trong truy vấn và lực lượng kết quả của chúng.

Dự đoán: Khi một truy vấn mới đến, thay vì dùng công thức heuristic, bộ tối ưu hóa sẽ sử dụng mô hình đã được huấn luyện để đưa ra dự đoán về lực lượng hoặc chi phí. Các mô hình như MSCN (Multi-Set Convolutional Network) đã chứng tỏ khả năng đưa ra ước lượng chính xác hơn nhiều lần so với các hệ thống CSDL thương mại hàng đầu.

4.1.2. Lựa chọn kế hoạch (Plan Selection)

- **Vấn đề:** Không gian tìm kiếm các kế hoạch thực thi là cực kỳ lớn. Thuật toán quy hoạch động, tuy tối ưu, nhưng trở nên quá chậm khi số lượng phép nối tăng lên.
- **Giải pháp ML (Học tăng cường - Reinforcement Learning):**
 - Tối ưu hóa truy vấn có thể được mô hình hóa như một trò chơi tuần tự. "Tác nhân" (agent) RL sẽ học cách xây dựng một kế hoạch truy vấn từng bước một (ví dụ: chọn cặp quan hệ nào để nối tiếp theo).
 - Sau khi xây dựng xong một kế hoạch và thực thi nó, "phần thưởng" (reward) sẽ được tính toán dựa trên thời gian thực thi (thời gian càng ngắn, phần thưởng càng lớn).
 - Qua hàng ngàn lần "chơi", tác nhân sẽ học được một **chính sách (policy)** để xây dựng các kế hoạch tốt một cách nhanh chóng mà không cần phải duyệt qua toàn bộ không gian tìm kiếm.

4.1.3. Thách thức và hạn chế

- **Chi phí huấn luyện:** Yêu cầu một khối lượng lớn dữ liệu huấn luyện và tài nguyên tính toán để xây dựng mô hình.
- **Vấn đề "Khởi động lạnh" (Cold Start):** Mô hình sẽ hoạt động kém hiệu quả trên một CSDL mới hoặc với các mẫu truy vấn chưa từng thấy trước đây.
- **Tính ổn định và khả năng giải thích:** Các mô hình ML phức tạp có thể hoạt động như một "hộp đen", gây khó khăn cho việc gỡ lỗi khi chúng đưa ra một quyết định tồi.

4.2. Xử lý truy vấn thích nghi (Adaptive Query Processing - AQP)

Nguyên tắc cốt lõi của AQP là: "Lên kế hoạch, nhưng hãy kiểm chứng". Thay vì tin tưởng tuyệt đối vào kế hoạch ban đầu, AQP liên tục theo dõi quá trình thực thi và điều chỉnh kế hoạch một cách linh hoạt nếu thực tế khác biệt đáng kể so với ước tính.

4.2.1. Các kỹ thuật AQP phổ biến

Lựa chọn Join thích nghi (Adaptive Join Selection):

- Hệ thống bắt đầu thực thi một toán tử join linh hoạt. Ví dụ, nó bắt đầu xây dựng bảng băm cho một Hash Join. Tuy nhiên, trong quá trình xây dựng, nó phát hiện ra rằng quan hệ bên trong nhỏ hơn nhiều so với ước tính ban đầu, khiến cho Nested-Loop Join trở nên hiệu quả hơn. Ngay lập tức, hệ thống có thể hủy bỏ việc xây dựng bảng băm và chuyển sang thực thi Nested-Loop Join. Kỹ thuật này được sử dụng trong Microsoft SQL Server.

Tối ưu hóa lại giữa chừng (Mid-query Re-optimization):

- Kế hoạch thực thi được chèn vào các điểm kiểm tra (checkpoints). Tại các điểm này, hệ thống sẽ so sánh lực lượng thực tế của dòng dữ liệu với ước tính ban đầu.
- Nếu sự chênh lệch vượt quá một ngưỡng nào đó, hệ thống sẽ tạm dừng truy vấn, sử dụng con số lực lượng chính xác vừa đo được để tối ưu hóa lại phần còn lại của kế hoạch, sau đó tiếp tục thực thi.

