[1] “OLAP.” [Online]. Available: https://en.wikipedia.org/wiki/Online_analytical_processing

[2] “OLTP.” [Online]. Available: https://en.wikipedia.org/wiki/Online_transaction_processing

[3] “Big Data.” [Online]. Available: https://en.wikipedia.org/wiki/Big_data

[4] “CPU Architecture Overview.” [Online]. Available: https://en.wikipedia.org/wiki/Central_processing_unit

[5] “GPU Architecture Overview.” [Online]. Available: https://en.wikipedia.org/wiki/Graphics_processing_unit

[6] DuckDB Documentation, “Introduction to DuckDB.” [Online]. Available: https://duckdb.org/docs/

[7] D. MacMillan, “Why DuckDB is Better than SQLite for Data Analysis.” [Online]. Available: https://datasciencecentral.com/why-duckdb-is-better-than-sqlite-for-data-analysis/

[8] “HeavyDB Technical Overview.” [Online]. Available: https://docs.heavy.ai/

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT VÀ TỔNG QUAN TÀI LIỆU

1.1. Cơ sở lý thuyết

1.1.1. Cơ sở dữ liệu phân tích

1.1.1.1. Định nghĩa cơ sở dữ liệu phân tích và sự khác biệt so với cơ sở dữ liệu giao dịch

Định nghĩa: Cơ sở dữ liệu phân tích (Analytical Database) là một hệ thống cơ sở dữ liệu chuyên dụng, được thiết kế để hỗ trợ các tác vụ phân tích dữ liệu phức tạp, xử lý khối lượng lớn dữ liệu lịch sử và cung cấp thông tin chi tiết phục vụ ra quyết định. Chúng thường được tối ưu hóa cho các truy vấn đọc nhiều (read-heavy), phức tạp như tổng hợp (aggregation), lọc (filtering), và kết nối bảng (join), thay vì xử lý giao dịch thời gian thực. Các hệ thống này thường áp dụng mô hình OLAP (Online Analytical Processing), tập trung vào hiệu suất phân tích dữ liệu đa chiều. [1]

Sự khác biệt với cơ sở dữ liệu giao dịch (OLTP): Cơ sở dữ liệu giao dịch (Online Transaction Processing - OLTP) được thiết kế để quản lý các giao dịch thời gian thực, chẳng hạn như thêm, sửa, xóa dữ liệu trong các ứng dụng như hệ thống bán lẻ, ngân hàng. Ngược lại, cơ sở dữ liệu phân tích tập trung vào xử lý dữ liệu lớn và phân tích dài hạn. [2]

Tiêu chí	OLTP	OLAP
Mục đích	Quản lý giao dịch	Phân tích dữ liệu
Loại truy vấn	Ngắn, đơn giản (INSERT, UPDATE)	Dài, phức tạp (SELECT, JOIN)
Khối lượng dữ liệu	Nhỏ, cập nhật thường xuyên	Lớn, chủ yếu dữ liệu lịch sử
Mô hình lưu trữ	Dạng hàng (row-based)	Dạng cột (column-based)
Ví dụ ứng dụng	Đặt hàng online	Phân tích doanh thu theo năm

Bảng 1: So sánh OLTP và OLAP

Ví dụ: Trong một hệ thống bán lẻ, OLTP ghi lại từng giao dịch mua hàng (ví dụ: "Khách hàng A mua sản phẩm X lúc 10:00"), trong khi OLAP phân tích dữ liệu lịch sử để trả lời câu hỏi "Sản phẩm nào bán chạy nhất trong quý 1?".

Kết luận: Sự khác biệt cốt lõi nằm ở mục tiêu thiết kế: OLTP ưu tiên tốc độ giao dịch và tính nhất quán (ACID), còn OLAP ưu tiên hiệu suất truy vấn và khả năng xử lý song song trên dữ liệu lớn

1.1.1.2. Tầm quan trọng của cơ sở dữ liệu phân tích trong xử lý dữ liệu lớn

Dữ liệu lớn (Big Data) được định nghĩa bởi ba đặc tính chính, thường gọi là 3Vs: khối lượng lớn (Volume), tốc độ xử lý cao (Velocity), và sự đa dạng (Variety) từ dữ liệu có cấu trúc đến phi cấu trúc. Khái niệm này phản ánh sự bùng nổ của dữ liệu trong thời đại số, từ các nguồn như mạng xã hội, cảm biến IoT, đến giao dịch trực tuyến. Trong bối cảnh đó, cơ sở dữ liệu phân tích trở thành công cụ không thể thiếu để xử lý và khai thác giá trị từ khối lượng dữ liệu khổng lồ này, vượt xa khả năng của các hệ thống truyền thống. [3]

Cơ sở dữ liệu phân tích đóng vai trò quan trọng trong Big Data nhờ khả năng vượt trội ở nhiều khía cạnh. Trước hết, chúng có thể quản lý và truy vấn dữ liệu ở quy mô terabyte hoặc thậm chí petabyte, điều mà các hệ thống OLTP truyền thống không thể thực hiện hiệu quả. Ngoài ra, nhờ tích hợp các kỹ thuật xử lý song song, cơ sở dữ liệu phân tích cung cấp tốc độ truy vấn nhanh chóng trên dữ liệu lớn. Chẳng hạn, HeavyDB tận dụng GPU để tăng tốc truy vấn lên hàng chục lần so với phương pháp thông thường. Hơn nữa, chúng hỗ trợ doanh nghiệp trích xuất thông tin chi tiết từ dữ liệu thô, giúp dự đoán xu hướng, tối ưu chuỗi cung ứng, hoặc cá nhân hóa trải nghiệm khách hàng, từ đó nâng cao khả năng ra quyết định chiến lược.

Ứng dụng thực tiễn của cơ sở dữ liệu phân tích được thể hiện rõ qua nhiều lĩnh vực. Trong kinh doanh, Walmart sử dụng hệ thống OLAP để phân tích dữ liệu bán hàng, xác định sản phẩm bán chạy và tối ưu hóa quản lý kho hàng. Trong khoa học, các hệ thống này hỗ trợ xử lý dữ liệu genomic trong y học hoặc dữ liệu khí tượng để dự báo thời tiết với độ chính xác cao. Về công nghệ, Netflix áp dụng cơ sở dữ liệu phân tích để phân tích log hệ thống, phát hiện sự cố và tối ưu hóa dịch vụ streaming, mang lại trải nghiệm tốt hơn cho người dùng.

Số liệu từ các tổ chức uy tín càng khẳng định tầm quan trọng của cơ sở dữ liệu phân tích. Theo IDC, khối lượng dữ liệu toàn cầu dự kiến đạt 175 zettabyte vào năm 2025, trong đó 60% là dữ liệu doanh nghiệp cần được phân tích để tạo ra giá trị. Gartner cũng dự báo rằng đến năm 2025, 80% tổ chức sẽ phụ thuộc vào phân tích dữ liệu để duy trì lợi thế cạnh tranh, nhấn mạnh vai trò không thể thay thế của các hệ thống này trong tương lai gần.

Tầm quan trọng cụ thể của cơ sở dữ liệu phân tích không chỉ nằm ở khả năng lưu trữ và truy vấn dữ liệu lớn, mà còn ở sự tích hợp với các công nghệ tiên tiến như trí tuệ nhân tạo (AI) và học máy (Machine Learning). Sự kết hợp này cho phép xử lý dữ liệu thời gian thực và đưa ra dự đoán chính xác hơn. Ví dụ, HeavyDB không chỉ tăng tốc truy vấn mà còn hỗ trợ chạy các mô hình học máy trực tiếp trên dữ liệu nhờ tích hợp GPU, mở ra tiềm năng mới trong phân tích dữ liệu lớn. Nhờ đó, cơ sở dữ liệu phân tích không chỉ đáp ứng nhu cầu hiện tại mà còn định hình cách các tổ chức khai thác Big Data trong tương lai.

