HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Lê Dương Phong

NGHIÊN CỨU PHÁT TRIỂN NỀN TẢNG TÍCH HỢP PHÂN TÍCH DỮ LIỆU DÒNG

LUẬN VĂN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

TP. HỒ CHÍ MINH - 2023

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Lê Dương Phong

NGHIÊN CỨU PHÁT TRIỂN NỀN TẢNG TÍCH HỢP PHÂN TÍCH DỮ LIỆU DÒNG

CHUYÊN NGÀNH: HỆ THỐNG THÔNG TIN

MÃ SỐ: 8.48.01.04

LUẬN VĂN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC

PGS.TS. THOẠI NAM

TP. HỒ CHÍ MINH – 2023

LỜI CAM ĐOAN

Tôi cam đoan rằng luận văn: **"Nghiên Cứu Phát Triển Nền Tảng Tích Hợp Phân Tích Dữ Liệu Dòng"** là công trình nghiên cứu của chính tôi.

Tôi cam đoan các số liệu, kết quả nêu trong luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Không có sản phẩm/nghiên cứu nào của người khác được sử dụng trong luận văn này mà không được trích dẫn theo đúng quy định.

TP. Hồ Chí Minh, ngày 28 tháng 02 năm 2023

Học viên thực hiện luận văn

Lê Dương Phong

LỜI CẢM ƠN

Trong suốt quá trình học tập và nghiên cứu thực hiện luận văn, ngoài nỗ lực của bản thân, tôi đã nhận được sự hướng dẫn nhiệt tình quý báu của quý Thầy Cô, cùng với sự động viên và ủng hộ của gia đình, bạn bè và đồng nghiệp. Với lòng kính trọng và biết ơn sâu sắc, tôi xin gửi lời cảm ơn chân thành tới:

Ban Giám Đốc , Phòng Đào tạo Sau đại học và quý Thầy Cô đã tạo mọi điều kiện thuận lợi giúp tôi hoàn thành luận văn.

Tôi xin chân thành cảm ơn Thầy **PGS.TS. Thoại Nam**, người thầy kính yêu đã hết lòng giúp đỡ, hướng dẫn, động viên, tạo điều kiện cho tôi trong suốt quá trình thực hiện và hoàn thành luận văn.

Tôi xin chân thành cảm ơn gia đình, bạn bè, đồng nghiệp trong cơ quan đã động viên, hỗ trợ tôi trong lúc khó khăn để tôi có thể học tập và hoàn thành luận văn.

Mặc dù đã có nhiều cố gắng, nỗ lực, nhưng do thời gian và kinh nghiệm nghiên cứu khoa học còn hạn chế nên không thể tránh khỏi những thiếu sót. Tôi rất mong nhận được sự góp ý của quý Thầy Cô cùng bạn bè đồng nghiệp để kiến thức của tôi ngày một hoàn thiện hơn.

Xin chân thành cảm ơn!

TP. Hồ Chí Minh, ngày 28 tháng 02 năm 2023

Học viên thực hiện luận văn

Lê Dương Phong

MỤC LỤC

DANH SÁCH HÌNH VẼ

DANH SÁCH BẢNG

DANH MỤC CÁC THUẬT NGỮ, CHỮ VIẾT TẮT

Viết tắt	Tiếng Anh	Tiếng Việt
ITS	Intelligence Transportation System	Hệ thống giao thông thông minh
HDFS	Hadoop File System	Hệ thống lưu dữ liệu được sử dụng bởi Hadoop
ACID	Atomicity, Consistency, Isolation, Durability	Bốn thuộc tính quan trọng của một hệ quản trị cơ sở dữ liệu: tính nguyên tử, tính nhất quán, tính cô lập, tính bền vững.
ETL	Extract Transform and Load	Trích xuất, chuyển đổi và tải
ELT	Extract Load and Transform	Trích xuất, tải và chuyển đổi

MỞ ĐẦU

Hiện nay, các tỉnh/thành phố trên cả nước nói chung cũng như tỉnh Tây Ninh đang tập trung xây dựng đô thị thông minh. Vấn đề chung của các đô thị thông minh đó là phải đối phó với lượng dữ liệu khổng lồ, đa định dạng, đa kích cỡ từ nhiều nguồn cung cấp khác nhau cho hệ thống giám sát đô thị. Vì vậy, hệ thống giám sát đô thị cần phải được xây dựng trên hạ tầng dữ liệu hiện đại, có khả năng lưu trữ, xử lý cũng như truy vấn khối lượng lớn dữ liệu .

Luận văn "Nghiên cứu phát triển nền tảng tích hợp phân tích dữ liệu dòng" được nghiên cứu và xây dựng mô hình trực quan bằng dữ liệu giám sát giao thông của tỉnh Tây Ninh để sử dụng vào đo đếm lưu lượng giao thông, nhận diện biển số phương tiện giao thông cũng như dự báo lưu lượng phương tiện giao thông tại một thời điểm nhất định. Qua đó, đề xuất được giải pháp lưu trữ, truy vấn dữ liệu của hệ thống giám sát để phục vụ các yêu cầu phân tích dữ liệu theo tư duy riêng của mình.

CHƯƠNG 1: GIỚI THIỆU

1.1. Tính cấp thiết của đề tài

Hiện nay, theo xu hướng xây dựng đô thị thông minh tại Việt Nam cũng như trên thế giới, hệ thống camera giám sát an ninh, giao thông, hỗ trợ du lịch là một thành phần cấu thành không thể thiếu luôn được ưu tiên khi lựa chọn đầu tư triển khai. Việc lắp đặt camera giám sát an ninh ở khu dân cư, các nút giao thông, các điểm du lịch với mục đích chính là phục vụ hiệu quả công tác phòng, chống các loại tội phạm về trật tự xã hội, bảo đảm an ninh trật tự trên địa bàn, góp phần giảm thiểu tai nạn giao thông, ùn tắc giao thông. Bên cạnh hệ thống camera giám sát an ninh, hệ thống camera còn tích hợp các công nghệ thông minh để hỗ trợ trong việc nhận diện biển số xe, nhận diện khuôn mặt, đo đếm lưu lượng phương tiện giao thông tại các điểm cửa ngõ của tỉnh/thành phố; hỗ trợ phát hiện, theo dõi các xe nghi ngờ, lưu trữ và trích xuất dữ liệu phục vụ công tác điều tra của các cơ quan quản lý nhà nước, v.v.

Đối với các hệ thống giám sát đặc biệt là hệ thống giám sát giao thông hiện đại ngày nay, số lượng dữ liệu được sinh ra ngày càng tăng do các hệ thống này được kết nối vô số cảm biến. Các cảm biến này có thể được lắp đặt trên các phương tiện giao thông di chuyển trên đường (thiết bị giám sát hành trình) hay là các hệ thống camera giám sát trên đường, bảng báo điện tử, thiết bị di động, v.v. Để đối phó với dữ liệu phức tạp, các hệ thống giám sát cần phải được xây dựng trên hạ tầng dữ liệu hiện đại, có khả năng lưu trữ, xử lý cũng như truy vấn khối lượng lớn dữ liệu.

Vì vậy, việc nghiên cứu phát triển nền tảng tích hợp phân tích dữ liệu dòng trong thời gian thực ở thời điểm hiện tại là rất cần thiết, đáp ứng nhu cầu xây dựng đô thị thông minh của các địa phương. Đó cũng chính là động lực để thực hiện luận văn này.

1.2. Mục tiêu và nhiệm vụ nghiên cứu

Mục tiêu của luận văn hướng đến là hiện thực vận dụng các giải pháp cho bài toán lưu trữ dữ liệu đo đếm phương tiện giao thông qua các công việc như sau:

- Nghiên cứu các giải pháp lưu trữ dữ liệu;
- Đề xuất giải pháp lưu trữ cho hệ thống tích hợp lưu trữ dữ liệu giao thông;
- Hiện thực triển khai thực tế giải pháp lưu trữ dữ liệu lớn cho dữ liệu đo đếm phương tiện giao thông song song với việc đánh giá hiệu năng;
- Hiện thực mô hình dự báo ngắn hạn lưu lượng giao thông sử dụng Support Vector Regression.

1.3. Phạm vi nghiên cứu

- Tìm hiểu kiến trúc Data Lakehouse;
- Tìm hiểu công nghệ lưu trữ dữ liệu lớn;
- Tìm hiểu giải thuật Support Vector Regression;
- Xây dựng kiến trúc triển khai thí điểm giải pháp trên thực tế;
- Đánh giá thực nghiệm dựa trên dữ liệu thực.

1.4. Kết cấu luận văn

Phần còn lại của luận văn bao gồm các chương sau:

- *Phần cơ sở lý thuyết:* Cơ sở lý thuyết điểm qua tình hình nghiên cứu trong và ngoài nước; nêu cơ sở lý luận về hệ thống lưu trữ dữ liệu giao thông; cuối cùng là lý thuyết về kiến trúc và thuật ngữ;
- *Phần bài toán hệ thống lưu trữ và truy vấn dữ liệu giao thông:* nêu rõ bài toán cần giải quyết; phân tích đặc trưng dữ liệu thực; phân tích yêu cầu về lưu trữ và truy vấn của hệ thống; dự báo ngắn hạn dữ liệu lưu lượng giao thông;
- *Phần xây dựng giải pháp và đánh giá hiệu quả:* bao gồm việc đề xuất giải pháp về kiến trúc; đề xuất giải pháp về công nghệ; kỹ thuật nâng cao hiệu năng truy vấn; giải thuật Support Vector Regression;
- *Phần kết luận:* Tổng kết nội dung đã trình bày.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Apache Kafka

2.1.1. Giới thiệu về Kafka

Xuất bản (Publish) / đăng ký (subscribe) nhắn tin (messaging) là một mô hình được đặc trưng bởi người gửi (nhà xuất bản / publisher) của một phần dữ liệu (tin nhắn / message) không hướng cụ thể nó đến người nhận. Thay vào đó, nhà xuất bản phân loại tin nhắn bằng cách nào đó và người nhận (người đăng ký / subscriber) đăng ký nhận một số loại thông báo nhất định. Các hệ thống pub / sub thường có một nhà môi giới (broker), một điểm trung tâm nơi các thông báo được xuất bản, để tạo điều kiện thuận lợi cho việc này.

Apache Kafka là một hệ thống nhắn tin đăng ký / xuất bản mã nguồn mở, được thiết kế để để xây dựng các đường ống dữ liệu trực tuyến thời gian thực lấy dữ liệu giữa nhiều hệ thống hoặc ứng dụng độc lập một cách đáng tin cậy. Nó thường được mô tả là "nhật ký cam kết phân tán" hoặc gần đây hơn là "nền tảng phát trực tuyến phân phối". Hệ thống tệp hoặc nhật ký cam kết cơ sở dữ liệu được thiết kế để cung cấp một bản ghi lâu dài về tất cả các giao dịch để chúng có thể được phát lại để dễ dàng xây dựng trạng thái của hệ thống. Tương tự, dữ liệu trong Kafka được lưu trữ lâu dài, theo thứ tự và có thể được đọc một cách xác định. Ngoài ra, dữ liệu có thể được phân phối trong hệ thống để cung cấp các biện pháp bảo vệ bổ sung chống lại các lỗi, cũng như không có cơ hội đáng kể để mở rộng hiệu suất.

Kafka cho phép:

- Xuất bản (publishing) và đăng ký (subscribing) luồng hồ sơ;
- Lưu trữ các luồng hồ sơ theo cách lâu bền, chịu được lỗi.

Nó cung cấp một nền tảng thống nhất, thông lượng cao, độ trễ thấp, có thể mở rộng theo chiều ngang được sử dụng trong sản xuất ở hàng nghìn công ty. Dưới đây là một số khái niệm cần thiết khi làm việc với Kafka.

Hình 2.1: Apache Kafka

2.1.2. Một số thành phần quan trọng của Kafka

Messages và Batches

Đơn vị dữ liệu trong Kafka được gọi là tin nhắn (message). Một tin nhắn chỉ đơn giản là một mảng byte, vì vậy dữ liệu chứa bên trong nó không có định dạng hoặc ý nghĩa cụ thể đối với Kafka. Một tin nhắn có thể có một bit tùy chọn của metadata, được gọi là khóa (key). Khóa cũng là một mảng byte giống như tin nhắn, không có ý nghĩa cụ thể đối với Kafka. Các khóa được sử dụng khi tin nhắn được ghi vào các phân vùng (partition) theo một cách có kiểm soát hơn.

Để đạt được hiệu quả, các tin nhắn được viết vào Kafka theo từng lô (batch). Một lô chỉ là một tập hợp các tin nhắn, tất cả đều được tạo cho cùng một chủ đề và phân vùng. Việc di chuyển liên tục trên mạng cho mỗi tin nhắn sẽ dẫn đến quá nhiều tin nhắn và việc thu thập các tin nhắn cùng nhau thành một loạt sẽ giảm thiểu điều này. Tất nhiên, đây là sự cân bằng giữa độ trễ và thông lượng: các lô càng lớn thì càng có nhiều thông báo có thể được xử lý trên một đơn vị thời gian, nhưng thời gian truyền tải một tin nhắn riêng lẻ càng lâu. Các lô cũng thường được nén, cung cấp khả năng truyền và lưu trữ dữ liệu hiệu quả hơn với chi phí là một số công suất xử lý.

Lược đồ (schemas)

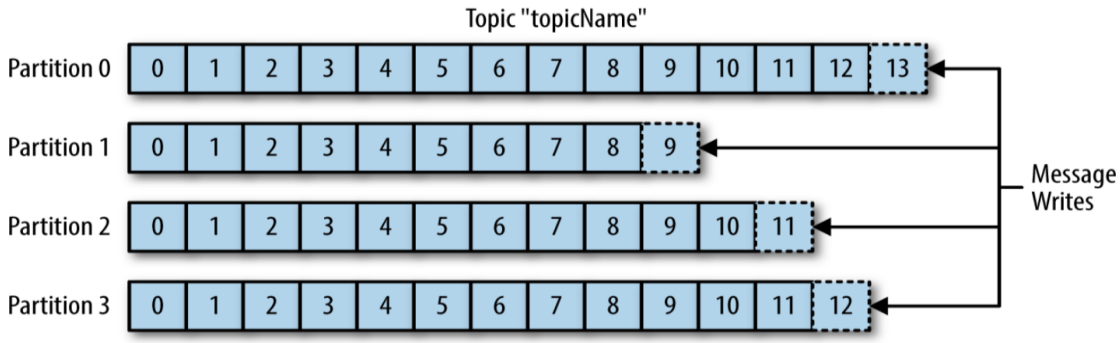
Nên áp đặt cấu trúc hoặc lược đồ bổ sung lên nội dung tin nhắn để có thể dễ dàng hiểu được nội dung của nó. Có nhiều tùy chọn có sẵn cho lược đồ tin nhắn, tùy thuộc vào nhu cầu cá nhân của ứng dụng của bạn. Các hệ thống đơn giản hóa, chẳng hạn như Ký hiệu đối tượng Javascript (JSON) và Ngôn ngữ đánh dấu mở rộng (XML),

rất dễ sử dụng và con người có thể đọc được. Tuy nhiên, chúng thiếu các tính năng như xử lý kiểu mạnh mẽ và tính khả dụng giữa các phiên bản lược đồ.