Phản hồi kế hoạch (Plan Feedback):

- Đây là một hình thức học hỏi giữa các lần thực thi. Sau khi một truy vấn hoàn tất, hệ thống so sánh các ước tính và kết quả thực tế.
- Nếu phát hiện một ước tính sai lệch lớn, nó sẽ lưu lại thông tin điều chỉnh này. Trong lần tiếp theo khi một truy vấn tương tự được chạy, bộ tối ưu hóa sẽ sử dụng thông tin đã điều chỉnh để tạo ra một kế hoạch ban đầu tốt hơn.

AQP giúp hệ thống trở nên mạnh mẽ và dễ dự đoán hơn, đặc biệt hiệu quả trong việc xử lý các truy vấn đột xuất hoặc khi thống kê dữ liệu không còn chính xác.

4.3. Hệ thống tối ưu hóa tự động và mở rộng

Động lực đằng sau các hệ thống này là nhu cầu truy vấn trên nhiều nguồn dữ liệu đa dạng (hồ dữ liệu, CSDL NoSQL, dòng dữ liệu thời gian thực). Các bộ tối ưu hóa truyền thống, vốn được thiết kế nguyên khối và gắn chặt với một engine CSDL cụ thể, không thể đáp ứng được yêu cầu này. Xu hướng hiện đại là xây dựng các bộ tối ưu hóa dưới dạng framework: module hóa, có thể mở rộng, và không phụ thuộc vào engine lưu trữ.

4.3.1. Apache Calcite

- **Vai trò:** Calcite không phải là một CSDL hoàn chỉnh mà là một framework nền tảng để xây dựng các bộ xử lý và tối ưu hóa truy vấn. Nó được ví như một "hộp công cụ" cho các nhà phát triển hệ thống dữ liệu.
- **Kiến trúc:** Cung cấp các thành phần có thể tái sử dụng:
 - Bộ phân tích cú pháp SQL tiêu chuẩn.
 - Một thư viện phong phú các luật biến đổi đại số quan hệ.
 - Một bộ tối ưu hóa dựa trên chi phí có thể cắm-và-chạy (pluggable).

- **Tính mở rộng:** Các nhà phát triển có thể dễ dàng tích hợp một nguồn dữ liệu mới bằng cách viết một "adapter", định nghĩa các luật tối ưu hóa riêng, và tùy chỉnh mô hình chi phí.
- **Ứng dụng:** Sự thành công của Calcite được chứng minh qua việc nó trở thành bộ não của hàng loạt hệ thống dữ liệu hàng đầu như Apache Hive, Apache Flink, Apache Druid, và Google BigQuery.

4.3.2. Presto / Trino

- **Vai trò:** Là một engine truy vấn SQL phân tán, hiệu năng cao, được thiết kế để truy vấn dữ liệu lớn tại chỗ (in-situ), tức là không cần phải nạp dữ liệu vào một hệ thống lưu trữ riêng.
- **Tối ưu hóa hiện đại trong Presto/Trino:**
 - **Kiến trúc Connector:** Cho phép Presto kết nối và truy vấn dữ liệu từ nhiều hệ thống khác nhau (HDFS, Amazon S3, MySQL, Kafka,...) trong cùng một câu lệnh SQL.
 - **Tối ưu hóa dựa trên chi phí tính vi:** Sử dụng các thông tin thống kê do các connector cung cấp để đưa ra các quyết định tối ưu về thứ tự nối, chiến lược phân phối dữ liệu (broadcast join hay partitioned join), và đẩy các phép toán tính toán xuống hệ thống nguồn (predicate pushdown).
 - **Lọc động (Dynamic Filtering):** Một kỹ thuật tối ưu hóa tại thời gian chạy. Trong một phép nối, sau khi xây dựng xong bảng băm ở phía build, Presto sẽ thu thập các giá trị của khóa nối và gửi chúng như một bộ lọc đến phía probe. Điều này giúp phía probe đọc ít dữ liệu hơn đáng kể từ nguồn, tăng tốc độ truy vấn một cách rõ rệt. Đây cũng là một dạng của xử lý thích nghi.