1.1.1.3. Các phương pháp tối ưu hóa truy vấn trong cơ sở dữ liệu phân tích.

a) Tối ưu hóa lưu trữ dữ liệu

- Lưu trữ theo cột (Columnar Storage): Dữ liệu được lưu theo cột thay vì hàng giúp truy vấn nhanh hơn do chỉ đọc các cột cần thiết, đồng thời tăng hiệu quả nén dữ liệu. Các hệ thống như Google BigQuery, ClickHouse, DuckDB tận dụng kỹ thuật này để cải thiện hiệu suất truy vấn.
- Chỉ mục (Indexing): Chỉ mục giúp tăng tốc truy vấn bằng cách giảm lượng dữ liệu cần quét. Các loại chỉ mục phổ biến gồm B-Tree Index, Bitmap Index, Clustered Index, được sử dụng rộng rãi trong PostgreSQL, Oracle, SQL Server.
- Phân vùng dữ liệu (Partitioning): Chia dữ liệu thành các phần nhỏ theo phạm vi (range), danh mục (list) hoặc băm (hash) giúp hạn chế lượng dữ liệu quét khi truy vấn. Kỹ thuật này phổ biến trong PostgreSQL, Hive, Snowflake.

b) Tối ưu hóa thực thi truy vấn

- Xử lý song song (Query Parallelization): Truy vấn được chia thành nhiều tác vụ nhỏ và thực thi đồng thời trên nhiều lõi CPU hoặc máy chủ, giúp tăng tốc xử lý. Các hệ thống Apache Spark SQL, Snowflake, Redshift sử dụng phương pháp này.
- Cache kết quả truy vấn (Query Caching): Lưu kết quả truy vấn vào bộ nhớ cache giúp tăng tốc độ phản hồi cho các truy vấn lặp lại. Hệ thống như MemSQL, Apache Druid, HeavyDB áp dụng kỹ thuật này để tối ưu hiệu suất.

- Tối ưu hóa kế hoạch thực thi (Query Execution Plan Optimization): Hệ quản trị cơ sở dữ liệu chọn kế hoạch thực thi tối ưu bằng cách viết lại truy vấn (Rewriting Query), tối ưu JOIN (Join Optimization), đẩy bộ lọc xuống sớm (Pushdown Predicates). Các hệ thống như PostgreSQL, MySQL, Oracle tận dụng phương pháp này để cải thiện hiệu suất.

c) Tận dụng GPU để tăng tốc truy vấn

- Xử lý song song trên GPU (GPU-Accelerated Query Processing): GPU có hàng nghìn lõi giúp xử lý các phép toán truy vấn nhanh hơn nhiều so với CPU. Hệ thống như HeavyDB (OmniSci), RAPIDS cuDF, BlazingSQL tận dụng GPU để tăng tốc phân tích dữ liệu lớn.
- Columnar Execution trên GPU: GPU hoạt động hiệu quả hơn với dữ liệu theo cột, cho phép xử lý nhanh hơn các phép toán quét, tổng hợp. Hệ thống như HeavyDB, ClickHouse sử dụng kỹ thuật này để khai thác tối đa hiệu năng của GPU.

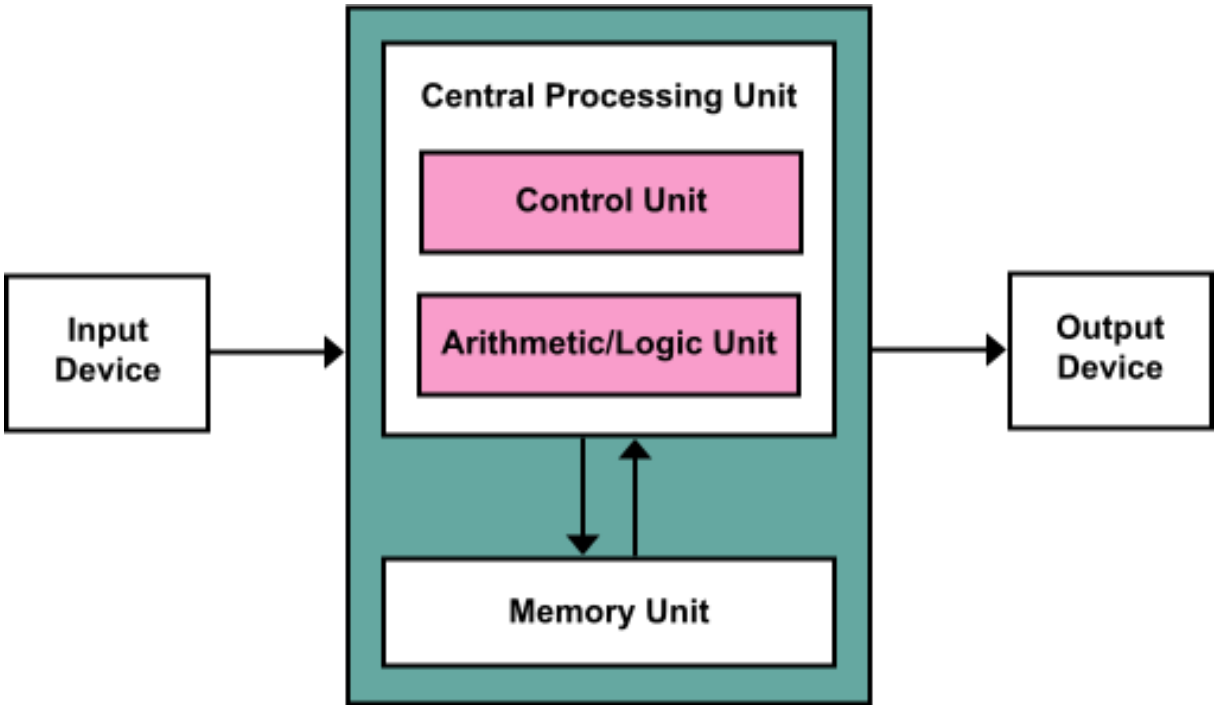
d) Tối ưu hóa chi phí và hiệu suất

- Lưu trữ theo nhiều cấp độ (Multi-Tier Storage): Dữ liệu được lưu trong RAM hoặc SSD (Hot Storage) cho truy vấn nhanh, HDD hoặc Object Storage (Warm Storage) cho dữ liệu ít dùng hơn, và Cold Storage (AWS Glacier, S3) cho dữ liệu lâu dài. Kỹ thuật này phổ biến trong BigQuery, Redshift, Snowflake.
- Auto Scaling và Elastic Query Execution: Các hệ thống như Snowflake có thể tự động tăng/giảm tài nguyên theo nhu cầu, đảm bảo hiệu suất tốt mà vẫn tối ưu chi phí vận hành.

1.1.2. So sánh kiến trúc CPU và GPU trong xử lý truy vấn

1.1.2.1. Nguyên lý hoạt động của CPU trong xử lý truy vấn dữ liệu

CPU viết tắt của chữ Central Processing Unit (tiếng Anh), tạm dịch là Bộ xử lý trung tâm, là mạch điện tử thực hiện các câu lệnh của chương trình máy tính bằng cách thực hiện các phép tính số học, logic, so sánh và các hoạt động nhập/xuất dữ liệu (Input/Output) cơ bản từ mã lệnh được định sẵn trong một máy tính. [4]



Hình 1: Kiến trúc của CPU

CPU đóng vai trò cốt lõi trong việc thực thi các truy vấn dữ liệu bằng cách xử lý tuần tự hoặc song song các lệnh từ phần mềm cơ sở dữ liệu. Nguyên lý hoạt động của CPU dựa trên chu trình "lấy lệnh - giải mã - thực thi" (fetch-decode-execute). Đầu tiên, CPU lấy lệnh từ bộ nhớ chính (RAM) thông qua thanh ghi, sau đó giải mã lệnh thành các thao tác cụ thể như đọc dữ liệu, thực hiện phép tính, hoặc ghi kết quả. Với truy vấn dữ liệu, chẳng hạn "SELECT * FROM employees WHERE salary > 50000", CPU sẽ tuần tự xử lý từng bước: đọc bảng, so sánh giá trị cột "salary", và trả về kết quả phù hợp.