Định dạng dữ liệu nhất quán là rất quan trọng trong Kafka, vì nó cho phép phân tách các phép viết và đọc các bài viết. Khi các tác vụ này được kết hợp chặt chẽ với nhau, các ứng dụng ghi chú phụ vào thư phải được cập nhật để xử lý định dạng dữ liệu mới, song song với định dạng cũ. Chỉ khi đó, các ứng dụng xuất bản tin nhắn mới có thể được cập nhật để sử dụng định dạng mới. Bằng cách sử dụng các lược đồ được xác định rõ ràng và lưu trữ chúng trong một kho lưu trữ tổng hợp, các thông điệp trong Kafka có thể được hiểu mà không cần phối hợp.

Topics và partitions

Tin nhắn trong Kafka được phân loại thành các chủ đề. Các phép loại suy gần nhất cho một chủ đề là một bảng cơ sở dữ liệu hoặc một thư mục trong hệ thống tệp. Các chủ đề cũng được chia nhỏ thành một số phân vùng. Tin nhắn được viết vào log theo kiểu chỉ phần phụ và được đọc theo thứ tự từ đầu đến cuối. Lưu ý rằng vì một chủ đề thường có nhiều phân vùng, nên không có gì đảm bảo về thứ tự thời gian của tin nhắn trên toàn bộ chủ đề, chỉ trong một phân vùng duy nhất. Hình dưới đây cho thấy một chủ đề có bốn phân vùng, với các phần viết được nối vào cuối mỗi phần. Các phân vùng cũng là cách mà Kafka cung cấp dự phòng và khả năng mở rộng. Mỗi phân vùng có thể được lưu trữ trên một máy chủ khác nhau, có nghĩa là một chủ đề duy nhất có thể được điều chỉnh theo chiều ngang trên nhiều máy chủ để cung cấp hiệu suất vượt xa khả năng của một máy chủ.



Hình 2.2: Một chủ đề được biểu diễn với nhiều phân vùng

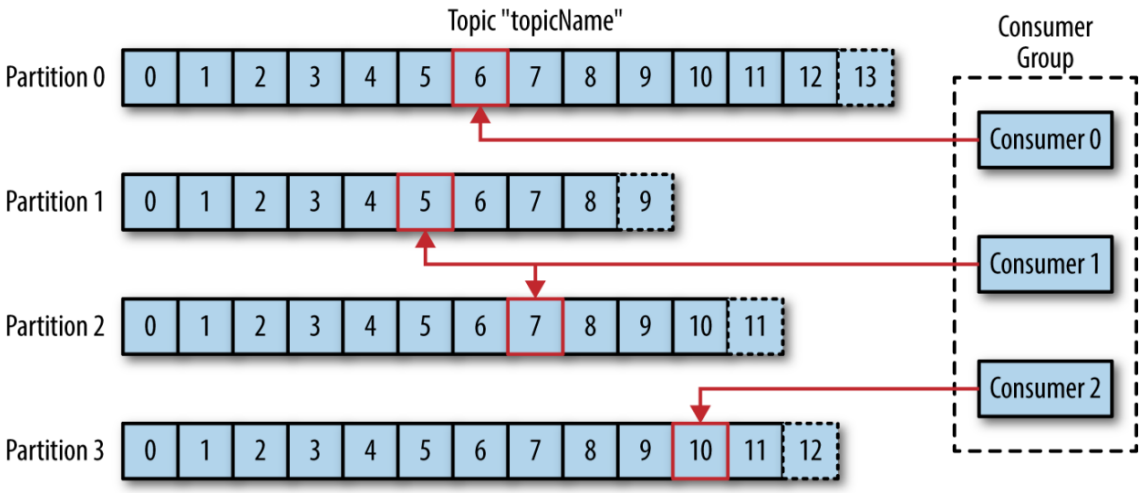
Thuật ngữ luồng (stream) thường được sử dụng khi thảo luận về dữ liệu trong các hệ thống như Kafka. Thông thường, một luồng được coi là một chủ đề dữ liệu duy nhất, bất kể số lượng phân vùng. Điều này thể hiện một luồng dữ liệu duy nhất di chuyển từ nhà sản xuất đến người tiêu dùng.

Người sản xuất và người tiêu dùng (Producers and Consumers)

Khách hàng của Kafka là những người sử dụng hệ thống, và có hai loại cơ bản: người sản xuất và người tiêu dùng. Nhà sản xuất tạo ra các thông điệp mới. Trong các hệ thống đăng ký / xuất bản khác, chúng có thể được gọi là nhà xuất bản hoặc nhà văn. Nói chung, một thông điệp sẽ được tạo ra cho một chủ đề cụ thể. Theo mặc định, nhà sản xuất không quan tâm phân vùng nào mà một thông báo cụ thể được viết tới và sẽ cân bằng các thông báo trên tất cả các phân vùng của một chủ đề một cách đồng đều. Trong một số trường hợp, trình quản lý sẽ chuyển các thông báo đến các phân vùng cụ thể. Điều này thường được thực hiện bằng cách sử dụng khóa thông báo và một trình phân vùng sẽ tạo ra một hàm băm của khóa và ánh xạ nó đến một phân vùng cụ thể. Điều này đảm bảo rằng tất cả các thông báo được tạo bằng một khóa nhất định sẽ được ghi vào cùng một phân vùng. Nhà sản xuất cũng có thể sử dụng một trình phân vùng tùy chỉnh tuân theo các quy tắc nghiệp vụ khác để ánh xạ thư tới các phân vùng. Người tiêu dùng đọc tin nhắn. Trong các hệ thống xuất bản / đăng ký khác, những khách hàng này có thể được gọi là người đăng ký hoặc người đọc. Người tiêu dùng đăng ký một hoặc nhiều chủ đề và đọc các thông báo theo thứ tự chúng được tạo ra. Người tiêu dùng theo dõi những thông điệp mà họ đã sử dụng bằng cách theo dõi phần bù của thông báo.

Người tiêu dùng làm việc như một phần của nhóm người tiêu dùng, là một hoặc nhiều người tiêu dùng làm việc cùng nhau để tiêu thụ một chủ đề. Nhóm đảm bảo rằng mỗi phân vùng chỉ được xác định bởi một thành viên. Trong hình dưới đây, có ba người tiêu dùng trong một nhóm duy nhất đang nghiên cứu một chủ đề. Hai trong số những người tiêu dùng đang làm việc từ một phân vùng, trong khi người tiêu dùng thứ ba đang làm việc từ hai phân vùng. Ánh xạ của người tiêu dùng tới một

phân vùng thường được người tiêu dùng gọi là quyền sở hữu phân vùng đó. Bằng cách này, người tiêu dùng có thể mở rộng quy mô theo chiều ngang để sử dụng các chủ đề có số lượng thông điệp lớn. Ngoài ra, nếu một người tiêu dùng đơn lẻ không thành công, các thành viên còn lại của nhóm sẽ cân bằng lại các phân vùng đang được tiêu thụ để tiếp quản cho thành viên bị thiếu.

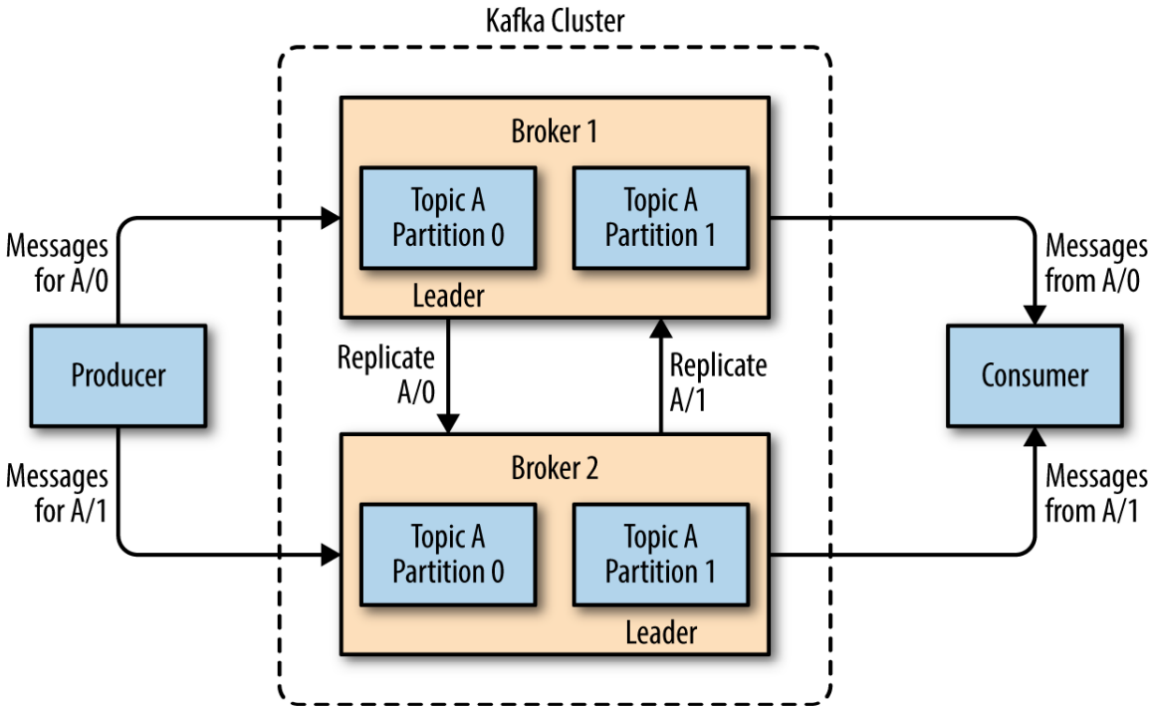


Hình 2.3: Nhóm người dùng cùng nghiên cứu một chủ đề

Brokers và Clusters

Một máy chủ Kafka duy nhất được gọi là một nhà môi giới. Người môi giới nhận tin nhắn từ nhà sản xuất, ấn định hiệu số cho họ và cam kết lưu trữ các tin nhắn trên đĩa. Nó cũng phục vụ người tiêu dùng, đáp ứng các yêu cầu tìm nạp cho các phân vùng và đáp ứng với các trung gian đã được cam kết với đĩa. Tùy thuộc vào phần cứng cụ thể và đặc điểm hiệu suất của nó, một nhà môi giới duy nhất có thể dễ dàng xử lý hàng nghìn phân vùng và hàng triệu thông báo mỗi giây. Các nhà môi giới Kafka được thiết kế để hoạt động như một phần của một cụm. Trong một nhóm các nhà môi giới, một nhà môi giới cũng sẽ hoạt động như người điều khiển cụm (được bầu tự động từ các thành viên trực tiếp của nhóm). Bộ điều khiển chịu trách nhiệm về hoạt động quản trị, bao gồm chỉ định phân vùng cho nhà môi giới và giám sát các lỗi của nhà môi giới. Một phân vùng được sở hữu bởi một nhà môi giới duy nhất trong cụm và nhà môi giới đó được gọi là nhà lãnh đạo của phân vùng. Một phân

vùng có thể được gán cho nhiều nhà môi giới, điều này sẽ dẫn đến việc phân vùng được nhân rộng. Điều này cung cấp dự phòng các thông báo trong phân vùng, để một nhà môi giới khác có thể tiếp quản quyền lãnh đạo nếu có lỗi nhà môi giới. Tuy nhiên, tất cả người tiêu dùng và nhà sản xuất hoạt động trên phân vùng đó phải kết nối với người lãnh đạo.



Hình 2.4: Nhân rộng các phân vùng trong một cụm

Một tính năng chính của Apache Kafka là lưu giữ, tức là lưu trữ lâu dài các tin nhắn trong một khoảng thời gian. Các nhà môi giới Kafka được định cấu hình với cài đặt mặc định cho các chủ đề, giữ lại các tin nhắn trong một khoảng thời gian (ví dụ: 7 ngày) hoặc cho đến khi chủ đề đạt đến kích thước nhất định tính bằng byte (ví dụ: 1 GB). Khi đạt đến các giới hạn này, thư sẽ hết hạn và bị xóa để cấu hình lưu giữ là lượng dữ liệu tối thiểu có sẵn bất kỳ lúc nào. Các chủ đề riêng lẻ cũng có thể được định cấu hình bằng các cài đặt lưu giữ của riêng chúng để các tin nhắn chỉ được lưu trữ miễn là chúng hữu ích. Ví dụ: một chủ đề theo dõi có thể được giữ lại trong vài ngày, trong khi số liệu ứng dụng có thể được giữ lại chỉ trong vài giờ. Các chủ đề

cũng có thể được định cấu hình dưới dạng log compacted, có nghĩa là Kafka sẽ chỉ giữ lại vị hiền triết cuối cùng được tạo ra với một khóa cụ thể.

2.2. Apache Spark

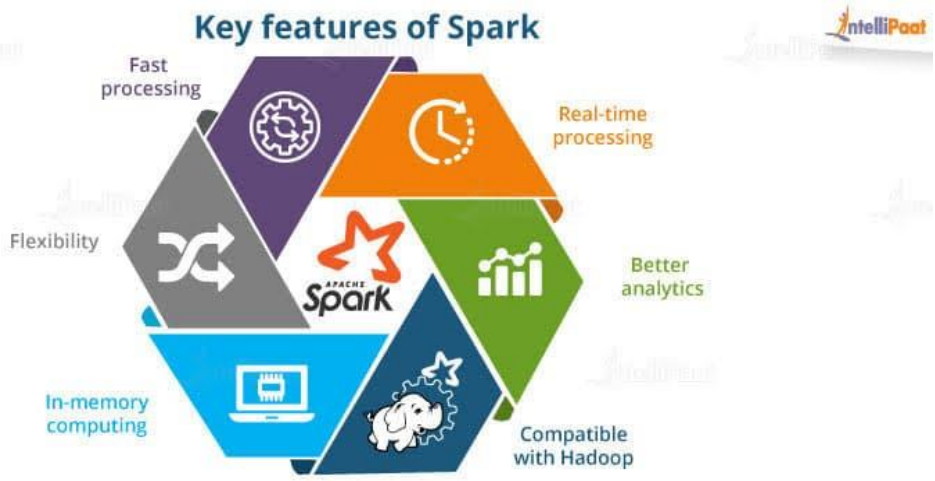
2.2.1. Giới thiệu về Apache Spark

Việc xử lý khối lượng lớn dữ liệu không phải là một vấn đề hoàn toàn mới, nhưng vẫn đang cần được giải quyết thấu đáo trong hoàn cảnh lượng dữ liệu cần được xử lý đang ngày càng tăng về cả độ lớn và độ phức tạp. Một trong những ứng viên sáng giá cho hệ thống xử lý dữ liệu lớn có thể kể đến như Hadoop. Tuy nhiên, cho dù Hadoop là một công cụ phân tích Big Data rất mạnh và phổ biến nhưng nó đã bộc lộ rất nhiều nhược điểm cần giải quyết như không hỗ trợ các tập tin có kích thước nhỏ, chỉ cho phép phân tích dữ liệu theo bó (batch) mà không hỗ trợ xử lý theo thời gian thực, độ trễ cao vì phụ thuộc HDFS, khó sử dụng,… Các vấn đề trên lại là những thách thức quan trọng được đặt ra khi giải quyết các bài toán phân tích Big Data. Do đó Apache Spark là một công cụ phân tích Big Data hiệu quả hơn nhằm tăng hiệu suất khai thác hệ thống SuperNode-XP.