Sự trỗi dậy của các framework như Calcite và các engine như Presto/Trino cho thấy tương lai của tối ưu hóa truy vấn nằm ở các giải pháp linh hoạt, có khả năng thích ứng với bối cảnh dữ liệu không ngừng biến đổi và ngày càng đa dạng.

CHƯƠNG 5

ỨNG DỤNG VÀ THÁCH THỨC THỰC TẾ

5.1. Ứng dụng trong các hệ thống thực tế

Hiệu năng và khả năng mở rộng do các hệ CSDL phân tán và bộ tối ưu hóa truy vấn của chúng mang lại là nền tảng cho sự thành công của nhiều lĩnh vực công nghiệp quan trọng.

5.1.1. Thương mại điện tử và bán lẻ (*E-commerce and Retail*)

- **Nhu cầu:** Các nền tảng như Amazon, Alibaba, hay Shopee phải phục vụ hàng trăm triệu người dùng trên toàn cầu, xử lý hàng triệu giao dịch mỗi ngày. Lưu lượng truy cập có thể tăng đột biến vào các dịp lễ (Black Friday, 11/11), đòi hỏi hệ thống phải có khả năng co giãn linh hoạt và hoạt động 24/7. Dữ liệu về sản phẩm, khách hàng, đơn hàng, và kho hàng được phân tán theo địa lý để giảm độ trễ và tăng tốc độ truy cập cho người dùng cuối.
- **Vai trò của tối ưu hóa truy vấn:**
 - **Truy vấn tương tác thời gian thực:** Khi một người dùng tìm kiếm sản phẩm, hệ thống phải thực hiện một câu truy vấn nối (join) phức tạp qua các bảng Sản_phẩm, Kho_hàng, Nhà_cung_cấp, và Đánh_giá, vốn được phân mảnh và sao chép trên nhiều khu vực. Một bộ tối ưu hóa hiệu quả phải chọn được kế hoạch thực thi trong vài mili-giây để trả về kết quả, nếu không sẽ ảnh hưởng trực tiếp đến trải nghiệm mua sắm và doanh thu.
 - **Phân tích hoạt động kinh doanh:** Các nhà phân tích cần chạy các câu truy vấn phức tạp để theo dõi doanh số theo thời gian thực, quản lý chuỗi cung ứng, và cá nhân hóa các gợi ý sản phẩm. Các truy vấn này có thể phải quét qua hàng terabytes dữ liệu lịch sử và nối với dữ liệu thời gian thực. Bộ tối ưu hóa phải có khả năng tạo ra các kế hoạch xử lý song song hiệu quả để cung cấp thông tin chi tiết một cách nhanh chóng mà không làm ảnh hưởng đến hiệu năng của các giao dịch chính.

5.1.2. Tài chính - Ngân hàng (*Finance and Banking*)

- **Nhu cầu:** Các hệ thống tài chính yêu cầu khả năng xử lý giao dịch thông lượng cao (high-throughput), đảm bảo tính nhất quán mạnh (ACID), hoạt động trên quy mô toàn cầu, và có độ sẵn sàng cũng như bảo mật cực kỳ cao.
- **Vai trò của tối ưu hóa truy vấn:**
 - **Phát hiện gian lận thời gian thực:** Để phê duyệt một giao dịch thẻ tín dụng, hệ thống có thể cần phải phân tích lịch sử giao dịch gần đây của khách hàng, so sánh với các mẫu hành vi thông thường và các quy tắc gian lận trong vòng chưa đầy 100 mili-giây. Đây là một bài

toán tối ưu hóa truy vấn cực kỳ khắt khe, đòi hỏi các phép nối và tổng hợp có độ trễ cực thấp trên dữ liệu chuỗi thời gian được phân tán.

- **Phân tích rủi ro:** Các ngân hàng đầu tư thường xuyên chạy các mô phỏng Monte Carlo hoặc các truy vấn phân tích phức tạp trên danh mục đầu tư khổng lồ để tính toán rủi ro thị trường (Value at Risk - VaR). Bộ tối ưu hóa phải có khả năng phân phối các tác vụ tính toán nặng này trên một cụm máy tính lớn để có được kết quả trong khung thời gian cho phép.