Sức mạnh của CPU trong xử lý truy vấn đến từ số lượng nhân xử lý (cores) và khả năng thực thi đa luồng (multithreading). Mỗi nhân hoạt động như một đơn vị xử lý độc lập, cho phép CPU chia nhỏ công việc thành nhiều phần và xử lý đồng thời nếu truy vấn có thể song song hóa. Ví dụ, DuckDB tận dụng đa nhân CPU để thực hiện vectorized execution, trong đó dữ liệu được chia thành các khối (vectors) và xử lý cùng lúc trên nhiều nhân. Tuy nhiên, với số lượng nhân giới hạn (thường từ 4 đến 32 trong CPU hiện đại), hiệu suất song song của CPU không thể so sánh với GPU khi xử lý dữ liệu rất lớn.

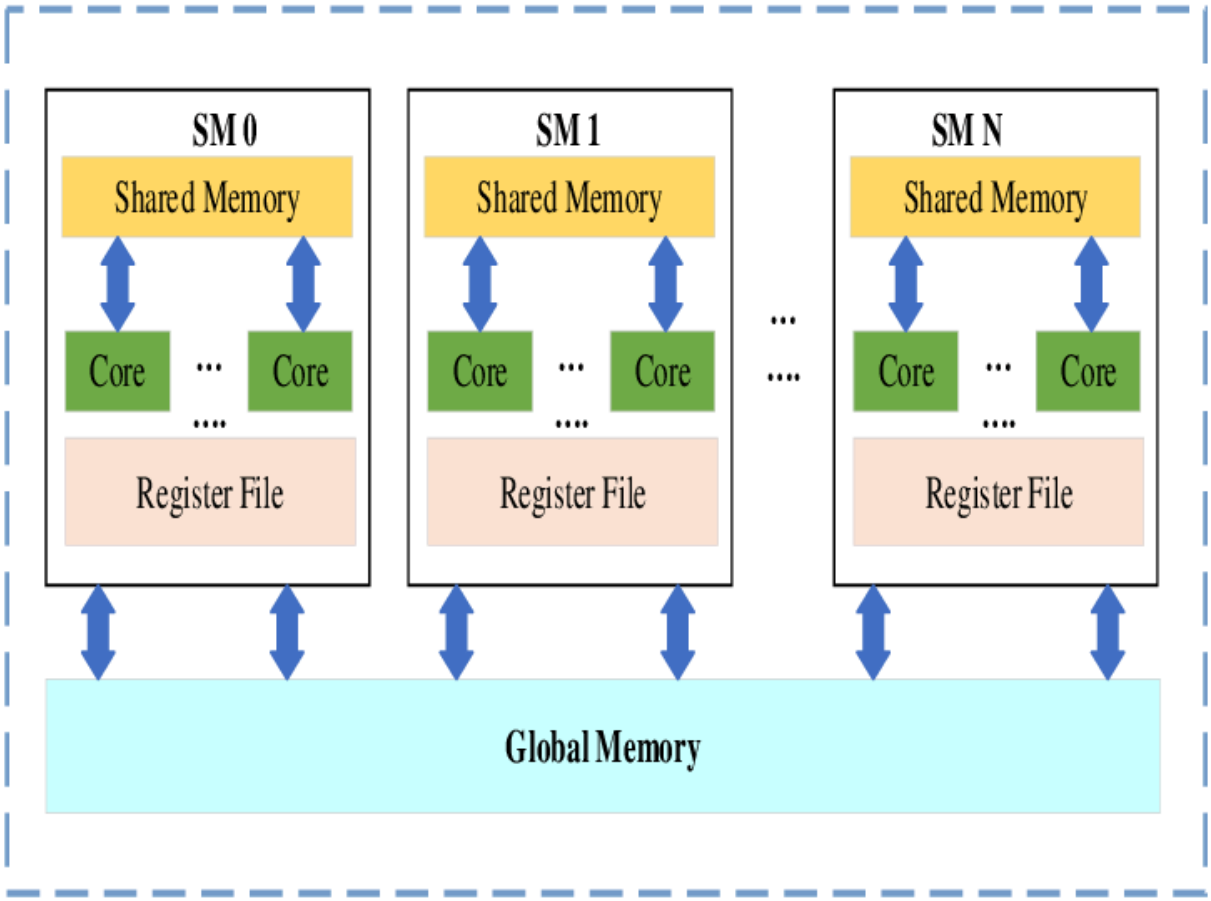
Bộ nhớ đệm (cache) là một thành phần quan trọng giúp CPU tăng tốc xử lý truy vấn. Cache lưu trữ dữ liệu thường dùng ngay trên chip CPU, với dung lượng nhỏ (vài MB) nhưng tốc độ cực nhanh, giảm độ trễ khi truy cập dữ liệu từ RAM. Trong một truy vấn như "SELECT AVG(salary) GROUP BY department", CPU có thể lưu tạm các giá trị "salary" hoặc kết quả trung gian trong cache, tránh việc đọc lại từ bộ nhớ chính. Điều này đặc biệt hữu ích với các truy vấn cần truy xuất ngẫu nhiên hoặc lặp lại trên tập dữ liệu nhỏ, giúp giảm thời gian chờ (latency) đáng kể.

CPU cũng sử dụng đơn vị điều khiển (control unit) và đơn vị số học logic (ALU - Arithmetic Logic Unit) để thực hiện các phép tính và so sánh cần thiết cho truy vấn. Đơn vị điều khiển quản lý luồng thực thi, quyết định thứ tự xử lý các lệnh, trong khi ALU thực hiện các phép toán như cộng, trừ, hoặc so sánh điều kiện (ví dụ: "salary > 50000"). Với các truy vấn phức tạp như kết nối bảng (JOIN), CPU xử lý từng cặp dòng từ hai bảng, so sánh khóa kết nối, và tạo kết quả. DuckDB tối ưu hóa quá

trình này bằng cách sử dụng các thuật toán hiệu quả như hash join, tận dụng sức mạnh tính toán của CPU.

1.1.2.2. Cấu trúc và khả năng tính toán song song của GPU

Bộ phận xử lý đồ họa (GPU, graphics processing unit) là một vi mạch chuyên dụng được thiết kế để thao tác và truy cập bộ nhớ đồ họa một cách nhanh chóng, đẩy nhanh việc tạo ra các hình ảnh trong bộ đệm khung hình (framebuffer) trước khi xuất ra màn hình hiển thị. [5]



Hình 2: Kiến trúc của GPU

GPU có cấu trúc khác biệt rõ rệt so với CPU, được thiết kế để xử lý tính toán song song trên khối lượng dữ liệu lớn, đặc biệt phù hợp với các tác vụ như xử lý truy vấn trong cơ sở dữ liệu phân tích. GPU bao gồm hàng nghìn nhân xử lý nhỏ (CUDA cores trong GPU NVIDIA), được tổ chức thành các khối tính toán gọi là Streaming Multiprocessors (SM). Mỗi SM chứa nhiều nhân, bộ nhớ đệm nhỏ, và các đơn vị điều khiển, cho phép thực hiện đồng thời hàng trăm hoặc hàng nghìn luồng (threads). Ví dụ, một GPU NVIDIA RTX 3090 có hơn 10.000 CUDA cores, vượt xa số nhân của CPU, tạo nền tảng cho khả năng song song hóa cao.