Apache Spark là một framework xử lý dữ liệu giúp nhanh chóng thực hiện các tác vụ xử lý trên các tập dữ liệu rất lớn. Spark giúp phân phối các tác vụ xử lý dữ liệu trên nhiều máy tính hoặc cùng với các công cụ tính toán phân tán khác. Spark cũng giúp giảm gánh nặng cho các nhà phát triển với các công cụ đơn giản hoá việc phân bổ tài nguyên cho lưu trữ và tính toán song song, phân tán trên nhiều node.

Từ khởi đầu khiêm tốn trong AMPLab tại U.C. Berkeley trong 2009, Apache Spark đã trở thành một trong những framework xử lý phân tán Big Data quan trọng nhất trên thế giới. Trước khi Spark ra đời, Hadoop MapReduce đang là nột trong những framework xử lý dữ liệu phổ biến nhất vào thời gian đó. Sau khi được đưa ra cộng đồng và trở thành mã nguồn mở trong năm 2010, Spark đã tăng trưởng và được chuyển giao cho Apache Software Foundation vào năm 2013. Được sử dụng bởi các ngân hàng, các công ty viễn thông, các công ty trò chơi trực tuyến, chính phủ và cả

những gã khổng lồ công nghệ lớn như Apple, Facebook, IBM và Microsoft, Spark đã đem lại rất nhiều lợi ích với những tính năng chính như sau:



Hình 2.5: Các tính năng chính của Spark

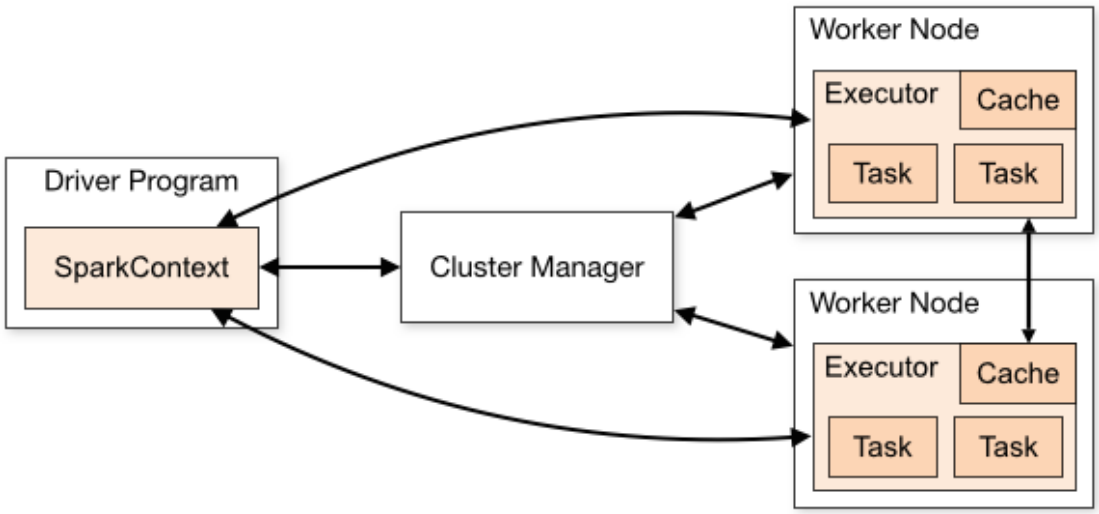
- **Xử lý với tốc độ nhanh**: Spark sử dụng Resilient Distributed Datasets (RDDs) giúp giảm thời gian đọc và ghi dữ liệu trên hệ thống cluster, từ đó tăng hiệu suất từ 10 tới 100 lần so với Hadoop MapReduce.
- **Tính linh hoạt**: Apache Spark hỗ trợ rất nhiều ngôn ngữ lập trình, cho phép các nhà phá triển xây dựng các ứng dụng phân tích Big Data trên Java, Scala, R hoặc Python. SparkSQL cũng giúp các nhà phân tích dữ liệu có thể tiếp tục thực thi các câu lệnh SQL trên hệ thống Big Data.
- **Tính toán trên bộ nhớ**: Spark lưu trữ dữ liệu trên RAM của các node tính toán. Điều này giúp các đơn vị tính toán có thể truy cập dữ liệu một các nhanh chóng, giúp đẩy nhanh tốc độ phân tích dữ liệu.
- **Xử lý theo thời gian thực**: Không như MapReduce chỉ có khả năng xử lý các tập dữ liệu cục bộ trên HDFS, Spark có khả năng xử lý các luồng dữ liệu đầu vào theo thời gian thực, từ đó cho ra các kết quả trong thời gian rất nhanh.
- **Phân tích thông minh**: Trái ngược với MapReduce chỉ có tính năng Map và Reduce dữ liệu, Spark cung cấp một tập hợp đa dạng các công cụ như các câu truy vấn bằng SQL, các mô hình và thuật toán học máy (machine learning),

biểu diễn đồ thị,… giúp người dùng có thể sử dụng Spark cho nhiều ngữ cảnh khác nhau.

- **Tương thích với Hadoop và nhiều trình quản lý tài nguyên**: Spark không chỉ là một framework đơn lẻ. Spark có thể kết hợp với Hadoop, bao gồm trình quản lý tập tin Hadoop HDFS, trình quản lý tài nguyên Hadoop YARN. Hơn nữa, Spark có thể được triển khai với Apache Mesos hay Kubernetes, giúp đơn giản hoá quá trình quản lý tài nguyên phân tán trên hệ thống.

2.2.2. Kiến trúc của Spark

Một ứng dụng chạy trên Apache Spark bao gồm hai thành phần chính: một driver (trình điều khiển) giúp chuyển đổi các dòng lệnh (code) của người sử dụng thành nhiều tác vụ có thể được xử lý phân tán, và các executor (trình thực thi) xử lý các tác vụ được giao từ driver. Với kiến trúc master-worker (hay còn gọi là master-slave) của Spark, driver được cài đặt trên master node trong khi các executor chạy trên các worker node.



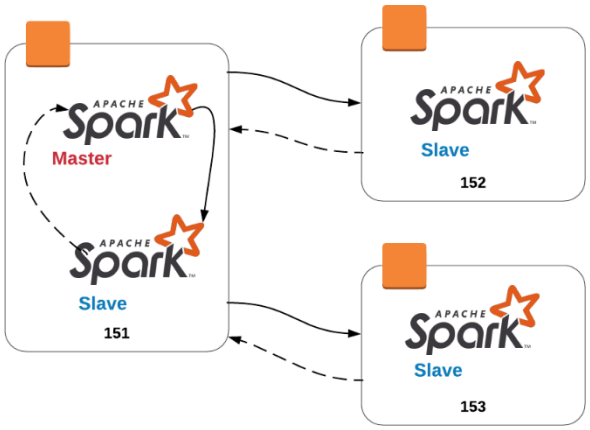
Hình 2.6: Kiến trúc của Apache Spark

Hình 1.6 mô tả kiến trúc của Apache Spark. Các ứng dụng trên Spark được thực thi bằng các tập hợp tiến trình độc lập trên cụm các node tính toán. Đầu tiên, chương trình driver trên node master gọi chương trình chính của ứng dụng và tạo ra một

SparkContext. Để chạy chương trình đó trên các node tính toán, SparkContext kết nối tới trình điều khiển cluster. Sau khi kết nối thành công, Spark yêu cầu cung cấp các executor trên các worker node và gửi tới các executor đó mã nguồn của ứng dụng (tập tin JAR hoặc Python). Cuối cùng, SparkContext gửi các tác vụ (task) tới các executor để thực thi. Kết quả tính toán được các executor gửi lại về SparkContext.

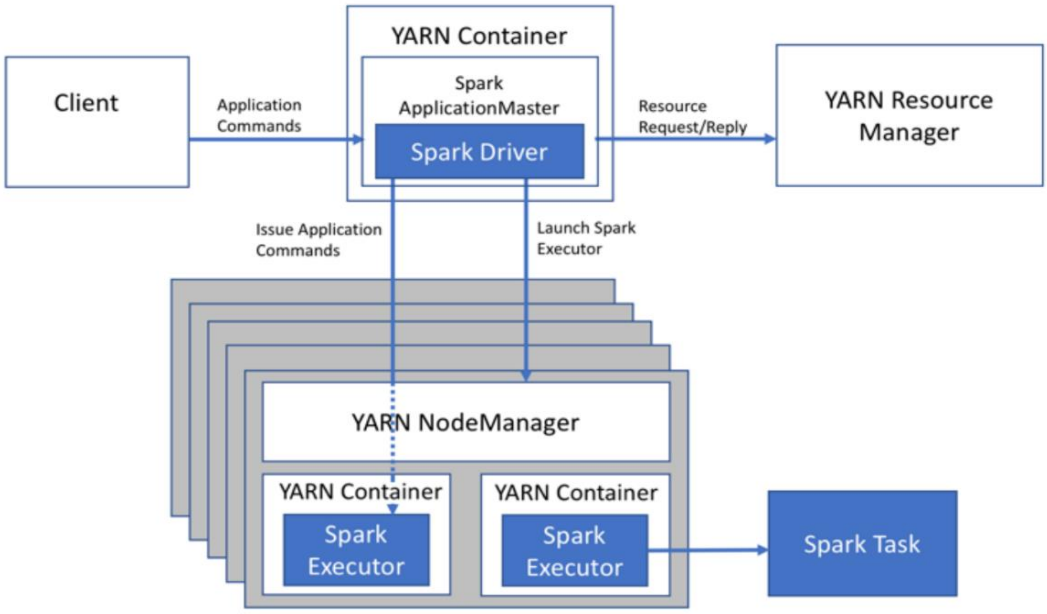
SparkContext có thể làm việc với nhiều Cluster Manager khác nhau:

- **Standalone Cluster Manager:** Trong chế độ này, chỉ có duy nhất một Executor chạy tác vụ trên mỗi Woker node với một Master node duy nhất. Client thiết lập kết nối với Master node, yêu cầu tài nguyên tính toán và bắt đầu quá trình thực thi trên các Woker node. Đây là phương pháp đơn giản nhất và cung cấp gần như đầy đủ các tính năng như các chương trình Cluster Manager khác, nếu như không có yêu cầu gì thêm ngoài Spark.



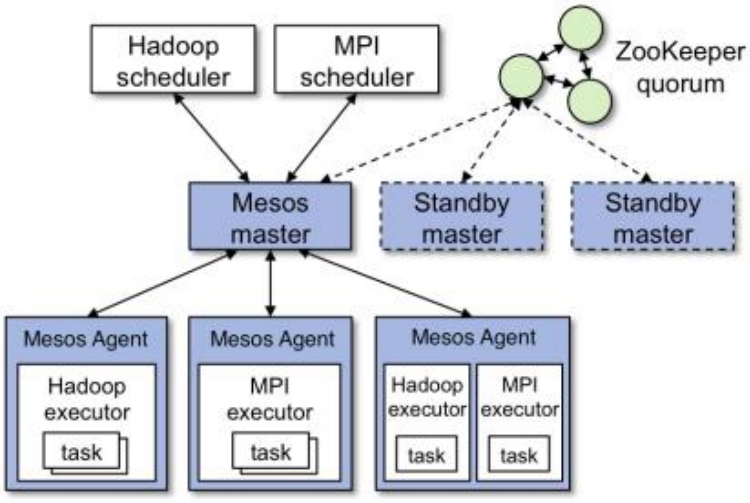
Hình 2.7: Spark trong chế độ Standalone Cluster Manager

- **YARN (Yet Another Resources Negotiator):** YARN là trình quản lý tài nguyên của Hadoop, chạy dựa trên trình quản lý tập tin HDFS. Khi Spark được chạy cùng với YARN, Resource Manager đảm nhận vai trò của Spark Master và NodeManager hoạt động như là executors. Mỗi Spark executor được chạy như là một container trên YARN. Chi tiết được mô tả trong Hình 1.8 bên dưới:



Hình 2.8: Spark trong chế độ hoạt động với YARN

- **Apache Mesos:** Apache Mesos dựa trên kiến trúc master-agents. Kiến trúc của Mesos ở trên Hình 1.9, bao gồm master deamon có nhiệm vụ quản lý các agent daemon chạy trên các node trong cụm xử lý. Khi Spark hoạt động với Mesos, Mesos master sẽ thay thế Spark master trong việc quản lý và phân bổ tài nguyên và tác vụ tính toán trên các node trong cụm. Khi Spark driver tạo một công việc tính toán mới và bắt đầu tạo ra các tác vụ để định thời, Mesos xác định các máy tính có thể đảm trách việc xử lý các tác vụ đó.

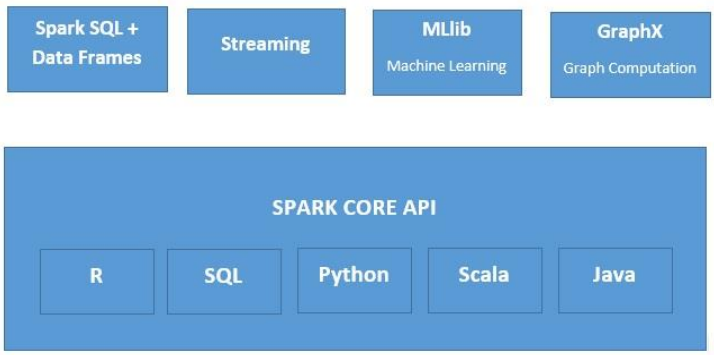


Hình 1.9: Kiến trúc của Apache Mesos

- **Kubernetes:** Spark cũng có thể chạy trên Kubernetes. Spark driver được chạy trên một pod của Kubernetes và các executor cũng được triển khai trên các Kubernetes pod. Tuy nhiên, việc chạy Spark trên Kubernetes vẫn trong quá trình thử nghiệm.

Bên cạnh đó, một trong những đặc điểm quan trọng của Spart khiến nó dẫn đầu trong xử lý Big Data là việc hỗ trợ nhiều thư viện xử lý trong hệ sinh thái:

- **Spark SQL:** Là một module trong Spark, giúp tích hợp xử lý các quan hệ trên dữ liệu với các API lập trình hàm trên Spark. Spark SQL hỗ trợ truy vấn dữ liệu cả qua SQL và qua Hive Query Language. Dữ liệu trên Spark được tăng cường tính trừu tượng (abstraction) với DataFrame và Dataset API từ Spark SQL.
- **Spark Streaming:** Dùng để xử lý các luồng dữ liệu theo thời gian thực.
- **MLlib:** Là các thư viện hỗ trợ cho các tác vụ học máy (machine learning) trên Spark
- **GraphX:** Là các API hỗ trợ cho biểu diễn đồ thị và tính toán song song trên đồ thị.