5.1.3. Phân tích dữ liệu lớn và khoa học dữ liệu

- **Nhu cầu:** Đây là lĩnh vực mà CSDLPT và tối ưu hóa truy vấn tỏa sáng nhất. Dữ liệu được lưu trữ trong các hồ dữ liệu (data lakes) trên các hệ thống lưu trữ phân tán như HDFS hoặc Amazon S3, với các định dạng đa dạng (Parquet, ORC, JSON).
- **Vai trò của tối ưu hóa truy vấn:**
 - **Truy vấn liên nguồn (Federated Queries):** Các công cụ hiện đại như Presto/Trino và Spark SQL cho phép các nhà phân tích viết một câu lệnh SQL duy nhất để nối dữ liệu từ một kho dữ liệu Hive, một CSDL quan hệ PostgreSQL và một dòng dữ liệu từ Kafka. Bộ tối ưu hóa truy vấn phân tán đóng vai trò trung tâm, thực hiện nhiệm vụ cực kỳ phức tạp là tạo ra một kế hoạch thực thi thống nhất, đẩy các phép tính xuống hệ thống nguồn một cách thông minh, và tổng hợp kết quả. Đây là một ví dụ điển hình của bài toán tối ưu hóa hiện đại.

5.2. Các thách thức thực tế

Mặc dù có nhiều ứng dụng thành công, việc triển khai CSDLPT và tối ưu hóa truy vấn trong thực tế phải đối mặt với nhiều vấn đề gai góc mà các mô hình lý thuyết thường bỏ qua.

5.2.1. Dữ liệu không đồng nhất (Data Heterogeneity)

- **Vấn đề:** Trong thực tế, dữ liệu hiếm khi nằm gọn trong một hệ CSDL quan hệ duy nhất. Chúng nằm rải rác trên nhiều hệ thống: dữ liệu có cấu trúc trong CSDL SQL, dữ liệu bán cấu trúc trong CSDL NoSQL (như MongoDB, Cassandra), và dữ liệu phi cấu trúc trong các hồ dữ liệu.
- **Thách thức cho tối ưu hóa:** Một bộ tối ưu hóa liên nguồn phải trở thành một "nhà ngoại giao" tài ba. Nó cần "nói chuyện" được với mọi hệ thống nguồn để hiểu được khả năng của chúng (ví dụ: hệ thống đó có hỗ trợ phép lọc hay không?), lấy được các thông tin thống kê (nếu có), và ước tính chi phí cho việc đẩy một phần công việc xuống hệ thống đó. Việc này đòi hỏi một kiến trúc connector linh hoạt và một hệ thống quản lý siêu dữ liệu (metadata) phức tạp.

5.2.2. Độ trễ mạng và sự không ổn định

- **Vấn đề:** Các mô hình chi phí thường giả định mạng ổn định. Mạng trong thực tế, đặc biệt là mạng Internet công cộng hoặc kết nối giữa các trung tâm dữ liệu xuyên lục địa, có độ trễ biến thiên, jitter, và tỷ lệ mất gói tin không thể lường trước.
- **Thách thức cho tối ưu hóa:** Một kế hoạch được cho là tối ưu dựa trên độ trễ trung bình có thể trở nên rất tệ nếu mạng bị nghẽn đột ngột. Đây chính là động lực lớn nhất cho sự phát triển của Xử lý truy vấn thích nghi (AQP), cho phép hệ thống phản ứng lại các biến động tại thời gian chạy. Thách thức này cũng thúc đẩy sự ra đời của các bộ tối ưu hóa "nhận biết về mạng" (network-aware), có thể chủ động chọn một kế hoạch tốn nhiều CPU hơn tại các trạm cục bộ để tránh phải gửi dữ liệu qua một liên kết mạng đang bị quá tải.