Khả năng tính toán song song của GPU dựa trên mô hình lập trình SIMD (Single Instruction, Multiple Data). Trong mô hình này, một lệnh duy nhất được áp dụng đồng thời cho nhiều phần tử dữ liệu khác nhau. Với truy vấn cơ sở dữ liệu như “SELECT * FROM sales WHERE revenue > 1000”, GPU có thể thực hiện phép so sánh “revenue > 1000” trên hàng triệu dòng cùng lúc, mỗi dòng do một luồng xử lý riêng. HeavyDB tận dụng đặc điểm này để tăng tốc các tác vụ như lọc, tổng hợp, hoặc kết nối bảng, đạt hiệu suất cao hơn hàng chục lần so với CPU khi xử lý dữ liệu lớn.

Bộ nhớ của GPU, thường gọi là VRAM (Video RAM), có dung lượng lớn (từ 8GB đến 80GB trong các dòng hiện đại) và băng thông cao, được tối ưu để lưu trữ và truy cập dữ liệu song song. Tuy nhiên, VRAM có độ trễ cao hơn so với cache của CPU, vì nó phục vụ hàng nghìn luồng cùng lúc thay vì ưu tiên truy xuất ngẫu nhiên nhanh. Trong xử lý truy vấn, dữ liệu phải được chuyển từ RAM hệ thống sang VRAM trước khi tính toán, gây ra chi phí khởi tạo (overhead). HeavyDB giảm thiểu vấn đề này bằng cách lưu trữ dữ liệu nóng trong VRAM, cho phép truy vấn liên tục mà không cần chuyển dữ liệu nhiều lần.

GPU sử dụng các framework lập trình như CUDA hoặc OpenCL để khai thác khả năng song song. Các truy vấn được lập trình thành các kernel—các hàm nhỏ chạy đồng thời trên nhiều luồng. Ví dụ, trong một kernel lọc dữ liệu, mỗi luồng kiểm tra một dòng và trả về kết quả, sau đó GPU tổng hợp kết quả từ tất cả luồng. Điều này đòi hỏi truy vấn phải được thiết kế để song song hóa tốt, nếu không hiệu suất sẽ giảm. HeavyDB tận dụng CUDA để tối ưu các truy vấn phân tích, biến GPU thành công cụ mạnh mẽ cho dữ liệu lớn, nhưng kém linh hoạt với các tác vụ tuần tự hoặc logic phức tạp.

Tóm lại, cấu trúc GPU với hàng nghìn nhân nhỏ, mô hình SIMD, và VRAM dung lượng lớn mang lại khả năng tính toán song song vượt trội, lý tưởng cho các truy vấn dữ liệu lớn như trong HeavyDB. Tuy nhiên, nó phụ thuộc vào việc chuyển dữ liệu ban đầu và khả năng song song hóa của truy vấn. So với CPU, GPU ưu việt hơn trong các tác vụ phân tích dữ liệu lớn, nhưng kém hiệu quả với dữ liệu nhỏ hoặc truy vấn không thể chia nhỏ, làm nổi bật sự khác biệt trong ứng dụng thực tế.

1.1.2.3. So sánh ưu, nhược điểm của CPU và GPU khi thực hiện các tác vụ truy vấn dữ liệu lớn

Khi xử lý các tác vụ truy vấn dữ liệu lớn, CPU (Central Processing Unit) và GPU (Graphics Processing Unit) mang lại những ưu điểm và nhược điểm riêng, phụ thuộc vào cấu trúc và cách chúng thực thi công việc. CPU có ưu thế về tính linh hoạt và khả năng xử lý tuần tự. Với số lượng nhân ít (thường từ 4 đến 32) nhưng mỗi nhân mạnh mẽ, CPU vượt trội trong các truy vấn phức tạp đòi hỏi logic điều khiển hoặc xử lý từng bước, như kết nối bảng (JOIN) với điều kiện không song song hóa được. DuckDB tận dụng CPU để thực hiện vectorized execution trên dữ liệu vừa phải, đảm bảo hiệu suất tốt mà không cần phần cứng chuyên dụng. Ngoài ra, bộ nhớ đệm (cache) nhanh của CPU giúp giảm độ trễ khi truy cập dữ liệu ngẫu nhiên, rất hữu ích với các truy vấn cần lặp lại trên tập dữ liệu nhỏ.

Tuy nhiên, CPU có nhược điểm rõ ràng khi xử lý dữ liệu lớn. Số nhân giới hạn khiến nó kém hiệu quả trong các tác vụ song song hóa cao, như lọc hoặc tổng hợp trên hàng tỷ dòng dữ liệu. Điều này dẫn đến thời gian xử lý kéo dài khi khối lượng dữ liệu tăng lên. Hơn nữa, CPU tiêu tốn nhiều năng lượng hơn trên mỗi phép tính so với GPU trong các tác vụ nặng, làm tăng chi phí vận hành với khối lượng công việc lớn. Với các hệ thống như DuckDB, hiệu suất CPU phụ thuộc nhiều vào thuật toán tối ưu, nhưng vẫn không thể cạnh tranh với GPU khi dữ liệu vượt quá khả năng song song của nó.

Ngược lại, GPU nổi bật với khả năng tính toán song song vượt trội nhờ hàng nghìn nhân nhỏ (CUDA cores). Điều này khiến GPU lý tưởng cho các truy vấn dữ liệu lớn có thể chia nhỏ, như lọc (WHERE), tổng hợp (SUM, AVG), hoặc kết nối bảng song song. HeavyDB tận dụng GPU để xử lý hàng tỷ dòng trong vài giây, vượt xa CPU về tốc độ với các tập dữ liệu khổng lồ. Ngoài ra, VRAM dung lượng lớn của GPU (thường từ 8GB đến 80GB) cho phép lưu trữ toàn bộ dữ liệu cần xử lý, giảm nhu cầu chuyển dữ liệu từ RAM hệ thống, đặc biệt khi truy vấn lặp lại trên cùng tập dữ liệu.

Dù vậy, GPU cũng có những hạn chế đáng kể. Chi phí khởi tạo cao (overhead) khi chuyển dữ liệu từ RAM sang VRAM làm giảm hiệu suất với các truy vấn nhỏ hoặc không liên tục. GPU cũng kém linh hoạt hơn CPU do phụ thuộc vào các framework như CUDA, yêu cầu truy vấn phải được thiết kế để

song song hóa tối đa. Nếu truy vấn không thể chia nhỏ, hiệu suất GPU giảm mạnh, thậm chí kém hơn CPU. HeavyDB đòi hỏi phần cứng GPU tương thích và cấu hình phức tạp, làm tăng chi phí triển khai so với các giải pháp CPU như DuckDB.

Tóm lại, CPU phù hợp với truy vấn dữ liệu lớn có logic phức tạp hoặc kích thước vừa phải, nhờ tính linh hoạt và bộ nhớ đệm nhanh, nhưng yếu thế khi cần song song hóa cao. GPU vượt trội trong các tác vụ truy vấn lớn, song song, với tốc độ và dung lượng VRAM ấn tượng, nhưng bị hạn chế bởi chi phí khởi tạo và yêu cầu lập trình đặc thù.

1.1.3. Song song hóa trong cơ sở dữ liệu

1.1.3.1. Cách GPU tăng tốc truy vấn bằng cách thực thi song song (Parallel Execution)

GPU (Graphics Processing Unit) tăng tốc truy vấn dữ liệu bằng cách tận dụng khả năng thực thi song song (parallel execution), một đặc trưng cốt lõi của kiến trúc phần cứng của nó. Không giống CPU với số nhân giới hạn, GPU có hàng nghìn nhân nhỏ (CUDA cores trong GPU NVIDIA), được tổ chức thành các Streaming Multiprocessors (SM). Mỗi SM có thể xử lý đồng thời hàng trăm luồng (threads), cho phép GPU thực hiện nhiều phép tính cùng lúc. Trong xử lý truy vấn, điều này có nghĩa là các tác vụ như lọc, tổng hợp, hoặc kết nối bảng có thể được chia nhỏ và giao cho các luồng riêng biệt, giảm đáng kể thời gian thực thi. HeavyDB là một ví dụ điển hình khi sử dụng GPU để tăng tốc truy vấn trên dữ liệu lớn.