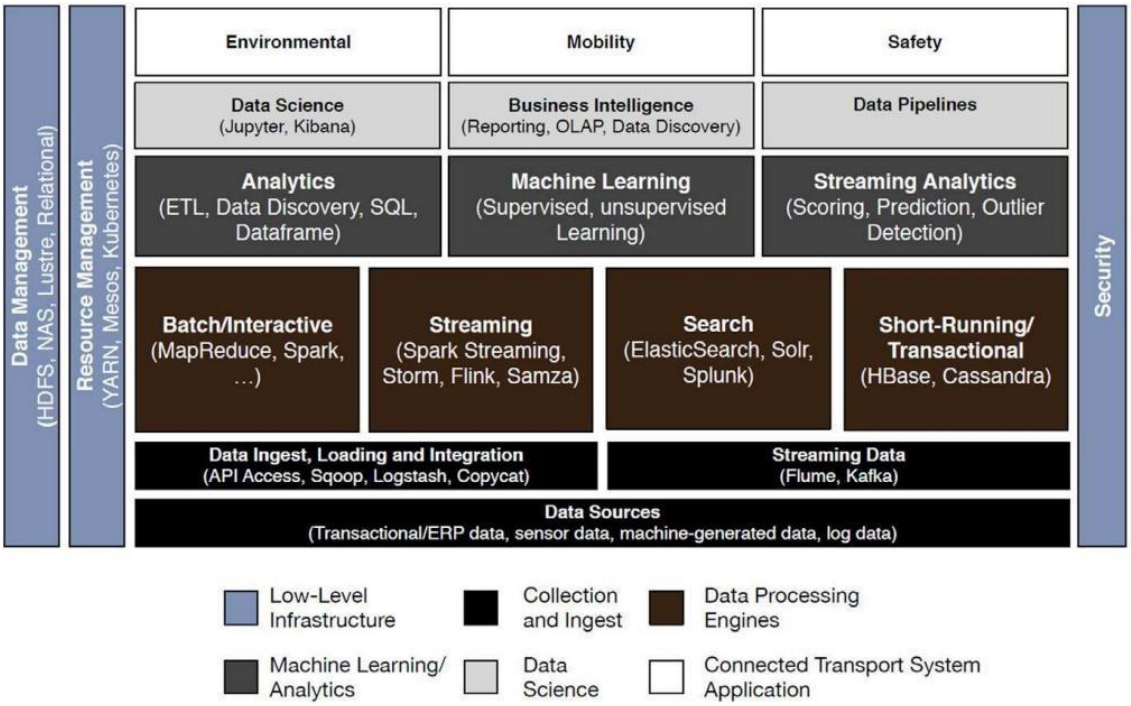
Hình 2.10: Hệ sinh thái Spark

2.3. Tình hình nghiên cứu trong nước

Đối với các hệ thống giao thông thông minh (Intelligence Transport System - ITS) hiện đại ngày nay, số lượng dữ liệu được sinh ra ngày càng tăng do các hệ thống ITS này được kết nối vô số cảm biến. Các cảm biến này có thể được lắp đặt trên các phương tiện giao thông di chuyển trên đường (thiết bị giám sát hành trình) hay là các hệ thống camera giám sát trên đường, bảng báo điện tử, thiết bị di động, v.v. Để giải quyết dữ liệu phức tạp, các hệ thống ITS cần phải được xây dựng trên hạ tầng dữ liệu hiện đại, có khả năng lưu trữ và xử lý khối lượng lớn dữ liệu. Sở Giao thông Vận tải Hoa kỳ giới thiệu tổng quan về một kiến trúc tham chiếu để triển khai các hệ thống giao thông thông minh gọi là Connected Transport System Reference Implementation Architecture (ARC-IT). ARC-IT là một giải pháp hoàn chỉnh được hình thành với sự đóng góp của cộng đồng ITS (cán bộ giao thông vận tải, kỹ sư hệ thống, nhà phát triển hệ thống, chuyên gia công nghệ, nhà tư vấn, v.v.). ARC-IT là một kiến trúc tham chiếu cung cấp nền tảng chung cho các nhà lập kế hoạch và kỹ sư có liên quan trong một dự án ITS; trong đó mọi người sử dụng cùng chung một ngôn ngữ làm cơ sở để lập kế hoạch, lập trình và triển khai các hệ thống ITS. Trong một nghiên cứu về hạ tầng dữ liệu dành riêng cho các hệ thống ITS, nhóm tác giả trình bày kiến trúc về một hệ thống giao thông tích hợp dựa trên nền tảng Data Lake. Hình 1.11 trình bày kiến trúc 6 lớp của hệ thống giao thông tích hợp. Trong bài nghiên cứu, kiến trúc này được phân giải ở mức có thể triển khai được và dựa trên hệ sinh thái Apache Spark.

Trong nhiều năm gần đây, yêu cầu phát triển đô thị thông minh được lãnh đạo nhiều tỉnh/thành phố rất quan tâm. Nhiều dự án thí điểm về đô thị thông minh được xây dựng và triển khai như tại Bình Phước, Bắc Kạn, Thanh Hoá, v.v. Tuy nhiên bài toán đô thị thông minh là bài toán khó, nên các dự án thí điểm này chủ yếu tập trung vào một số nội dung chính như:

- Chính quyền điện tử
- Xây dựng trung tâm IOC cho đô thị
- Xây dựng giải pháp CCTV giám sát đô thị



Hình 2.11: Kiến trúc 6 tầng của một hệ thống giao thông tích hợp

Riêng đối với bài toán giao thông, đa phần các dự án dừng ở mức giám sát và phát hiện các hành vi vi phạm bằng thủ công hoặc bán tự động để xử lý phạt nguội. Khi số lượng camera bùng nổ dẫn đến nhu cầu giám sát giao thông tự động bằng phần mềm trí tuệ nhân tạo tăng cao. Tại Việt Nam có nhiều nhóm nghiên cứu, công ty xây dựng các giải pháp ứng dụng trí tuệ nhân tạo trong giám sát giao thông như Đại học Bách Khoa, Đại học quốc gia Thành phố Hồ Chí Minh, Viettel, VNPT v.v. Các ứng dụng này chủ yếu khai thác các mô hình trí tuệ nhân tạo để phân tích tự động hình ảnh camera. Đối với bài toán tích hợp và lưu trữ, bất kỳ một trong những đặc điểm trên của dữ liệu ITS đều có thể tạo ra thách thức đối với các hệ quản trị cơ sở dữ liệu truyền thống và một số đặc điểm là không thể xử lý được đối với các hệ thống lưu trữ dữ liệu truyền thống. Do đó, để đối phó với bài toán thu thập, tích hợp, lưu trữ dữ liệu giao thông cần khai thác tối đa sức mạnh của công nghệ trong đó phải kể đến các công nghệ về xử lý dữ liệu lớn. Vì vậy bài toán thu thập, tích hợp, lưu trữ dữ liệu giao thông là một bài toán khó. Để xây dựng được giải pháp giải quyết bài toán khó này đòi hỏi phải khai thác được triệt để sức mạnh công nghệ tiên tiến đồng thời kết hợp

được kiến thức sâu chuyên môn về giao thông vận tải, đặc biệt sự am hiểu về giao thông đặc thù của đô thị.

2.4. Cơ sở lý luận

Hệ thống giao thông là một dạng hệ thống phức tạp được cấu thành từ nhiều thành phần từ phần cứng thiết bị, đường truyền viễn thông, hạ tầng máy chủ, phần mềm ứng dụng… Các hệ thống ITS trong quá trình vận hành sinh ra một lượng dữ liệu khổng lồ. Đặc điểm của dữ liệu này có thể được mô tả gói gọn trong 5 tính chất "5V" của dữ liệu lớn: (1) Volume – dung lượng dữ liệu, Variety - đa dạng, (3) Velocity – tốc độ, (4) Veracity – tính xác thực, và (5) Value - giá trị.

- *Volume* - Dung lượng dữ liệu do hệ thống ITS tạo ra tăng theo cấp số. Với số lượng ngày càng tăng của các ứng dụng công nghệ giám sát giao thông cho phép việc thu thập dữ liệu ngày càng đa dạng và phức tạp. Vì thế lượng dữ liệu liên quan đến giao thông được tạo ra mỗi giây. Ví dụ, 500 camera của hệ thống CCTV ở thành phố London tạo ra 1,2 Gbps.
- *Variety* - Dữ liệu được thu thập ở nhiều định dạng và theo nhiều phương cách khác nhau. Mức độ định dạng của dữ liệu này cũng có thể thay đổi từ dữ liệu bán cấu trúc (ví dụ: nhật ký sửa chữa, hình ảnh, video và tệp âm thanh) đến dữ liệu có cấu trúc (ví dụ: dữ liệu từ hệ thống cảm biến, dữ liệu sự cố giao thông, v.v…). Các bộ dữ liệu khác nhau có các định dạng khác nhau về kích thước tệp, độ dài bản ghi và lược đồ mã hóa; nội dung của chúng có thể đồng nhất hoặc không đồng nhất. Các tập dữ liệu không đồng nhất này, được tạo ra bởi các nguồn khác nhau ở các định dạng khác nhau, đã đặt ra những thách thức đáng kể cho việc tích hợp và phân tích.
- *Velocity* – Sự đa dạng nguồn thu thập dữ liệu và yêu cầu giám sát thực tiễn về giao thông đã kéo theo tốc độ tạo ra dữ liệu lớn. Trong đó nhiều dữ liệu được thu thập liên tục, theo thời gian thực, trong khi các dữ liệu khác được thu thập định kỳ. Ví dụ các dữ liệu cảm biến lắp đặt trên phương tiện giao thông hay dữ liệu giám sát camera sẽ được thu thập thời gian thực với tần suất có thể tính

theo đơn vị giây. Tuy nhiên các dữ liệu về bản đồ có thể được cập nhật định kỳ trong khoảng thời gian dài hơn.

- *Veracity* - Được sử dụng để mô tả tính chắc chắn hoặc độ tin cậy của dữ liệu ITS. Ví dụ: bất kỳ quyết định nào được đưa ra từ một dữ liệu giao thông nào đó phải được phân tích dựa trên tính toàn vẹn của nguồn dữ liệu, nghĩa là, phải có các hiệu chuẩn chính xác dữ liệu truyền về từ cảm biến và có những giải thích chính xác về bất kỳ dữ liệu bị thiếu.
- *Value* – Đề cập đến vấn đề thời gian thu thập dữ liệu, tốc độ lấy mẫu của dữ liệu. Ví dụ, dữ liệu cũ vài giờ có thể không có giá trị đối với ứng dụng phân tích kẹt xe, nhưng có thể hữu ích trong ứng dụng quy hoạch giao thông. Như vậy hầu hết dữ liệu giao thông được thu thập đều có giá trị phục vụ cho bài toán phân tích giao thông, dữ liệu càng được làm mới, lưu trữ trong khoảng thời gian càng dài thì càng có giá trị.

Do đó, để đối phó với bài toán thu thập, tích hợp, lưu trữ dữ liệu giao thông cần khai thác tối đa sức mạnh của công nghệ trong đó phải kể đến các công nghệ về lưu trữ và xử lý dữ liệu lớn.

2.5. Lý thuyết về các kiến trúc và thuật ngữ

Hệ lưu trữ và phân tích dữ liệu trung tâm dựa trên các giải pháp về dữ liệu lớn. Truy xuất dữ liệu lớn không chỉ đơn giản là sao chép dữ liệu, nó là một công đoạn rất phức tạp và quan trọng để đảm bảo những dữ liệu được truy xuất có khả năng truy cập, trao đổi thông tin, tái sử dụng nhiều lần. Vấn đề đặt ra cho cho giải pháp kết nối là bảo toàn được cấu trúc và sự nhất quán của dữ liệu ở đầu vào của hệ thống lưu trữ.

2.5.1. Data Warehouse

Nhà kho dữ liệu (Data Warehouse - Hình 1.12) là khái niệm được giới thiệu lần đầu vào năm 1988 bởi 2 nhà nghiên cứu của IBM là Barry Devlin và Paul Murphy. Đây là nơi lưu trữ dữ liệu bằng thiết bị điện tử của một tổ chức, doanh nghiệp, nhằm hỗ trợ việc phân tích dữ liệu và lập báo cáo. Về cơ bản, có thể hiểu, Data Warehouse

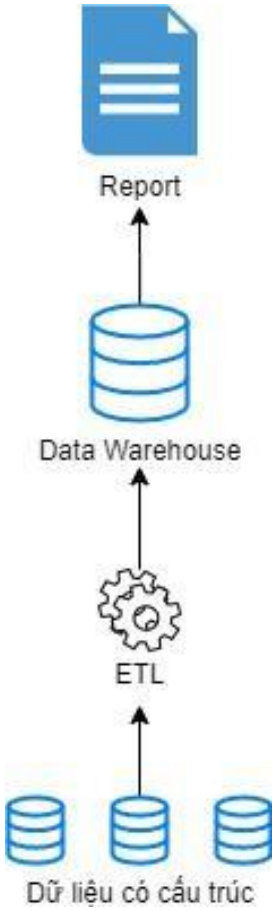
là một tập hợp dữ liệu hoặc thông tin có chung một chủ đề, được tổng hợp từ nhiều nguồn khác nhau trong nhiều mốc thời gian. Quá trình tập hợp và thao tác trên dữ liệu mang đặc tính ACID:

- Atomicity (Tính nguyên tử): dữ liệu được tập hợp từ nhiều nguồn khác nhau -> khi tập hợp phải thực hiện làm sạch, sắp xếp, rút gọn dữ liệu.
- Consistency (Tính nhất quán): chỉ lấy những dữ liệu có ích (các dữ liệu có cùng chủ đề).
- Isolation (Tính cô lập): Các dữ liệu truy xuất không bị ảnh hưởng bởi các dữ liệu khác hoặc tác động lên nhau.
- Durable (Tính bền vững): Dữ liệu không thể tạo thêm, xóa hay sửa đổi.

Dữ liệu trong kho dữ liệu là rất lớn và không có những thao tác như sửa đổi hay tạo mới. Kho dữ liệu dựa trên cơ sở mô hình dữ liệu đa chiều, được mô hình hóa vào đối tượng gọi là datacube. Datacube bao gồm các dữ kiện(fact), và những dữ kiện này tạo ra nhiều dữ kiện khác nhau (dimension). Đối với kiến trúc Data Warehouse, dữ liệu có cấu trúc từ các database thông qua quá trình trích xuất, biến đổi sẽ được lưu vào "nhà kho". Dữ liệu trong nhà kho này sẽ được sử dụng để xuất báo cáo, trực quan cho người sử dụng Mục đích của kiến trúc Data Warehouse là phục vụ các yêu cầu phân tích, hoặc khai phá cụ thể được gọi là chủ đề. Ví dụ chủ đề giao thông có thể bao gồm tổng số phương tiện trong ngày, thống kê số lượng xe nội tỉnh/ngoại tỉnh hoặc mật độ phương tiện qua các nút giao thông.