5.2.3. Tính chịu lỗi và xử lý lỗi (Fault Tolerance)

- **Vấn đề:** Với một hệ thống có hàng ngàn nút, lỗi phần cứng hoặc phần mềm không phải là trường hợp ngoại lệ, mà là một phần của hoạt động bình thường. Một truy vấn phân tích kéo dài hàng giờ phải có khả năng "sống sót" qua các sự cố này.
- **Thách thức cho Tối ưu hóa:** Bộ tối ưu hóa và engine thực thi phải được thiết kế để chịu lỗi. Kế hoạch thực thi cần tích hợp sẵn cơ chế đánh dấu điểm kiểm tra (checkpointing), lưu các kết quả trung gian vào một hệ thống lưu trữ bền vững (như S3). Khi một nút bị lỗi, bộ lập lịch có thể khởi động lại các tác vụ thất bại trên một nút khác, bắt đầu từ điểm kiểm tra gần nhất. Việc này tạo ra một chi phí phụ (overhead) mà bộ tối ưu hóa phải tính đến khi xây dựng kế hoạch.

5.2.4. Dữ liệu bị xiên (Data Skew)

- **Vấn đề:** Giả định rằng dữ liệu được phân bố đồng đều thường không đúng trong thực tế. Dữ liệu thường bị xiên (skewed), tức là một số giá trị khóa xuất hiện với tần suất cao hơn rất nhiều so với các giá trị khác. Ví dụ, trong một trang thương mại điện tử, một vài khách hàng doanh nghiệp ("cá voi") có thể có hàng triệu đơn hàng, trong khi phần lớn người dùng chỉ có vài đơn hàng.
- **Thách thức cho tối ưu hóa:** Một phép Hash Join tiêu chuẩn trên customer_id sẽ gửi tất cả dữ liệu của các "cá voi" này đến một nút duy nhất, gây quá tải cho nút đó trong khi các nút khác thì nhàn rỗi. Các bộ tối ưu hóa hiện đại cần các kỹ thuật để phát hiện dữ liệu xiên (dựa vào histogram) và tự động áp dụng các thuật toán join chống xiên (ví dụ: chia nhỏ khóa bị xiên và xử lý song song trên nhiều nút).

5.2.5. Quản lý tài nguyên và đa người dùng (*Resource Management and Multi-tenancy*)

- **Vấn đề:** Một cụm CSDL phân tán là tài nguyên chia sẻ cho nhiều người dùng và ứng dụng chạy đồng thời.
- **Thách thức cho tối ưu hóa:** Kế hoạch "tối ưu" có thể phụ thuộc vào tải hiện tại của hệ thống. Một kế hoạch cần nhiều bộ nhớ cho Hash Join có thể rất tốt trên một hệ thống rảnh rỗi, nhưng có thể thất bại hoặc gây tranh chấp tài nguyên trên một hệ thống đang bận rộn. Điều này dẫn đến nhu cầu về các bộ tối ưu hóa nhận biết về tài nguyên (resource-aware), không chỉ xem xét bản thân câu truy vấn mà còn cả tài nguyên CPU, bộ nhớ và I/O đang có sẵn trên toàn cụm để đưa ra quyết định.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Từ những khái niệm nền tảng về CSDLPT, các kỹ thuật tối ưu hóa cổ điển, cho đến các chiến lược hiện đại và ứng dụng thực tiễn, có thể thấy rằng tối ưu hóa truy vấn là một thành phần trung tâm, quyết định đến hiệu năng, khả năng mở rộng và tính kinh tế của các hệ thống dữ liệu quy mô lớn. Trong kỷ nguyên dữ liệu lớn, các hệ CSDL phân tán là một giải pháp kiến trúc tất yếu. Tuy nhiên, lợi ích về khả năng mở rộng và tính sẵn sàng của chúng chỉ có thể được hiện thực hóa trọn vẹn thông qua một bộ tối ưu hóa truy vấn thông minh và hiệu quả. Quá trình tối ưu hóa này là một bài toán phức tạp, được chia thành nhiều giai đoạn.