Cơ chế song song của GPU hoạt động dựa trên mô hình SIMD (Single Instruction, Multiple Data). Với mô hình này, một lệnh duy nhất được áp dụng đồng thời cho nhiều phần tử dữ liệu khác nhau. Chẳng hạn, trong truy vấn "SELECT * FROM sales WHERE revenue > 1000", GPU có thể giao mỗi dòng dữ liệu cho một luồng riêng, và tất cả luồng cùng thực hiện phép so sánh "revenue > 1000" cùng một lúc. Kết quả từ hàng triệu dòng được xử lý song song, sau đó tổng hợp lại nhanh chóng nhờ khả năng phối hợp của các nhân GPU. Điều này giúp GPU vượt trội so với CPU trong các truy vấn yêu cầu xử lý dữ liệu lớn mà không phụ thuộc lẫn nhau.

Để thực thi song song hiệu quả, GPU yêu cầu dữ liệu được chuyển từ RAM hệ thống sang VRAM (Video RAM) trước khi xử lý. VRAM có băng thông cao, cho phép hàng nghìn luồng truy cập dữ liệu đồng thời mà không gây tắc nghẽn. HeavyDB tối ưu hóa quá trình này bằng cách giữ dữ liệu trong VRAM cho các truy vấn liên tục, giảm chi phí chuyển dữ liệu ban đầu (overhead). Ví dụ, khi thực hiện tổng hợp "SELECT SUM(revenue) GROUP BY region", GPU chia dữ liệu thành các nhóm nhỏ, giao mỗi nhóm cho một tập hợp luồng, và tính tổng song song trên từng nhóm, sau đó kết hợp kết quả cuối cùng với tốc độ cao.

GPU sử dụng các kernel—các hàm nhỏ được lập trình bằng CUDA hoặc OpenCL—để điều phối thực thi song song. Mỗi kernel được thiết kế để chạy trên hàng nghìn luồng, với mỗi luồng xử lý một phần dữ liệu. Trong truy vấn kết nối bảng (JOIN), GPU có thể song song hóa việc so sánh khóa giữa hai bảng lớn, phân bổ mỗi cặp dòng cho một luồng riêng. HeavyDB tận dụng kernel để thực hiện các phép tính phức tạp như hash join trên GPU, đạt hiệu suất cao hơn nhiều so với CPU. Tuy nhiên, hiệu quả của song song hóa phụ thuộc vào việc truy vấn có thể chia nhỏ thành các tác vụ độc lập; nếu không, hiệu suất sẽ giảm đáng kể.

Tóm lại, GPU tăng tốc truy vấn bằng cách thực thi song song nhờ hàng nghìn nhân xử lý, mô hình SIMD, VRAM băng thông cao, và kernel lập trình tối ưu. Điều này cho phép xử lý dữ liệu lớn nhanh chóng, như được minh chứng trong HeavyDB, đặc biệt với các truy vấn như lọc, tổng hợp, hoặc kết nối bảng. Dù vậy, nó đòi hỏi dữ liệu phải được chuẩn bị tốt và truy vấn phải phù hợp với tính toán song song, nếu không sẽ không phát huy hết tiềm năng so với CPU.

1.1.3.2. Các kỹ thuật song song hóa phổ biến trong truy vấn cơ sở dữ liệu

- Song song hóa mức dữ liệu (Data Parallelism): Kỹ thuật này chia dữ liệu thành nhiều phần nhỏ và thực hiện các thao tác giống nhau trên từng phần dữ liệu đồng thời. GPU đặc biệt phù hợp với kỹ thuật này nhờ vào số lượng lõi xử lý lớn. Ví dụ:
  - Khi thực hiện phép lọc dữ liệu (WHERE clause), mỗi lõi có thể xử lý một phần dữ liệu riêng biệt.
  - Với phép tính tổng (SUM), dữ liệu được chia nhỏ, các lõi GPU tính toán đồng thời rồi hợp nhất kết quả sau cùng.
- Song song hóa mức lệnh (Instruction Parallelism): Ở mức này, nhiều lệnh khác nhau có thể được thực hiện đồng thời trên phần cứng. Ví dụ:
  - Trong truy vấn SELECT column1, column2 FROM table WHERE condition, GPU có thể thực hiện việc quét cột (column1, column2) và lọc dữ liệu cùng lúc thay vì tuần tự như CPU.
  - Với phép kết hợp bảng (JOIN), GPU có thể thực hiện nhiều phép so sánh khóa cùng lúc, giúp tăng tốc độ xử lý.
- Song song hóa mức truy vấn (Query Parallelism): Khi có nhiều truy vấn độc lập, hệ thống có thể chạy đồng thời trên nhiều lõi GPU mà không phải chờ từng truy vấn hoàn tất. Ví dụ:
  - Một hệ thống phân tích dữ liệu có thể thực thi nhiều truy vấn từ người dùng khác nhau trên GPU, thay vì xử lý tuần tự như CPU.
  - Các tác vụ như tạo chỉ mục, tính toán thống kê có thể chạy đồng thời mà không làm gián đoạn quá trình truy vấn.
- Song song hóa mức tác vụ (Task Parallelism): Từng phần của một truy vấn có thể được chia thành các tác vụ nhỏ hơn và thực thi đồng thời. Ví dụ:
  - Trong truy vấn phức tạp có GROUP BY, ORDER BY, GPU có thể xử lý nhóm dữ liệu, tính toán tổng hợp và sắp xếp đồng thời.
  - Với phép nối bảng (JOIN), GPU có thể thực hiện quét dữ liệu, tính toán hàm băm và so khớp dữ liệu đồng thời thay vì từng bước như trên CPU.
- Vectorization và SIMD (Single Instruction Multiple Data): Đây là kỹ thuật tận dụng GPU để thực hiện cùng một lệnh trên nhiều phần tử dữ liệu cùng lúc. Điều này đặc biệt hiệu quả với:
  - Phép toán ma trận, tính toán thống kê (trung bình, phương sai).
  - Xử lý tập dữ liệu lớn trong phân tích dữ liệu thời gian thực.

1.1.3.3. Các yếu tố ảnh hưởng đến hiệu suất của GPU trong xử lý dữ liệu

- Băng thông bộ nhớ:

Băng thông bộ nhớ quyết định tốc độ di chuyển dữ liệu giữa bộ nhớ GPU và các đơn vị xử lý. GPU có băng thông cao giúp giảm độ trễ khi truy xuất dữ liệu, từ đó cải thiện hiệu suất xử lý truy vấn. Nếu dữ liệu phải truyền liên tục giữa CPU và GPU, hiệu suất sẽ giảm đáng kể. Các công nghệ như HBM (High Bandwidth Memory) và kỹ thuật caching được sử dụng để giảm số lần truy cập bộ nhớ và tăng tốc xử lý.

- Mô hình lập trình và thuật toán tối ưu:

GPU hoạt động hiệu quả nhất khi các thuật toán được thiết kế để tận dụng khả năng xử lý song song. Việc tối ưu hóa bộ nhớ, chẳng hạn như sử dụng bộ nhớ chia sẻ (shared memory) thay vì bộ

nhớ toàn cục (global memory), giúp cải thiện tốc độ truy xuất. Ngoài ra, việc lập trình theo kiến trúc CUDA (NVIDIA) hoặc HIP (AMD) và sử dụng các thư viện tối ưu như cuBLAS, cuDF có thể nâng cao hiệu suất đáng kể.