- Ưu điểm:
  - Kiến trúc hệ thống xây dựng theo hướng phân tích thông tin quản trị thông minh BI (Business Intelligence) để có thể trình bày thông tin trên các báo cáo quản trị giúp điều hành doanh nghiệp. Đối tượng thụ hưởng hệ thống là những người phân tích thông tin và đưa ra các kế hoạch dài hạn hoặc điều hành ngắn hạn;
  - Là một kiến trúc mạnh về quản lý các chức năng ACID transaction.

- Nhược điểm:
  - Không hỗ trợ những liệu có đặc tính phi cấu trúc, bán cấu trúc;
  - Hạn chế trong việc đáp ứng dữ liệu biến đổi nhanh về mặt số lượng;
  - Chi phí cao cho việc lưu trữ những tập dữ liệu lớn;
  - Hầu như không hỗ trợ cho công việc liên quan đến khoa học dữ liệu và học máy do các công cụ không có khả năng đọc trực tiếp dữ liệu mà phải thông qua truy vấn SQL.



Hình 2.12: Hệ thống theo kiến trúc Data Warehouse

Nhà kho dữ liệu (Data Warehouse) giúp các nhà lãnh đạo có được những phân tích chuyên sâu bằng việc thu thập dữ liệu từ những cơ sở dữ liệu vận hành về một nhà kho, nơi mà sau này có thể được sử dụng như một nguồn thông tin hỗ trợ việc ra quyết định và các tác vụ phân tích kinh doanh (Business Intelligence). Dữ liệu trong

kho có thể được ghi lại bằng phương pháp schema-on-write, có nghĩa là mẫu dữ liệu được tối ưu hóa cho luồng tiêu thụ. Đây là thế hệ đầu tiên của các nền tảng phân tích dữ liệu Khoảng một thập kỷ trước, hệ thống thế hệ đầu tiên sử dụng Data Warehouse bắt đầu đối mặt với nhiều thách thức mới:

- Thứ nhất đó là chúng kết hợp giữa tính toán và lưu trữ trên cùng một máy. Điều này khiến cho doanh nghiệp càng phải chi nhiều tiền hơn khi dữ liệu phình to ra.
- Thứ hai, ngoài việc dữ liệu phình to về mặt kích thước, chúng còn trở nên đa dạng về mặt chủng loại, và hầu hết là những dữ liệu không cấu trúc. Đặc biệt là những dữ liệu dạng tài liệu văn bản, ảnh, audio… khiến cho Data Warehouse không còn khả năng lưu trữ và truy vấn nữa.
- Thứ ba, với việc dữ liệu thay đổi và phát triển quá nhanh, nhu cầu nhận được giá trị từ dữ liệu một cách nhanh chóng đã trở nên cấp bách hơn bao giờ hết. Hầu như các nhà lãnh đạo không đủ kiên nhẫn cho nhiều tháng phân tích và mô hình hóa dữ liệu cho mục đích sử dụng.

Vì nhu cầu phát sinh, hệ thống thế hệ thứ hai đã được nghiên cứu phát triển để giải quyết những khó khăn mà Data Warehouse gặp phải.

2.5.2. Data Lake

Vào thời điểm dữ liệu trong Data Warehouse có thể sử dụng được, nhu cầu về dữ liệu có thể đã thay đổi. Trong một nhánh tương tự như Data Warehouse, các kho dữ liệu cục bộ (data mart) nổi lên với một mục đích sử dụng cụ thể hoặc được phân loại theo một chất lượng nhất định. Data mart đã thành công hơn vì việc sử dụng dữ liệu được hiểu rõ hơn, và kết quả có thể được cung cấp nhanh hơn. Tuy nhiên, tính chất ngăn cách của các Data Mart đã khiến chúng trở nên ít hữu ích hơn đối với các bài toán có lượng dữ liệu khổng lồ, và cần sử dụng dữ liệu đó một cách đa chức năng

Vì lý do này, các hồ dữ liệu (Data Lake – Hình 1.13) đã phát triển do khả năng đáp ứng nhu cầu dữ liệu ở mọi quy mô. Chúng có thể tăng tốc mọi thứ, làm cho dữ

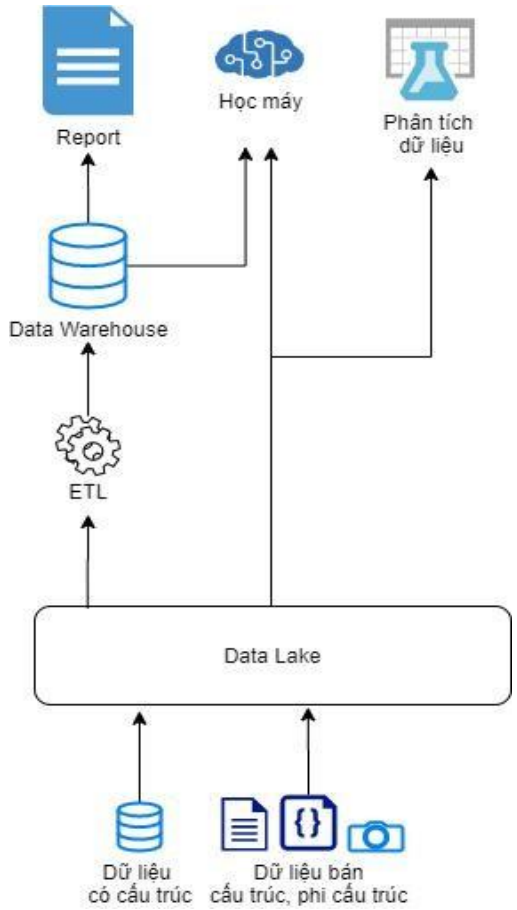
liệu dễ sử dụng hơn cho các nhu cầu chưa được xác định trước đó. Sự xuất hiện của điện toán đám mây quy mô lớn với sức mạnh tính toán khổng lồ và khả năng lưu trữ gần như vô hạn đã khiến phương pháp tiếp cận hồ dữ liệu này trở nên khả thi.

Mục đích cuối cùng của hồ dữ liệu là để giải quyết sự thất vọng mà hệ thống công nghệ thông tin gặp phải khi cố gắng tạo ra một kho lưu trữ dữ liệu chiến lược có tổ chức, để đưa ra những quyết định kinh doanh quan trọng. Việc sử dụng này có thể bao gồm từ việc phân tích dữ liệu để hiểu rõ hơn nhu cầu của người sử dụng, cho đến trí tuệ nhân tạo để giải quyết các thách thức trong thời gian thực.

- Ưu điểm:
  - Chi phí lưu trữ thấp để có thể lưu trữ tất cả những dữ liệu thô;
  - Dữ liệu có thể đọc trực tiếp từ các thư viện học máy của Apache Spark nhờ các định dạng mở Apache Parquet hay Apache ORC.
- Nhược điểm:
  - Nhiều phân hệ lưu trữ phát sinh khiến hệ thống trở nên phức tạp, gây khó khăn trong việc quản lý dữ liệu;
  - Phát sinh nhiều tác vụ ETL gây khó khăn trong việc quản lý lỗi;
  - Khó khăn trong việc đồng bộ dữ liệu giữa các tác vụ ETL và ELT.

Để giải quyết vấn đề của Data Warehouse, thế hệ hệ thống thứ hai đã ra đời, tải toàn bộ dữ liệu thô vào trong một hồ dữ liệu (Data Lake) với chi phí lưu trữ thấp, các API lưu trữ dữ liệu dưới định dạng mở như Apache Parquet, ORC. Cách tiếp cận này khởi đầu bằng sự chuyển dịch của hệ sinh thái Apache Hadoop, sử dụng HDFS để tối thiểu chi phí lưu trữ. Các hồ dữ liệu sử dụng kiến trúc schema-on-read cho phép sự linh động trong lưu trữ dữ liệu với chi phí thấp, nhưng bù lại đó là chất lượng dữ liệu và sự quản lý luồng lưu trữ. Trong kiến trúc này, một phân hệ của dữ liệu sẽ thông qua quá trình ETL và lưu trữ xuống nhà kho dữ liệu, sử dụng cho việc hỗ trợ ra quyết định và các ứng dụng BI. Việc sử dụng các định dạng mở như Parquet, ORC

cũng mở ra một phương hướng mới cho các cơ chế phân tích khác, điển hình là các hệ thống học máy.



Hình 2.13: Hệ thống theo kiến trúc Data Lake

Từ năm 2015 trở đi, các hồ dữ liệu đám mây như Amazon S3 bắt đầu thay thế dần HDFS. Cùng với sự phát triển của rất nhiều nền tảng lưu trữ kho dữ liệu mới như Amazon Redshift, kiến trúc hai tầng Data Lake + Data Warehouse đã thống trị nền công nghiệp dữ liệu.

Hiện nay, kiến trúc hai tầng này lại gặp phải các thử thách. Điển hình như việc dữ liệu ban đầu phải được ETL trước khi đưa vào Data Lake để tránh rác dữ liệu, tiếp theo đó lại ETL để đưa vào kho dữ liệu. Điều đó dẫn đến sự phức tạp trong trong quản lý dữ liệu và quản lý luồng ETL, cùng với đó là sự trì hoãn và các nguy cơ hỏng dữ liệu ngay trên luồng ETL. Hơn thế nữa, các doanh nghiệp đang hướng đến các

phân tích bậc cao hơn như học máy hay học sâu, nơi cần đến dữ liệu của cả hồ dữ liệu và kho dữ liệu. Điều đó dẫn đến 4 vấn đề:

- Độ tin cậy dữ liệu: Mỗi một tác vụ ETL mà dữ liệu đi qua đều tăng khả năng xảy ra lỗi khiến suy giảm chất lượng của dữ liệu
- Độ linh hoạt dữ liệu: Vì kiến trúc hai tầng Data Lake + Data Warehouse vẫn chịu ảnh hưởng từ sự trì trệ của Data Warehouse. Với mỗi tác vụ tải dữ liệu từ Data Lake về Data Warehouse có thể mất cả ngày, và thường vận hành theo chu kì. Dựa vào một báo cáo của Dimensional Research và Five-tran, 86% các nhà phân tích sử dụng dữ liệu quá cũ và 62% các báo cáo đang chờ nguồn dữ liệu mỗi tháng.
- Hỗ trợ đối với các phân tích bậc cao: Một điều rõ ràng rằng nếu sử dụng dữ liệu trong Data Warehouse sử dụng cho học máy thông qua JDBC/ODBC là không hiệu quả. Cách giải quyết có thể là đọc trực tiếp từ định dạng lưu trữ hoặc xuất file, tuy nhiên như vậy lại bỏ qua các chức năng ACID Transaction và Indexing.
- Chi phí lưu trữ dữ liệu: vì có sự hiện diện của Data Warehouse, một điều rõ ràng rằng chi phí lưu trữ hiện tại sẽ tăng so với chỉ lưu trữ trong Data Lake.
- Quản lý được dữ liệu trong hồ dữ liệu: Kiến trúc mới cần có chức năng lưu trữ dữ liệu thô chi phí thấp song song với hỗ trợ các tác vụ ETL/ELT và các ACID Transaction để cải thiện chất lượng cũng như quản lý dữ liệu.
- Hỗ trợ học máy và khoa học dữ liệu: các nền tảng học máy Tensorflow có thể đọc được dữ liệu trực tiếp từ lưu trữ.
- Có hiệu suất truy vấn: Kiến trúc mới phải có hiệu năng truy vấn SQL tốt trên các định dạng mở Parquet/ORC.

2.5.3. Data Lakehouse

Vấn đề đầu tiên của người dùng cuối hiện nay đó là chất lượng dữ liệu và sự tin cậy của dữ liệu. Triển khai đường dẫn dữ liệu đúng và ổn định là rất khó. Với kiến trúc hai tầng Data Lake + Data Warehouse càng làm tăng sự phức tạp của việc triển khai. Lấy ví dụ, hệ thống Data Lake và Data Warehouse có thể lưu trữ dữ liệu với schema khác nhau, dẫn đến gia tăng số lượng các tác vụ ETL, tăng khả năng sinh ra lỗi.

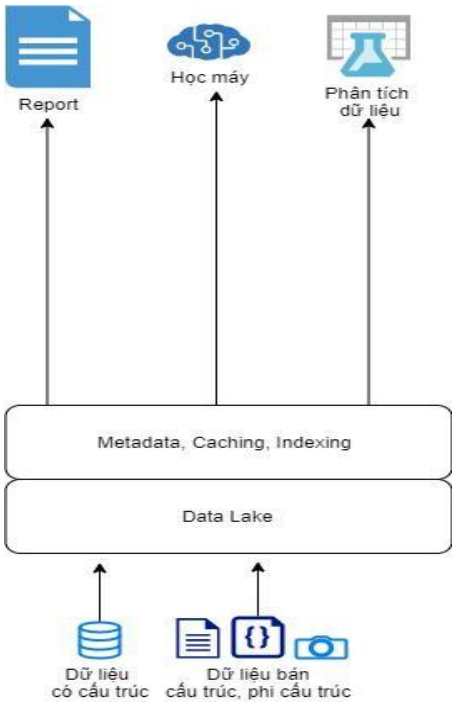
Vấn đề thứ hai, hiện nay nảy sinh rất nhiều nghiệp cần dữ liệu cập nhật liên tục. Các tổ chức thậm chí có thể triển khai nhiều hơn các luồng dữ liệu dòng (streaming data) cập nhật nhanh hơn vào Data Warehouse. Tuy nhiên, sự thiếu linh động của dữ liệu trong Data Warehouse có thể gây cản trở cho các hệ thống hỗ trợ khách hàng và các cơ chế gợi ý.

Vấn đề thứ ba, hiện nay một lượng lớn dữ liệu đến từ các nguồn là các dữ liệu phi cấu trúc, bao gồm hình ảnh, dữ liệu cảm biến, tài liệu. Các tổ chức ban ngành đoàn thể cần một hệ thống quản lý dữ liệu này một cách đơn giản nhất. Data Warehouse không hỗ trợ điểm này.

Vấn đề cuối cùng, đa số các tổ chức hiện nay đều đang triển khai các ứng dụng học máy và khoa học dữ liệu, và dữ liệu trong warehouse và lake không đáp ứng được những ứng dụng này. Lý do là vì những ứng dụng này cần xử lý một lượng lớn dữ liệu mà không sử dụng đến ngôn ngữ truy vấn SQL. Với những hệ thống phân tích bậc cao đang được phát triển liên tục, điều tất yếu việc cho chúng quyền truy cập trực tiếp dữ liệu trong một định dạng mở sẽ hiệu quả hơn việc thông qua ODBC hay JDBC. Bên cạnh đó, ứng dụng học máy và khoa học dữ liệu đều cần lượng dữ liệu chất lượng cao, nhất quán và cô lập. Vì thế, việc đem những tính năng của DBMS lên các định dạng lưu trữ trên Data Lake là cần thiết.