Đầu tiên, tối ưu hóa logic sử dụng các luật đại số quan hệ để biến đổi một truy vấn thành một dạng tương đương nhưng hiệu quả hơn về mặt tính toán, chủ yếu bằng cách thực hiện các phép chọn và phép chiếu càng sớm càng tốt để giảm kích thước dữ liệu trung gian. Tiếp theo, tối ưu hóa vật lý là giai đoạn cốt lõi, nơi bộ tối ưu hóa đưa ra các quyết định chiến lược dựa trên một mô hình chi phí. Các quyết định này bao gồm lựa chọn thuật toán thực thi (như Hash Join), chiến lược truyền dữ liệu, và đặc biệt là áp dụng các kỹ thuật đặc thù cho môi trường phân tán như phép bán nối (semi-join) để giảm thiểu chi phí truyền thông mạng – yếu tố thường là "nút cổ chai" của hệ thống.

Tuy nhiên những hạn chế của phương pháp cổ điển và giới thiệu các chiến lược hiện đại đang định hình lại lĩnh vực này. Việc áp dụng Học máy (Machine Learning) hứa hẹn mang lại các ước lượng chi phí và lực lượng chính xác hơn, trong khi Xử lý truy vấn thích nghi (Adaptive Query Processing) cho phép hệ thống tự điều chỉnh kế hoạch thực thi để đối phó với các giả định sai lầm ban đầu. Sự trỗi dậy của các framework mở như Apache Calcite và các engine truy vấn như Presto/Trino đang thúc đẩy một kỷ nguyên mới của các bộ tối ưu hóa linh hoạt, có khả năng truy vấn trên các nguồn dữ liệu không đồng nhất.

Báo cáo đã kết nối các lý thuyết này với thực tiễn, cho thấy vai trò không thể thiếu của tối ưu hóa truy vấn trong các ứng dụng từ thương mại điện tử, tài chính cho đến phân tích dữ liệu lớn, đồng thời nêu bật các thách thức thực tế như dữ liệu chéo, độ trễ mạng và tính chịu lỗi.

Hạn chế của báo cáo

Mặc dù đã cố gắng bao quát một cách toàn diện, báo cáo này vẫn có một số hạn chế nhất định:

- **Phạm vi tổng quan:** Báo cáo chủ yếu mang tính chất tổng hợp và phân tích tài liệu, không bao gồm việc thực hiện các thử nghiệm thực tế (benchmark) để so sánh hiệu năng của các kỹ thuật hay các hệ thống CSDL khác nhau. Các ví dụ được đưa ra chỉ mang tính chất minh họa cho khái niệm.
- **Tập trung vào mô hình quan hệ:** Trọng tâm chính của báo cáo là tối ưu hóa truy vấn trong các hệ CSDL quan hệ phân tán. Các thách thức tối ưu hóa trong các mô hình dữ liệu khác như CSDL đồ thị (Graph DBs), CSDL tài liệu (Document DBs), hay CSDL chuỗi thời gian (Time-series DBs) là những lĩnh vực rộng lớn và chưa được đi sâu phân tích.

- **Các chủ đề liên quan:** Để giữ được sự tập trung, báo cáo đã không phân tích chi tiết các lĩnh vực có liên quan mật thiết đến tối ưu hóa truy vấn như quản lý giao dịch và kiểm soát đồng thời phân tán, hay các chiến lược bảo mật và kiểm soát truy cập dữ liệu trên môi trường phân tán.
- **Tốc độ phát triển của công nghệ:** Lĩnh vực tối ưu hóa truy vấn đang phát triển rất nhanh. Báo cáo này phản ánh tình hình đến giữa năm 2025, và chắc chắn sẽ có những kỹ thuật, công nghệ mới xuất hiện ngay sau đó.

Định hướng nghiên cứu tương lai

Dựa trên các thách thức và xu hướng hiện tại, tương lai của lĩnh vực tối ưu hóa truy vấn phân tán hứa hẹn sẽ tập trung vào các hướng nghiên cứu đột phá sau:

Bộ tối ưu hóa tự động hoàn toàn (Autonomous / Self-Driving Optimizers):

Đây là tầm nhìn lớn nhất, hướng tới các hệ thống "zero-knob" không cần sự can thiệp của con người. Thay vì chỉ dùng ML cho từng tác vụ riêng lẻ, các nghiên cứu sẽ tập trung vào việc sử dụng Học tăng cường sâu (Deep Reinforcement Learning) để điều khiển toàn bộ quy trình tối ưu hóa, từ việc chọn luật biến đổi, xây dựng kế hoạch, cho đến phân bổ tài nguyên, và liên tục tự cải thiện dựa trên workload thực tế.