- Kích thước và mô hình dữ liệu:

GPU xử lý hiệu quả hơn với dữ liệu lớn và có cấu trúc dạng cột (columnar storage) so với dữ liệu dạng hàng (row-based storage). Nếu dữ liệu vượt quá dung lượng bộ nhớ GPU (VRAM), hiện tượng tràn bộ nhớ (Out of Memory - OOM) có thể xảy ra, làm chậm hoặc dừng quá trình xử lý. Để tối ưu, các định dạng dữ liệu như Apache Arrow, Parquet và các kỹ thuật phân vùng dữ liệu hợp lý được sử dụng.

- Độ trễ truyền dữ liệu giữa CPU và GPU:

Việc truyền dữ liệu giữa CPU và GPU có thể gây ra độ trễ đáng kể, đặc biệt khi dữ liệu phải di chuyển liên tục giữa hai bộ xử lý. Một giải pháp để giảm độ trễ là giữ dữ liệu trên GPU càng lâu càng tốt, hoặc sử dụng công nghệ GPUDirect Storage để tải dữ liệu trực tiếp từ ổ cứng NVMe vào GPU mà không cần thông qua CPU.

- Hiệu suất của lõi GPU:

GPU hiện đại có hàng nghìn lõi xử lý song song, nhưng hiệu suất thực tế còn phụ thuộc vào số lượng lõi CUDA (đối với NVIDIA) hoặc Stream Processors (đối với AMD), tốc độ xung nhịp GPU, và khả năng xử lý dữ liệu số thực (FP32) hoặc số nguyên (INT8/INT16). Các tác vụ phân tích dữ liệu thường yêu cầu GPU có hiệu suất cao trong xử lý dấu chấm động, chẳng hạn như NVIDIA A100 hoặc H100.

- Hỗ trợ phần mềm và hệ sinh thái:

Mức độ hỗ trợ của phần mềm ảnh hưởng lớn đến hiệu suất xử lý trên GPU. Các hệ quản trị cơ sở dữ liệu như HeavyDB, BlazingSQL và OmniSciDB được tối ưu hóa để khai thác GPU, trong khi các thư viện như RAPIDS, cuDF và cuML giúp tăng tốc xử lý dữ liệu mà không cần lập trình phức tạp. Lựa chọn phần mềm phù hợp với GPU sẽ giúp tối đa hóa hiệu suất xử lý dữ liệu.

1.2. Tổng quan tài liệu

1.2.1. Các nghiên cứu trước đây về GPU trong xử lý cơ sở dữ liệu

1.2.1.1. Tổng quan về các công trình nghiên cứu trước đây liên quan đến GPU-accelerated databases

Việc sử dụng GPU để tăng tốc xử lý cơ sở dữ liệu đã thu hút sự quan tâm đáng kể từ cộng đồng nghiên cứu và công nghiệp. Các nghiên cứu tập trung vào việc khai thác khả năng xử lý song song mạnh mẽ của GPU để cải thiện hiệu suất truy vấn và phân tích dữ liệu lớn. Ví dụ, hệ thống Crystal được thiết kế để tận dụng các lõi CUDA của GPU, cho thấy hiệu suất vượt trội so với các hệ thống truyền thống như HeavyDB và MonetDB . Ngoài ra, các nền tảng như BlazingSQL và PG-Strom cũng đã được phát triển để tích hợp khả năng tăng tốc GPU vào các hệ quản trị cơ sở dữ liệu hiện có.

1.2.1.2. Các bài báo khoa học về OmniSciDB (nay là HeavyDB) và các hệ thống tương tự

OmniSciDB, hiện được biết đến với tên HeavyDB, là một trong những hệ thống cơ sở dữ liệu tăng tốc bằng GPU nổi bật. HeavyDB là một hệ quản trị cơ sở dữ liệu quan hệ, lưu trữ dạng cột, sử dụng GPU để tăng tốc truy vấn OLAP . Nhiều bài báo khoa học đã phân tích kiến trúc và hiệu suất của hệ

thống này, nhấn mạnh khả năng xử lý truy vấn SQL trên hàng tỷ bản ghi trong thời gian mili-giây . Các nghiên cứu cũng đã so sánh hiệu suất của HeavyDB với các hệ thống khác, cho thấy sự cải thiện đáng kể về tốc độ truy vấn khi sử dụng tăng tốc GPU.

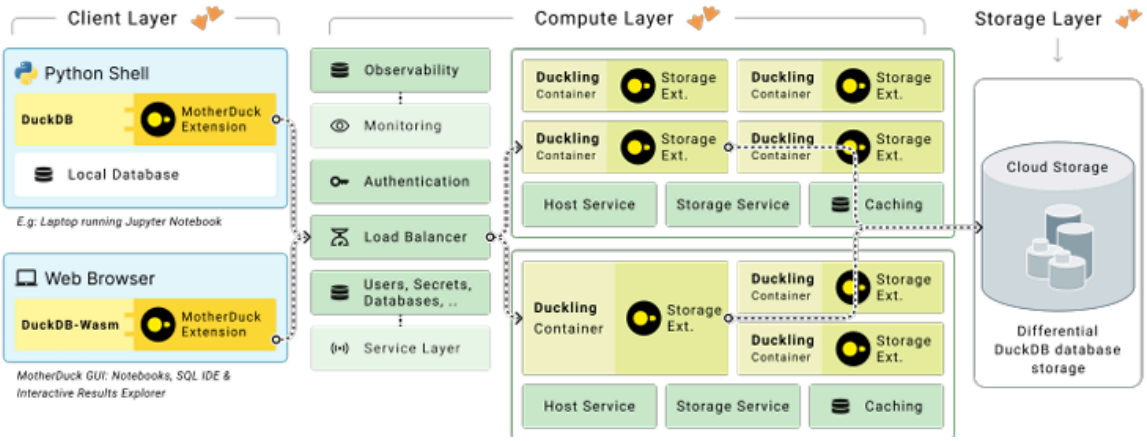
1.2.1.3. Hạn chế của các nghiên cứu trước và điểm cần cải thiện

Mặc dù các nghiên cứu đã chứng minh tiềm năng của việc sử dụng GPU trong xử lý cơ sở dữ liệu, vẫn tồn tại một số hạn chế cần được khắc phục. Nhiều hệ thống yêu cầu phần cứng chuyên dụng và có thể gặp khó khăn trong việc tích hợp với các hệ thống hiện có. Ngoài ra, việc tối ưu hóa hiệu suất giữa CPU và GPU vẫn là một thách thức, đòi hỏi các kỹ thuật lập lịch và quản lý tài nguyên hiệu quả. Hơn nữa, một số nghiên cứu tập trung vào các tập dữ liệu cụ thể, thiếu tính tổng quát khi áp dụng cho các loại dữ liệu và khối lượng công việc khác nhau. Do đó, cần có thêm các nghiên cứu để phát triển các phương pháp linh hoạt và hiệu quả hơn trong việc sử dụng GPU cho xử lý cơ sở dữ liệu.

1.2.2. Giới thiệu về DuckDB

DuckDB là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) được thiết kế chuyên biệt để xử lý phân tích dữ liệu trên CPU. Không giống như các hệ thống cơ sở dữ liệu truyền thống thường yêu cầu máy chủ chuyên dụng, DuckDB hoạt động theo mô hình nhúng (embedded database), giúp nó trở thành một công cụ linh hoạt và dễ dàng tích hợp vào các ứng dụng phân tích dữ liệu. Với khả năng hỗ trợ truy vấn trên các tập dữ liệu lớn mà không cần quá trình ETL (Extract, Transform, Load) phức tạp, DuckDB ngày càng được sử dụng rộng rãi trong nhiều lĩnh vực như khoa học dữ liệu, phân tích tài chính và hệ thống nhúng. [6]

1.2.2.1. Kiến trúc của DuckDB và cách hoạt động trên CPU



Hình 3: Kiến trúc xử lý truy vấn và lưu trữ trên DuckDB

DuckDB được thiết kế theo mô hình lưu trữ cột (columnar storage), cho phép tối ưu hóa các truy vấn phân tích dữ liệu. Không giống như cơ sở dữ liệu giao dịch (OLTP) sử dụng lưu trữ dòng (row-based storage) để tối ưu hóa các thao tác cập nhật dữ liệu, kiến trúc cột của DuckDB giúp tối ưu hóa việc đọc dữ liệu khi thực hiện các truy vấn tổng hợp (aggregation) hoặc truy vấn trên tập dữ liệu lớn.