Xu hướng hiện tại vẫn đang xây dựng mô hình dữ liệu hai tầng Lake + Warehouse. Những công nghệ đầu tiên được phát triển đó là các định dạng lưu trữ như Parquet và ORC cùng với những hỗ trợ của chúng, cho phép người dùng có thể

truy vấn dữ liệu trong Data Lake với cùng một cơ chế SQL như trong Data Warehouse. Tuy nhiên chúng vẫn không làm cho dữ liệu trong Data Lake dễ quản lý hơn, và chúng cũng không thể giải quyết được sự thiếu linh động, phức tạp và những thách thức dữ liệu đến từ các phân tích bậc cao. Thứ hai là, đã có những đầu tư cho các cơ chế truy vấn có thể truy vấn trực tiếp từ Data Lake như Spark SQL, Trino, Hive hay AWS Athena. Tuy nhiên, những cơ chế này không giải quyết các vấn đề của Data Lake: thiếu hụt những tính năng quản lý dữ liệu như ACID Transaction hay những phương thức truy cập dữ liệu như indexing để đạt hiệu năng của Data Warehouse. Nhà hồ dữ liệu (Data Lakehouse - Hình 1.14) là một hệ thống quản lý dữ liệu dựa trên nền tảng lưu trữ chi phí thấp và các cơ chế truy cập dữ liệu mà trong đó các tính năng quản trị dữ liệu truyền thống từ các DBMS và các tính năng như ACID transaction, indexing, tối ưu truy vấn… được hiện thực. Nhà hồ dữ liệu kết hợp những tính năng chủ chốt của cả Data Lake và Data Warehouse: lưu trữ chi phí thấp trên các định dạng mở và tính năng quản lý và tối ưu dữ liệu. Cần lưu ý rằng một hệ thống Lakehouse cần có khả năng triển khai trên môi trường đám mây, đó là lưu trữ phân tán và xử lý song song.



Hình 2.14: Hệ thống theo kiến trúc Data Lakehouse

- Ưu điểm:
  - Giữ được các lưu trữ chi phí thấp của Data Lake;
  - Có khả năng quản trị và đảm bảo chất lượng dữ liệu của Data Warehouse;
  - Tinh gọn, giảm độ phức tạp của hệ thống;
  - Có thể truy cập dữ liệu trực tiếp sử dụng các công cụ Tensorflow.
- Nhược điểm:
  - Các công nghệ hiện tại vẫn còn chưa hoàn thiện;
  - Vẫn còn lấp lửng giữa lý thuyết và thực tế, chưa có một hệ thống hình mẫu điển hình nào.

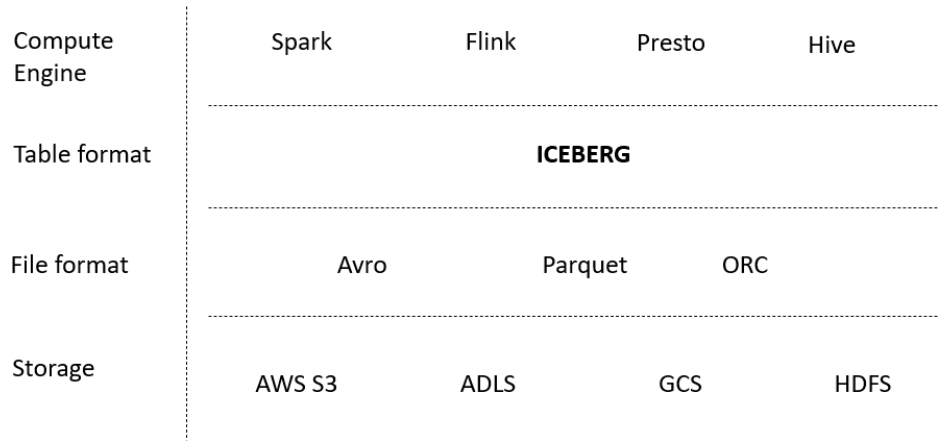
Kiến trúc Lakehouse được kì vọng sẽ giải quyết các vấn đề lớn mà mô hình hai lớp Data Lake và Data Warehouse, cho phép người sử dụng có một địa điểm tập trung duy nhất cho công việc phân tích dữ liệu, học máy cũng như xuất báo cáo. Đồng thời vẫn đảm bảo tốc độ nhanh chóng trong các truy vấn.

Bảng 2.1: So sánh giữa Data Warehouse, Data Lake và Data Lakehouse

	Data Warehouse	**Data Lake**	**Data Lakehouse**
Loại dữ liệu lưu trữ	Dữ liệu có cấu trúc	Dữ liệu bán cấu trúc và phi cấu trúc	Cả dữ liệu cấu trúc và phi cấu trúc
Tác vụ hỗ trợ	Sinh báo cáo	Học máy và phân tích dữ liệu	Phục vụ được cả hai tác vụ
Chi phí lưu trữ	Cao	Thấp	Thấp
ACID	Hỗ trợ	Không hỗ trợ	Hỗ trợ mức đọc ghi dữ liệu

2.5.4. Table Format

Các nền tảng lưu trữ đối tượng trên đám mây như Amazon S3 đang trở nên phổ biến rộng rãi, lưu trữ hàng exabytes dữ liệu cho hàng triệu khách hàng. Tuy rằng hầu hết các hệ thống đều hỗ trợ việc đọc ghi dữ liệu từ Cloud Object Storage, nhưng để đạt được hiệu quả trong việc kết nối và truy xuất dữ liệu từ các định dạng thuần túy như Parquet, ORC rất phức tạp và thử thách. Giải pháp được đề ra đó chính là một cơ chế lưu trữ định dạng bảng. Cơ chế này cho phép chúng ta tương tác với các Cloud Object thông qua tương tác API và metadata. Từ đó, những thách thức về độ tin cậy, sự nhất quán và hiệu năng của dữ liệu được giải quyết. Table Format có thể được hiểu là một giải pháp siêu dữ liệu cho các định dạng mở Apache Parquet/ORC. Mục tiêu chính của các Table Format chính là mang các chức năng CRUD và các ACID Transaction giống như database, lên các nền tảng lưu trữ trên Data Lake, từ đó hiện thực kiến trúc Data Lakehouse. Như hình 1.15, nhờ có các Table Format mà dữ liệu được lưu trữ dưới dạng file trên Data Lake được ánh xạ thành các bảng như trong cơ sở dữ liệu, cho chúng ta khả năng tương tác với các định dạng lưu trữ trên Data Lake như là trên database. Các Table Format sẽ cho phép sự tương tác từ các nền tảng phân tích dữ liệu như Apache Spark để lưu trữ và truy xuất dữ liệu sử dụng cho phân tích dữ liệu. Ngoài ra, trong bộ thư viện đi kèm các Table Format bao gồm các connector hỗ trợ kết nối đến các hệ thống lưu trữ dữ liệu có cấu trúc tập trung.

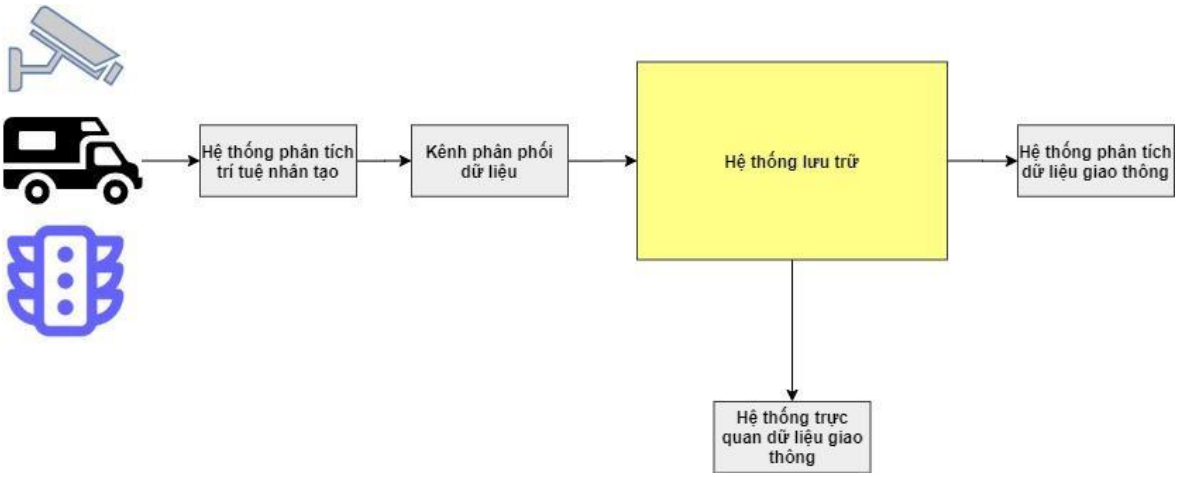


Hình 2.15: Vị trí của Table Format

CHƯƠNG 3: BÀI TOÁN VÀ GIẢI PHÁP CHO HỆ LƯU TRỮ VÀ TRUY VẤN DỮ LIỆU GIAO THÔNG

3.1. Mô tả bài toán

Do tính chất 5V của dữ liệu giao thông đã phân tích ở Chương 1 phần Cơ sở lý luận, hệ thống giám sát giao thông có lượng dữ liệu lớn, cụ thể là dữ liệu đo đếm phương tiện, cần được lưu trữ trên hạ tầng có khả năng lưu trữ, xử lý và truy vấn khối lượng lớn dữ liệu.



Hình 3.1: Hệ thống đo đếm và dự báo lưu lượng phương tiện giao thông

Trong lược đồ Hình 2.1, hệ thống phân tích trí tuệ nhân tạo xác định số lượng phương tiện đã được triển khai bao gồm ba chức năng:

(1) *Xác định số lượng phương tiện* tự động thông qua phân tích trực tiếp các luồng camera giao thông phát hiện ra số lượng các loại phương tiện đang di chuyển ngang qua điểm giao thông được giám sát. Hệ thống trả về số lượng phương tiện theo từng loại.

(2) *Đọc biển số phương tiện* thông qua phân tích trực tiếp các luồng camera phát hiện các biển số phương tiện đang lưu thông qua các nút giao thông. Kết quả trả về là thông tin biển số các phương tiện.

(3) *Dự báo số lượng phương tiện trong ngắn hạn* sử dụng phương pháp học máy truyền thống – cụ thể là mô hình Support Vector Regression.

Các kênh phân phối dữ liệu cũng được triển khai đồng thời với hệ thống phân tích trí tuệ nhân tạo, phục vụ nhu cầu của các ứng dụng khác trong hệ sinh thái giám sát giao thông chung. Cùng với đó, các ứng dụng phân tích và trực quan hóa thì tương đối phổ biến tại Việt Nam. Thành phần còn lại là một hệ thống có khả năng lưu trữ, xử lý dữ liệu lớn từ hệ thống phân tích trí tuệ nhân tạo, cũng như khả năng truy vấn hỗ trợ các ứng dụng phân tích và trực quan hóa được giải quyết trong luận văn. Các bài toán chính giải quyết trong luận văn như sau:

1. Giải pháp lưu trữ và truy vấn dữ liệu giao thông đô thị, cụ thể là dữ liệu đo đếm phương tiện giao thông.
2. Kỹ thuật nâng cao hiệu năng truy vấn.
3. Ứng dụng học máy vào trong công tác dự báo dữ liệu lưu lượng giao thông.

3.2. Các vấn đề phân tích để giải quyết bài toán

3.2.1. Phân tích đặc trưng dữ liệu thực tế

Căn cứ vào hệ thống đo đếm số lượng phương tiện, dữ liệu đầu vào có cấu trúc JSON bao gồm:

- Định danh CCTV mà hệ thống AI đã lấy dữ liệu hình ảnh và phân tích, thuộc kiểu chuỗi (string);
- Mốc thời gian dữ liệu được sinh ra, thuộc kiểu timestamp;
- Khoảng thời gian đo đếm, thuộc kiểu timestamp;
- Thông tin số lượng phương tiện theo từng loại mà hệ thống AI đã đếm được, thuộc kiểu mảng (array) của các json bao gồm:
  - Loại phương tiện thuộc kiểu string;
  - Số lượng phương tiện đếm được tương ứng: thuộc kiểu số (integer).

Số lượng của từng loại phương tiện được gom lại thành một chuỗi (array) 6 thành phần bao gồm: xe máy; xe bus; xe hơi; xe ô tô; xe tải; xe container. Dữ liệu đếm phương tiện được lấy từ hệ thống CCTV với 21 trạm quan sát, tốc độ sinh dữ liệu là 36k record/ngày.

Căn cứ vào hệ thống quan sát biển số phương tiện, dữ liệu đầu vào có cấu trúc JSON bao gồm:

- Định danh CCTV mà hệ thống AI đã lấy dữ liệu hình ảnh và phân tích, thuộc kiểu chuỗi (string);
- Thời gian sinh dữ liệu được sinh ra, thuộc kiểu timestamp;
- Thông tin về phương tiện đi kèm là biển số, thuộc kiểu struct_type bao gồm:
  - Loại phương tiện, kiểu string;
  - Biển số phương tiện, kiểu string.

Biển số của từng phương tiện sẽ đi kèm với loại phương tiện đó. Dữ liệu quan sát biển số phương tiện được lấy từ hệ thống CCTV với 24 trạm quan sát, tốc độ dữ liệu sinh ra từ hệ thống CCTV là 47k record/ngày.

3.2.2. Phân tích yêu cầu lưu trữ

- *Khả năng mở rộng:* Hệ thống lưu trữ cần có khả năng mở rộng để đáp ứng dữ liệu tăng lên liên tục về dung lượng và số lượng vì lượng dữ liệu do hệ thống phân tích trí tuệ nhân tạo sinh ra tăng theo cấp số. Với số lượng ngày càng tăng của các camera công nghệ giám sát giao thông, việc thu thập dữ liệu ngày càng đa dạng và phức tạp. Vì thế lượng dữ liệu đo đếm phương tiện giao thông được tạo ra mỗi giây;
- *Khả năng cập nhật nhanh:* Để hỗ trợ hệ thống trực quan có được những dữ liệu thời gian thực, hệ thống lưu trữ cần có khả năng cập nhật nhanh dữ liệu thời gian thực vì yêu cầu giám sát thực tiễn về giao thông đã kéo theo tốc độ tạo ra dữ liệu. Trong đó nhiều dữ liệu được thu thập liên tục, theo thời gian

thực. Ví dụ như dữ liệu camera sẽ được thu thập thời gian thực với tần suất có thể tính theo đơn vị giây;

- *Khả năng xử lý dữ liệu:* Để có thể xuất những báo cáo cho người dùng cuối, hệ thống lưu trữ cần đáp ứng khả năng chuẩn hóa dữ liệu, đưa về định dạng cấu trúc nhất định đúng với yêu cầu đặt ra. Lí do là vì mức độ định dạng của dữ liệu giao thông cũng có thể thay đổi từ dữ liệu bán cấu trúc đến dữ liệu có cấu trúc. Trong tương lai, nếu tích hợp với các hệ thống đo đếm khác trên địa bàn thành phố, các bộ dữ liệu có thể có các định dạng khác nhau về kích thước tệp, độ dài bản ghi và lược đồ mã hóa; nội dung của chúng có thể đồng nhất hoặc không đồng nhất;
- *Khả năng quản trị dữ liệu:* Hệ thống lưu trữ cần có khả năng quản lý dữ liệu, để dữ liệu giữa các tác vụ biến đổi ổn định và nhất quán, không gây ra tình trạng dư thừa, sai lệch dữ liệu. Nói cách khác, đó chính là tính năng ACID Transaction trong Data Warehouse.