Tối ưu hóa cho xử lý dữ liệu thời gian thực và Streaming: Các mô hình tối ưu hóa truyền thống được thiết kế cho dữ liệu tĩnh. Với sự bùng nổ của IoT và các ứng dụng thời gian thực, thách thức lớn là phát triển các mô hình chi phí và kỹ thuật tối ưu hóa cho các truy vấn liên tục (continuous queries) trên các dòng dữ liệu vô tận. Các hướng nghiên cứu bao gồm tối ưu hóa các phép toán cửa sổ (windowed operations) và khả năng thích ứng kế hoạch khi đặc tính của dòng dữ liệu thay đổi.

Tối ưu hóa nhận biết về phần cứng và kiến trúc Serverless:

- **Tăng tốc bằng phần cứng:** Các bộ tối ưu hóa trong tương lai cần phải "nhận biết" về các tài nguyên phần cứng đa dạng như GPU, FPGA, và bộ nhớ không bay hơi (NVM). Mô hình chi phí cần phải đủ thông minh để quyết định khi nào nên chuyển một phép nối lớn sang xử lý trên GPU, dù phải chịu chi phí truyền dữ liệu.
- **Tối ưu hóa cho Serverless:** Trong kiến trúc serverless, nơi tính toán và lưu trữ được tách rời và co giãn độc lập, bài toán không chỉ là tìm kế hoạch tối ưu mà còn là quyết định cấp phát bao nhiêu tài nguyên cho một truy vấn tại một thời điểm. Đây là một dạng tối ưu hóa nhận biết về tài nguyên ở cấp độ cao hơn.

Tối ưu hóa có trách nhiệm (Responsible Optimization):

- **Tối ưu hóa xanh (Green Optimization):** Khi chi phí năng lượng của các trung tâm dữ liệu ngày càng trở thành một vấn đề lớn, một hướng đi mới là tích hợp yếu tố tiêu thụ năng lượng vào mô hình chi phí. Kế hoạch "rẻ nhất" có thể không chỉ là nhanh nhất, mà còn là kế hoạch sử dụng ít năng lượng nhất để hoàn thành, mang lại lợi ích về cả chi phí và tính bền vững.

- **Tối ưu hóa cho quyền riêng tư (Privacy-Aware Optimization):**
Tích hợp các kỹ thuật bảo vệ quyền riêng tư ngay trong quá trình tối ưu hóa. Bộ tối ưu hóa có thể tự động biến đổi một truy vấn để nó vẫn trả về kết quả đúng nhưng giảm thiểu việc truy cập vào các dữ liệu nhạy cảm, đảm bảo tuân thủ các quy định như GDPR.

Những định hướng này cho thấy lĩnh vực tối ưu hóa truy vấn vẫn còn rất nhiều không gian để phát triển, hứa hẹn sẽ tiếp tục là một trong những lĩnh vực nghiên cứu năng động và có ảnh hưởng nhất trong khoa học máy tính.

TÀI LIỆU THAM KHẢO

- [1] Özsu, M. Tamer, and Patrick Valduriez. Principles of Distributed Database Systems. Springer, 2020.
- [2] Garcia-Molina, H., Ullman, J. D., & Widom, J. Database Systems: The Complete Book. Pearson, 2019.
- [3] Abadi, D. J., et al. "The Design of the Borealis Stream Processing Engine." CIDR, 2005.
- [4] Leis, V., et al. "How good are query optimizers, really?" Proceedings of the VLDB Endowment, 2015.
- [5] Pavlo, A., et al. "Self-driving database management systems." CIDR, 2017.
- Stonebraker, M., & Hellerstein, J. M. "What goes around comes around." Readings in Database Systems, 2015.