Một trong những điểm nổi bật của DuckDB là sử dụng vectorized execution để tăng tốc quá trình thực thi truy vấn trên CPU. Vectorized execution giúp xử lý dữ liệu theo từng khối (chunk) thay vì từng dòng riêng lẻ, giúp tận dụng tốt hơn bộ nhớ đệm (CPU cache) và tăng tốc độ xử lý trên CPU hiện đại. Ngoài ra, DuckDB còn hỗ trợ adaptive query execution, cho phép tối ưu hóa động các truy vấn dựa trên dữ liệu thực tế thay vì chỉ dựa trên kế hoạch truy vấn ban đầu.

DuckDB hoạt động theo mô hình in-memory processing, giúp giảm độ trễ khi thực thi truy vấn bằng cách lưu trữ dữ liệu trong RAM thay vì đọc từ đĩa. Tuy nhiên, DuckDB vẫn có khả năng làm việc với các tập dữ liệu lớn bằng cách sử dụng cơ chế columnar paging, giúp chia nhỏ dữ liệu thành các trang (pages) để truy xuất hiệu quả hơn khi dữ liệu vượt quá bộ nhớ RAM.

1.2.2.2. Ưu điểm của DuckDB trong xử lý truy vấn trên CPU

DuckDB mang lại nhiều lợi ích đáng kể khi xử lý truy vấn trên CPU, đặc biệt là trong các tác vụ phân tích dữ liệu. Một số ưu điểm chính bao gồm: [7]

- Tích hợp dễ dàng và không cần máy chủ: DuckDB hoạt động như một cơ sở dữ liệu nhúng, không yêu cầu cài đặt máy chủ hoặc cấu hình phức tạp, giúp người dùng dễ dàng triển khai trong các ứng dụng phân tích dữ liệu.
- Tối ưu hóa cho truy vấn phân tích: Nhờ vào kiến trúc lưu trữ cột và vectorized execution, DuckDB có thể xử lý truy vấn phân tích nhanh hơn so với các hệ thống OLTP truyền thống.
- Hỗ trợ mạnh mẽ các định dạng dữ liệu phổ biến: DuckDB có thể đọc và xử lý trực tiếp các định dạng dữ liệu phổ biến như Parquet, CSV, Arrow, giúp giảm bớt quá trình chuyển đổi dữ liệu và cải thiện hiệu suất xử lý.
- Hiệu suất cao trên CPU đa lõi: Với khả năng xử lý song song trên nhiều lõi CPU, DuckDB có thể tận dụng tối đa hiệu suất phần cứng mà không yêu cầu GPU hay phần cứng chuyên dụng.
- Hỗ trợ SQL đầy đủ: DuckDB tuân thủ tiêu chuẩn SQL và hỗ trợ nhiều tính năng nâng cao như window functions, common table expressions (CTEs), và subqueries, giúp dễ dàng thực hiện các truy vấn phân tích phức tạp.

1.2.2.3. Các trường hợp ứng dụng thực tế của DuckDB

DuckDB được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau, đặc biệt là trong các tình huống yêu cầu phân tích dữ liệu lớn mà không cần cơ sở dữ liệu máy chủ phức tạp. Một số ứng dụng tiêu biểu của DuckDB bao gồm:

- Khoa học dữ liệu và Machine Learning: DuckDB giúp các nhà khoa học dữ liệu thực hiện truy vấn SQL trực tiếp trên dữ liệu mà không cần chuyển đổi sang hệ thống cơ sở dữ liệu khác. Nhờ khả năng làm việc tốt với Pandas, Arrow và Parquet, DuckDB được sử dụng để tiền xử lý dữ liệu nhanh chóng trước khi huấn luyện mô hình Machine Learning.
- Phân tích tài chính: DuckDB giúp các tổ chức tài chính xử lý nhanh chóng dữ liệu giao dịch, thực hiện các phép tính thống kê và phân tích xu hướng tài chính mà không cần cài đặt hệ thống cơ sở dữ liệu phức tạp.
- Hệ thống nhúng và ứng dụng trên thiết bị di động: Do không yêu cầu máy chủ, DuckDB có thể được sử dụng trong các ứng dụng nhúng, chẳng hạn như trên các thiết bị IoT hoặc phần mềm phân tích chạy trực tiếp trên máy tính cá nhân.
- Xử lý dữ liệu log và giám sát hệ thống: DuckDB được sử dụng để phân tích dữ liệu log từ các hệ thống lớn, giúp xác định nhanh chóng các vấn đề về hiệu suất và bảo mật.

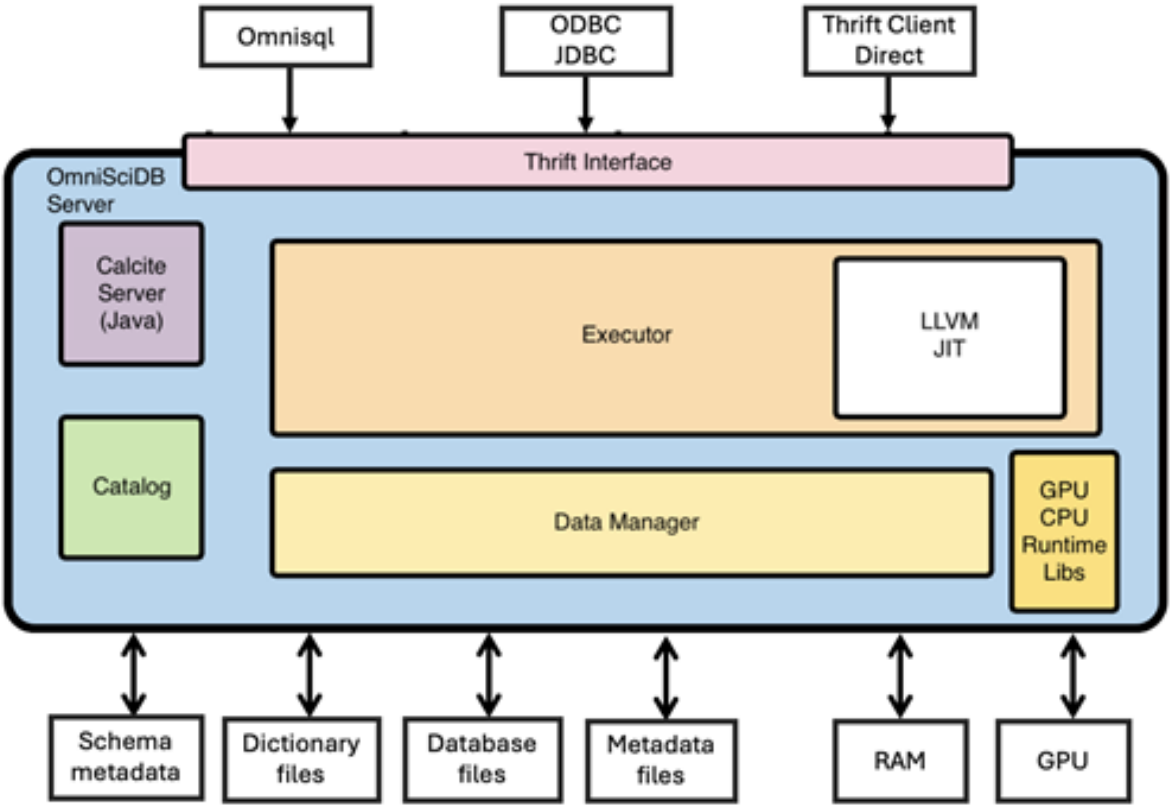
1.2.3. Giới thiệu về HeavyDB

HeavyDB (trước đây là OmniSciDB) là một hệ quản trị cơ sở dữ liệu phân tích tăng tốc bằng GPU, được thiết kế để xử lý khối lượng lớn dữ liệu với hiệu suất cao. Hệ thống này khai thác khả năng tính toán song song của GPU để tăng tốc các truy vấn SQL, giúp giảm thời gian phản hồi từ vài phút

xuống còn mili-giây trong các hệ thống truyền thống. HeavyDB đặc biệt phù hợp cho các ứng dụng phân tích thời gian thực, chẳng hạn như phân tích không gian, khoa học dữ liệu và tài chính. [8]

1.2.3.1. Kiến trúc và đặc điểm chính của HeavyDB

HeavyDB có kiến trúc hiện đại được thiết kế để tận dụng tối đa tài nguyên GPU và CPU.