3.2.3. Phân tích yêu cầu truy vấn

Giải pháp truy vấn về dữ liệu có nhiều yếu tố cần đánh giá. Trong luận văn này quan tâm đến giải pháp truy vấn dữ liệu đo đếm phương tiện giao thông nên hướng đến hai tham số sau:

- *Thời gian đáp ứng:* Các báo cáo thời gian thực gặp vấn đề về độ trễ truy vấn dữ liệu. Ví dụ, dữ liệu cũ vài giờ không có giá trị đối với ứng dụng cảnh báo kẹt xe. Như vậy dữ liệu càng được làm mới, tốc độ truy vấn càng nhanh thì báo cáo càng có giá trị. Hệ thống lưu trữ cần hỗ trợ khả năng truy vấn độ trễ thấp để đáp ứng nhu cầu của các báo cáo này;
- *Truy cập dễ dàng:* bên cạnh đó, hệ thống lưu trữ sẽ cần hỗ trợ khả năng truy cập trực tiếp dữ liệu để phục vụ cho các công việc về học máy và phân tích dữ liệu cấp cao. Các format sử dụng để lưu trữ trong hệ thống tất yếu phải khả dĩ truy cập trực tiếp dữ liệu bằng các công cụ học máy mà không thông qua các tác vụ truy vấn SQL.

3.2.4. Dự báo lưu lượng giao thông ngắn hạn

Đối với bài toán dự báo giao thông thì xét về cơ bản đây là bài toán dự báo dữ liệu chuỗi thời gian. Do vậy, chúng ta có thể áp dụng các phương pháp dự báo truyền thống như các mô hình tự hồi qui (ví dụ như ARIMA, SARIMA, VAR, ARCH, …), trung bình trượt, làm trơn hàm mũ, . . . Các mô hình truyền thống đều giả sử mối quan hệ cụ thể (thông thường là giả sử các biến độc lập và phụ thuộc có mối quan hệ tuyến tính) giữa các biến độc lập (còn được gọi là đặc trưng) và biến phụ thuộc, từ đó ước lượng các tham số của mô hình bằng cách khớp dữ liệu theo các mối quan hệ giả định trước, sử dụng phương pháp bình phương tối thiểu thông thường (odinary least squares - OLS). Tuy vậy, các mô hình dựa trên OLS như thế không xử lý được các vấn đề dữ liệu có số chiều lớn, đa cộng tuyến, quan hệ phi tuyến, tự động trích chọn đặc trưng phù hợp. Các mô hình nhân tố, ví dụ như mô hình nhân tố động (dynamic factor model), có thể xử lý dữ liệu có chiều cao và đa cộng tuyến bằng các phương pháp thu giảm chiều dữ liệu, nhưng vẫn chưa xử lý được vấn đề phụ thuộc phi tuyến và trích chọn đặc trưng phù hợp.

Trong luận văn sẽ ứng dụng mô hình Support Vector Regression để dự báo ngắn hạn thông tin lưu lượng. Luận văn sử dụng hệ số tương quan Pearson R để đánh giá độ chính xác của mô hình dự báo – với mong muốn R^2 có giá trị lớn hơn 0.8.

Dữ liệu sử dụng cho mô hình dự báo là dữ liệu lưu lượng xe máy thu thập tại một camera trong khoảng thời gian từ *05/05/2022* đến *23/05/2022*. Do tính chất bài toán đặt ra là dự báo ngắn hạn nên dữ liệu sử dụng huấn luyện mô hình không cần phải quá lớn. Kích thước dữ liệu gồm *26.199* mẫu dữ liệu, mỗi mẫu dữ liệu cách nhau một phút. Các dữ liệu ở các thời điểm bị thiếu sẽ được nội suy bằng cách gán bằng với giá trị dữ liệu gần kề trong quá khứ. Việc chuẩn hóa dữ liệu sẽ được chuẩn hóa bằng chuẩn hóa min-max. Bộ dữ liệu sẽ được phân chia làm 3 phần: 60% dùng để training, 20% dùng để validating và 20% dùng để testing.

3.3. Đề xuất giải pháp cho hệ lưu trữ, truy vấn

Mục tiêu của chúng ta là thiết kế và hiện thực một hệ thống lưu trữ dữ liệu tập trung theo hướng tiếp cận của kiến trúc Data Lakehouse, để hệ thống đạt được khả nang quản trị và xử lý dữ liệu. Ngoài ra, mục tiêu theo đuổi các công nghệ cloud-native cũng cần được xem xét cho khả năng triển khai mở rộng hệ thống dễ dàng. Ý tưởng đầu tiên để triển khai một Data Lakehouse đó là triển khai một hệ thống lưu trữ dữ liệu chi phí thấp (điển hình như Amazon S3 hoặc HDFS) sử dụng các chuẩn định dạng dữ liệu như Apache Parquet, kèm theo đó là một lớp siêu dữ liệu mà trong đó các bảng dữ liệu được định nghĩa trên những đối tượng lưu trữ. Điều này cho phép hệ thống triển khai những tính năng quản lý như ACID transaction mà vẫn lưu trữ dữ liệu ở hệ thống lưu trữ chi phí thấp, bên cạnh đó cho phép người dùng có thể đọc trực tiếp dữ liệu từ nơi lưu trữ. Tuy rằng lớp siêu dữ liệu hỗ trợ cho khả năng quản lý dữ liệu, nhưng thời gian đáp ứng truy vấn vẫn chưa thực sự tốt. Data Lakehouse xây dựng dựa trên định dạng lưu trữ hiện tại không dễ dàng đạt được hiệu năng truy vấn như Data Warehouse. Tuy vậy, chúng ta vẫn có rất nhiều cách để tối ưu khả truy vấn dữ liệu từ Lakehouse, tiềm năng sẽ đạt được hiệu năng tiệm cận với Data Warehouse.

Cuối cùng, Data Lakehouse hỗ trợ quản lý dữ liệu và tăng tốc xử lý của các phân tích bậc cao nhờ vào sự hỗ trợ đến từ các nền tảng như TensorFlow hay Spark MLlib đã có thể đọc những định dạng file như Apache Parquet.

Hệ thống Lakehouse sẽ có các thành phần chính:

- Hệ thống file phân tán được sử dụng để lưu trữ các dữ liệu
- Hệ quản lý siêu dữ liệu đóng vai trò cầu nối giữa các khối trong hệ thống
- Hệ xử lý phân tích dữ liệu được sử dụng để thực hiện các thao tác của dữ liệu
- Hệ truy vấn dữ liệu được sử dụng để thực hiện các tác vụ SQL
- Table Format được đề cập ở phần 1.5.4 là thành phần quan trọng nhất, mang tính năng quản trị và xử lý dữ liệu lên hệ thống.

3.3.1. Giải pháp công nghệ

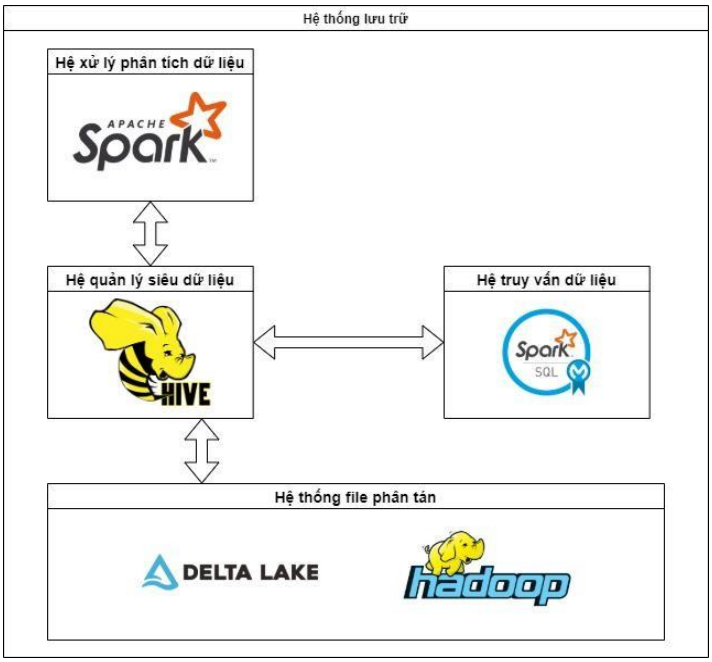
Giải pháp Delta + HDFS

Apache Spark là một công cụ phân tích dữ liệu phân tán, sử dụng cho cơ chế truy xuất và lưu trữ dạng batch và stream của dữ liệu được phát triển để loại bỏ giới hạn của cơ chế Map Reduce. Đây là nơi các tác vụ ETL/ELT được thực thi. Hive Metastore sử dụng cho việc lưu trữ Metadata, các catalog của Table Format. Nó chứa rất nhiều thông tin về các bảng, và cả địa chỉ lưu trữ các phân vùng dữ liệu. ETL là các script được viết bằng Python và chạy bằng cơ chế Pyspark trong Apache Spark. Script sẽ tương tác trực tiếp với hệ lưu trữ phân tán thông qua URI cùng với các config, package được khai báo khi thực thi câu lệnh spark-submit. Hive-metastore đóng vai trò là lớp lưu trữ Metadata. Những thành phần này là cơ bản đối với một hệ thống Data Lakehouse. Trong luận văn này, Spark được chọn cho hệ xử lý dữ liệu và Hive-metastore được chọn hệ quản lý siêu dữ liệu vì nhiều giải pháp khác cũng chọn lựa giải pháp công nghệ tương tự do ưu điểm của Spark và Hive-metastore cùng một hệ sinh thái Apache Hadoop, đều có mã nguồn mở và việc triển khai kết nối dễ dàng. Việc chọn lựa giải pháp công nghệ cho hệ truy vấn và hệ thống file phân tán được phân tích ở các nội dung sau. Việc đưa chức năng quản lý dữ liệu lên Data Lake là một trong những tiêu chí để hiện thực một hệ lưu trữ Data Lakehouse. Để làm được điều đó, định dạng lưu trữ phụ thuộc rất nhiều vào lớp siêu dữ liệu. Các table format cung cấp các driver cho hệ thống xử lý dữ liệu Apache Spark cũng như các catalog với vai trò là siêu dữ liệu. Nhờ ứng dụng các table format trong lưu trữ, dữ liệu trong các định dạng mở Apache Parquet đã có thể được quản lý. Vì nền tảng xử lý dữ liệu được chọn trong hệ thống mà luận văn xây dựng là Apache Spark. Bài toán hiện tại cần một Table Format có thể tương thích cả hai nền tảng này. Vì vậy, Delta Lake là một lựa chọn phù hợp. Delta Lake là một Table Format mã nguồn mở, cung cấp khả năng xây dựng kiến trúc Data Lakehouse trên nền tảng Data Lake. Delta Lake cung cấp các ACID transaction trên các hạ tầng lưu trữ thông dụng như s3, s3a, HDFS… Ngoài ra, Delta Lake cung cấp rất nhiều connector cũng như các API có thể đọc trực

tiếp dữ liệu, đặc biệt là đối với ngôn ngữ Python. Vì vậy, Delta Lake rất thích hợp cho hệ thống này HDFS là một hệ thống file phân tán truyền thống và có những đặc điểm như:

- Hadoop framework cho phép người dùng nhanh chóng viết và kiểm tra các hệ thống phân tán. Đây là cách hiệu quả cho phép phân phối dữ liệu và công việc xuyên suốt các máy trạm nhờ vào cơ chế xử lý song song của các lõi CPU;
- Hadoop có thể phát triển lên nhiều server với cấu trúc master-slave để đảm bảo thực hiện các công việc linh hoạt và không bị ngắt quãng do chia nhỏ công việc cho các server slave được điều khiển bởi server master;
- Apache Spark có thể được tích hợp trong hệ sinh thái Hadoop. Nhờ đó, chúng ta có thể rút gọn công đoạn triển khai Apache Spark.

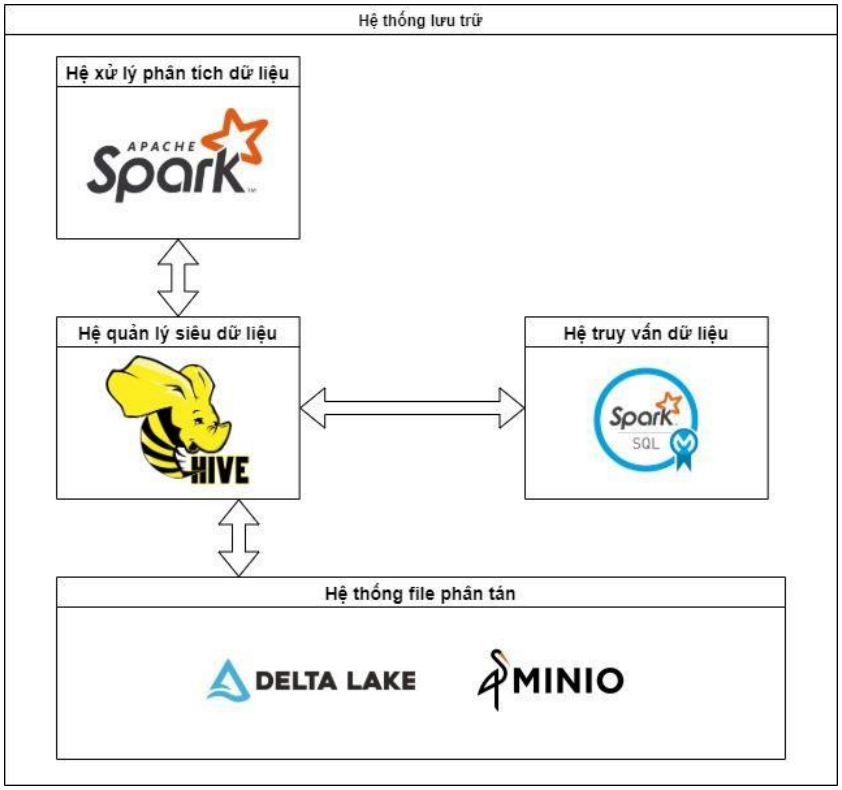
Vì các công nghệ đều nằm trong hệ sinh thái Hadoop rất phổ biến trong 10 năm trở lại đây khiến cho việc triển khai hệ thống trong trở nên đơn giản. Kể cả hệ truy vấn dữ liệu sử dụng cũng xuất phát từ Spark tích hợp với Hadoop. Tuy nhiên HDFS gặp một số vấn đề sẽ được phân tích ở phần sau.



Hình 3.2: Giải pháp Delta + HDFS

Giải pháp Delta + MinIO

HDFS và hệ sinh thái Hadoop có thể có chi phí thấp khi lưu trữ phân tán một lượng lớn dữ liệu. Tuy nhiên, trong một số kiến trúc cloud-native, những tiện ích mà HDFS mang lại là quá ít so với độ phức tạp trong vận hành. Vì thế, các tổ chức dần thay thế nền tảng S3 cho lưu trữ. MinIO là một nền tảng lưu trữ hiệu năng cao mã nguồn mở, cung cấp các API tương thích với nền tảng lưu trữ đám mây Amazon S3. So sánh với HDFS , MinIO có lợi thế về IO trong việc đọc dữ liệu.