Hình 4: Kiến trúc của HeavyDB

Kiến trúc này có thể được chia thành các thành phần chính như sau:

- Giao diện truy vấn: HeavyDB hỗ trợ nhiều giao diện kết nối như OmniSQL, ODBC/JDBC, và Thrift Client Direct, giúp người dùng dễ dàng gửi truy vấn từ các ứng dụng khác nhau.
- Thrift Interface: Là lớp trung gian cho phép giao tiếp giữa các ứng dụng bên ngoài với máy chủ HeavyDB, đảm bảo truy vấn được xử lý hiệu quả.
- OmniSciDB Server (HeavyDB Server):
  - Calicte Server (Java): Thành phần chịu trách nhiệm phân tích cú pháp và tối ưu hóa truy vấn SQL.
  - Catalog: Quản lý thông tin về schema của cơ sở dữ liệu, giúp tối ưu việc truy xuất dữ liệu.
  - Executor: Thành phần thực thi truy vấn, sử dụng LLVM JIT để biên dịch truy vấn thành mã máy, giúp tăng tốc đáng kể.
  - Data Manager: Quản lý dữ liệu, đảm bảo truy vấn được thực thi trên GPU hoặc CPU một cách tối ưu.

    - GPU/CPU Runtime Libraries: Các thư viện hỗ trợ thực thi trên phần cứng tương ứng, giúp điều phối tài nguyên giữa GPU và CPU linh hoạt.
- Tầng lưu trữ dữ liệu:
    - Schema metadata: Lưu trữ thông tin về cấu trúc bảng, cột và kiểu dữ liệu.
    - Dictionary files: Lưu trữ dữ liệu từ điển để hỗ trợ nén dữ liệu và tối ưu hiệu suất truy vấn.
    - Database files: Chứa dữ liệu chính của hệ thống.
    - Metadata files: Lưu trữ thông tin giúp tối ưu hóa truy vấn, chẳng hạn như chỉ mục hoặc thống kê dữ liệu.
    - RAM & GPU: Bộ nhớ chính và GPU được sử dụng để tăng tốc truy vấn và xử lý dữ liệu song song.

Đặc điểm chính của HeavyDB:

- Hiệu suất cao với GPU Acceleration: Sử dụng GPU để tăng tốc xử lý truy vấn, giúp phân tích dữ liệu lớn nhanh hơn nhiều lần so với các hệ thống truyền thống.
- Vectorized Query Execution: Sử dụng mô hình thực thi truy vấn theo vector, cho phép xử lý hàng triệu bản ghi đồng thời.
- Columnar Storage: Dữ liệu được lưu trữ theo cột thay vì theo hàng, giúp tối ưu hóa hiệu suất truy vấn và nén dữ liệu tốt hơn.
- LLVM JIT Compilation: Biên dịch truy vấn SQL trực tiếp thành mã máy để tối ưu hiệu suất thực thi.
- Hybrid Execution (GPU & CPU): Hỗ trợ chạy trên cả GPU và CPU, đảm bảo khả năng mở rộng linh hoạt.
- Tích hợp dễ dàng: Hỗ trợ nhiều chuẩn kết nối như ODBC, JDBC, và API REST, giúp tích hợp dễ dàng với các công cụ phân tích dữ liệu khác.

1.2.3.2. Cách HeavyDB tận dụng GPU để tối ưu hóa truy vấn

HeavyDB sử dụng GPU để thực thi các truy vấn SQL bằng cách phân chia công việc thành các tác vụ nhỏ hơn và xử lý song song trên hàng nghìn lõi GPU. Các kỹ thuật tối ưu hóa bao gồm:

- Thực thi song song (Parallel Execution): Thay vì xử lý từng dòng dữ liệu tuần tự trên CPU, HeavyDB chia dữ liệu thành nhiều phần và xử lý đồng thời trên nhiều lõi GPU. Điều này giúp tăng tốc đáng kể các truy vấn quét dữ liệu lớn.
- LLVM JIT Compilation: HeavyDB sử dụng LLVM JIT để biên dịch truy vấn SQL trực tiếp thành mã máy tối ưu cho GPU, giúp giảm thời gian thực thi.
- Vectorized Query Execution: HeavyDB thực hiện xử lý dữ liệu theo vector thay vì từng dòng một, giúp cải thiện hiệu suất truy vấn đáng kể khi làm việc với tập dữ liệu lớn.
- Memory Management Optimization: HeavyDB tận dụng bộ nhớ tốc độ cao của GPU để lưu trữ và xử lý dữ liệu, giảm thiểu độ trễ khi truy xuất dữ liệu từ ổ đĩa.

1.2.3.3. Các lợi ích khi sử dụng HeavyDB trong phân tích dữ liệu lớn

HeavyDB mang lại nhiều lợi ích vượt trội khi áp dụng trong các bài toán phân tích dữ liệu lớn, bao gồm:

- Tốc độ xử lý vượt trội: Nhờ vào GPU, HeavyDB có thể thực hiện các truy vấn phức tạp trên tập dữ liệu lớn nhanh hơn nhiều lần so với các hệ thống chỉ dùng CPU.
- Hỗ trợ phân tích thời gian thực (Real-time Analytics): HeavyDB cho phép phân tích dữ liệu ngay lập tức mà không cần chờ thời gian xử lý lâu, phù hợp với các hệ thống yêu cầu cập nhật dữ liệu liên tục.
- Khả năng mở rộng: HeavyDB có thể tận dụng nhiều GPU để mở rộng khả năng xử lý dữ liệu mà không làm giảm hiệu suất.
- Tiết kiệm tài nguyên: So với các hệ thống phân tán truyền thống, HeavyDB giúp giảm chi phí phần cứng và năng lượng nhờ khả năng xử lý nhanh chóng trên GPU mà không cần triển khai cụm máy chủ lớn.
- Dễ dàng tích hợp: HeavyDB hỗ trợ nhiều giao diện kết nối tiêu chuẩn như SQL, ODBC, JDBC, giúp dễ dàng tích hợp vào các hệ thống hiện có.

1.3. Kết luận chương

Chương 2 đã trình bày các cơ sở lý thuyết và tổng quan tài liệu liên quan đến cơ sở dữ liệu phân tích, các phương pháp tối ưu hóa truy vấn, và so sánh giữa CPU và GPU trong xử lý dữ liệu. Đồng thời, chương này cũng làm rõ ưu, nhược điểm của DuckDB và HeavyDB, hai hệ quản trị cơ sở dữ liệu tiêu biểu. Những nội dung này không chỉ cung cấp nền tảng lý thuyết vững chắc mà còn làm nổi bật khoảng trống nghiên cứu, đặc biệt là sự thiếu vắng các đánh giá thực nghiệm toàn diện giữa CPU và GPU. Đây sẽ là cơ sở để triển khai các phương pháp nghiên cứu và thực nghiệm trong các chương tiếp theo.