Hình 3.3: Giải pháp Delta + MinIO

Với bối cảnh các tác vụ hiện tại đa phần chú trọng vào việc đọc dữ liệu, MinIO có lợi thế hơn so với HDFS. Vì được triển khai những public API giống như AWS S3 nên những ứng dụng có thể config để giao tiếp với Minio thì cũng có thể giao tiếp với AWS S3. Là một server lưu trữ object nên có thể được sử dụng để lưu trữ những dữ liệu phi cấu trúc như ảnh, video, log files, backups. Dung lượng của 1 object có

```
org.apache.hadoop.hdfs.BlockMissingException: Could not obtain
block: BP-467931813-10.3.20.155-1514489559979
```

thể dao động từ một vài KB tới tối đa là 5TB. File cũng được gom lại trong 1 buckets, nó là được chỉ cùng với access key khi dùng app. Thông qua quá trình thực nghiệm cho thấy được HDFS xảy ra lỗi do trong quá trình đọc ghi nếu sinh ra lỗi, HDFS có thể làm mất các pointer đến block pool dữ liệu.

Cùng với lỗi đến từ việc khi các tác vụ streaming liên tục ghi vào, HDFS bị nghẽn IO và khiến tác vụ ghi bị hỏng.

```
java.io. IOException: Unable to close files because the last
block does not have enough replicas
```

Sau khi chuyển qua MinIO, các lỗi trên không còn xuất hiện. MinIO cũng không bắt buộc các worker của spark phải kết nối với các datanode như HDFS. Tuy nhiên MinIO lại đòi hỏi khắt khe hơn khi deploy cluster, khi mà theo như yêu cầu cần một cluster gồm 4 máy theo như Erasure Code . Vì lí do đó, giải pháp sử dụng MinIO hướng đến các nền tảng cloud-native và sẽ được triển khai sử dụng công cụ Kurbernetes. Mặc dù nó được thiết lập đơn thuần chỉ là một thư mục hiện hành thông qua S3 API nhưng được thiết kế để đề phòng việc disk failures và coruption dữ liệu thông qua hệ thông dữ liệu phân tán giữa nhiều disks và nhiều node được host bởi MinIO.

Giải pháp Iceberg + MinIO + Trino

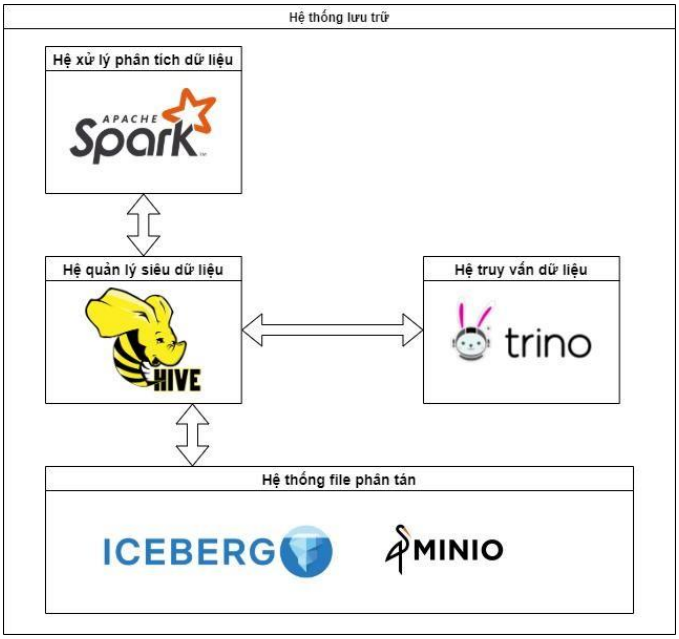
Trino có nguồn gốc từ Apache Presto, là một công cụ SQL phân tán mã nguồn mở. Presto bắt nguồn từ nhu cầu phân tích dữ liệu của Facebook, được tạo ra bởi Martin Traverso, David Phillips, Dain Sundstrom, và Eric Hwang vào năm 2012 khi còn làm việc tại Facebook. Presto được tạo ra mục đích để truy vấn 300 PB dữ liệu trong Hive Data Warehouse. Sau đó vào năm 2018, Martin, Dain, và David rời Facebook để xây dựng cộng đồng Presto Open Source với tên khác là PrestoSQL. Sau đó vào tháng 12 năm 2020, Presto được đổi tên thành Trino.Trino rất hữu ích để thực hiện các truy vấn thậm chí là hàng petabyte dữ liệu. Kiến trúc mở rộng và giao diện plugin lưu trữ rất dễ tương tác với các hệ thống tệp khác. Hầu hết các công ty công nghiệp tốt nhất hiện nay đang áp dụng Trino vì tốc độ tương tác và hiệu suất độ

trễ thấp. Spark SQL về cơ bản là một spark job được quản lý mặc định bởi Apache Spark, nó sẽ thiếu đi các thành phần caching và optimizing trong truy vấn. Trong khi đó, Trino là cơ chế truy vấn được tạo ra chuyên dùng cho truy vấn dữ liệu lớn, tốc độ kiểm nghiệm cho thấy hiệu năng truy vấn tốt hơn Spark SQL. Tuy nhiên, một số tác vụ Trino tương tác ở các Table Format vẫn chưa được hoàn thiện, vì thế nên cân nhắc sử dụng linh hoạt giữa Trino và Spark SQL. Cũng như Delta Lake, Iceberg là một table format lưu trữ mã nguồn mở, cung cấp khả năng xây dựng kiến trúc Data Lakehouse trên nền tảng Data Lake. Delta Lake cung cấp các ACID transaction trên các hạ tầng lưu trữ thông dụng như s3, s3a, HDFS. Hiện tại Iceberg vẫn chưa hỗ trợ nhiều connector và chưa hoàn thiện chức năng tải dữ liệu sử dụng ngôn ngữ Python, vì thế Iceberg đối với Machine Learning vẫn còn hạn chế. Tuy nhiên lợi thế của Iceberg so với Delta chính là Phân vùng ẩn (Hidden Partition). Khi dữ liệu đã được phân vùng, chúng ta muốn tận dụng những tiện ích mà phân vùng mang lại. Đối với dữ liệu được phân vùng trong Delta, chúng ta phải truyền vào điều kiện phân vùng. Ví dụ:

```
SELECT level, count(1) as count FROM logs
WHERE event_time
    BETWEEN '2018-12-01 10:00:00' AND '2018-12-01 12:00:00'
    AND event_date = '2018-12-01'
```

Trong câu truy vấn ở trên, dữ liệu đã được phân vùng theo event_date. Để có thể tăng tốc truy vấn, chúng ta phải truyền vào điều kiện *event_date = ’2018-12-01’*, điều này dẫn đến:

- Delta không thể xác thực phân vùng dữ liệu nếu như người viết câu truy vấn không cung cấp điều kiện phân vùng;
- Việc không cung cấp điều kiện phân vùng bắt buộc người viết câu truy vấn phải hiểu rõ cấu trúc dữ liệu và phân vùng của bảng lưu trữ. Nếu không, hiệu quả truy vấn sẽ giảm mạnh vì lúc này các cơ chế phải scan hết tập dữ liệu.



Hình 3.4: Giải pháp Iceberg + MinIO + Trino

Bên cạnh việc giảm hiệu quả truy vấn, nếu như cột phân vùng không hiện hữu trong cấu trúc dữ liệu ban đầu, chúng ta bắt buộc phải bổ sung bằng việc tùy biến giá trị phân vùng từ một cột được chọn và lưu lại thành cột mới. Dẫn đến việc format dữ liệu ban đầu bị thay đổi, có thể dẫn đến sự nhầm lẫn cho người dùng cuối. Iceberg Table Format cung cấp giá trị phân vùng bằng cách sử dụng giá trị của một cột và tùy biến nó. Việc chuyển đổi giá trị phân vùng sẽ hoàn toàn do các cơ chế trong Iceberg chịu trách nhiệm. Cơ chế của Iceberg không cần sinh ra cột mới, nên được gọi là Phân vùng ẩn. Như vậy, giá trị phân vùng luôn được sinh một cách chính xác và luôn được sử dụng trong các câu truy vấn mà không cần phải cung cấp điều kiện như Delta. Nói cách khác, đối với người dùng cuối Iceberg, họ không cần quan tâm cấu trúc bảng đã lưu mà vẫn có được câu truy vấn hiệu quả. Ngoài ra, vì không phụ thuộc vào layout vật lý của bảng, phân vùng dữ liệu có thể linh hoạt chuyển đổi sang giá trị khác mà không cần các migration phức tạp.

3.3.2. Kỹ thuật tối ưu

Hiệu năng truy vấn không chỉ phụ thuộc vào công cụ mà còn phụ thuộc việc tổ chức lưu trữ dữ liệu trên hệ thống lưu trữ. Các nội dung này sẽ được phân tích để

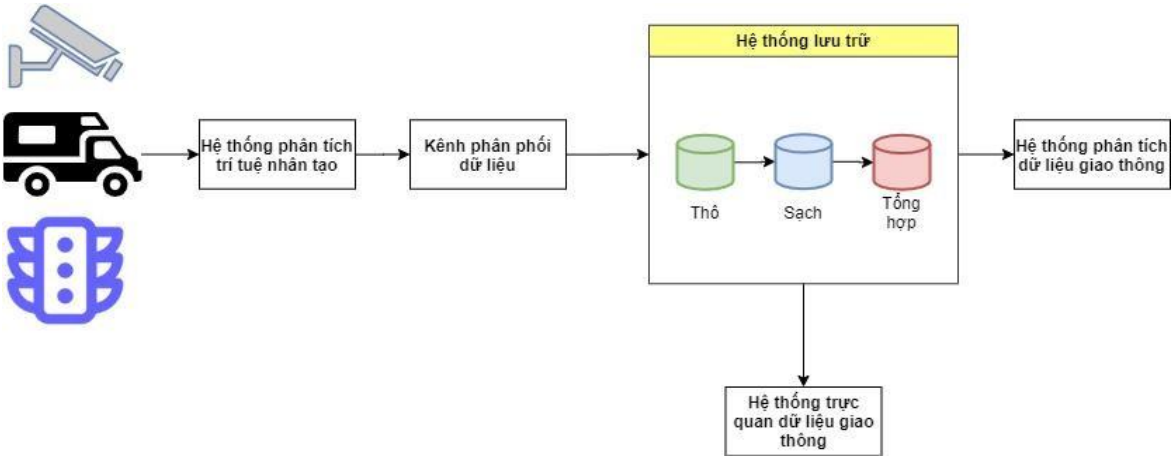
tìm giải pháp tổ chức lưu trữ hợp lý cho hệ thống lưu trữ dữ liệu đo đếm phương tiện giao thông. Bài toán về kỹ thuật nâng cao hiệu năng truy vấn sẽ được xem xét và giải quyết trong phần này. Ngoài ra, sự tiện lợi và tính sẵn sàng của dữ liệu cho dữ liệu đo đếm phương tiện giao thông cũng được xét đến.

Mô hình dữ liệu tam cấp

Dữ liệu được thu thập từ nhiều thông tin, và mỗi thông tin đều mang những giá trị khác nhau tùy theo từng loại bài toán. Vì thế, dữ liệu cần được trích xuất, sàng lọc và xử lý theo từng cấp độ, từng bài toán với mục đích đem lại sự tiện lợi và tính sẵn sàng cho dữ liệu. Một hệ thống lưu trữ được thiết kế tốt sẽ tận dụng được tối đa những thông tin mà dữ liệu mang lại.

Dữ liệu sẽ được lưu theo 3 tầng:

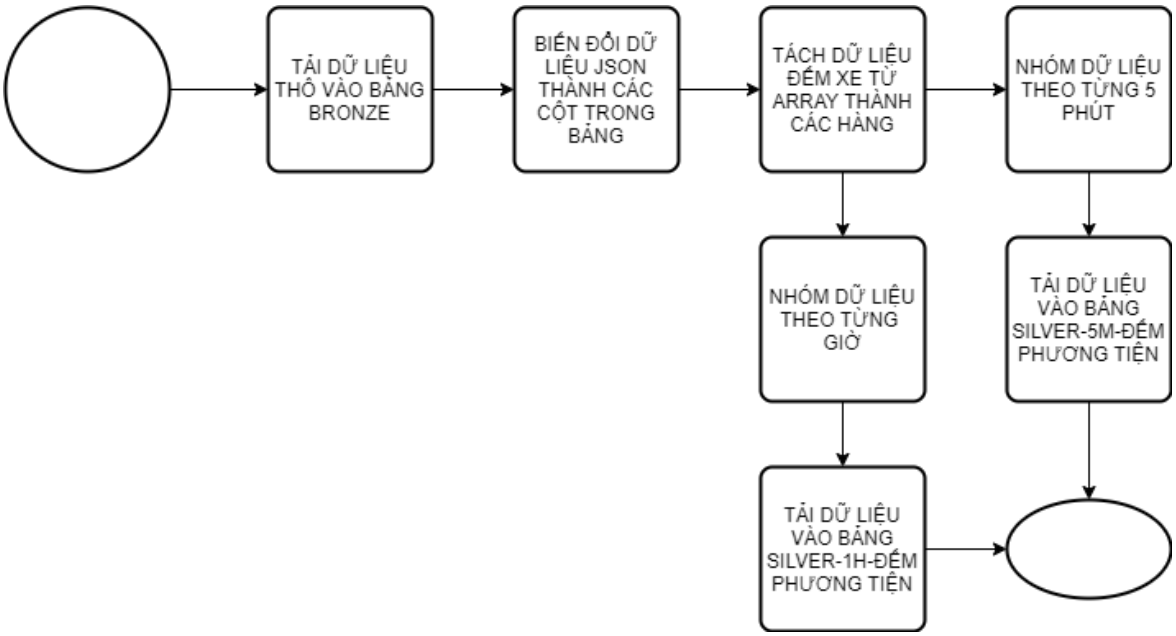
- Tầng dữ liệu thô hay còn gọi là Bronze là nơi lưu trữ những dữ liệu trích xuất nguyên bản từ nguồn thông tin mà không thông qua bất kỳ thay đổi nào. Mục đích của tầng dữ liệu này đó là lưu lại giá trị nguyên bản của dữ liệu, tạo điều kiện để giải quyết nhiều bài toán xung quanh chủ đề mà dữ liệu này đề cập đến;
- Tầng dữ liệu sạch hay còn gọi là Silver là nơi tập trung những dữ liệu đã được làm sạch. Mục đích của tầng dữ liệu này đó là lưu trữ và cung cấp một lượng dữ liệu đã được biến đổi xử lý từ những dữ liệu thô, từ những dữ liệu nhiễu, dư thừa, thiếu trật tự và thông tin thành những dữ liệu có cấu trúc, đầy đủ về mặt thông tin và sẵn sàng đưa vào phân tích;
- Tầng dữ liệu tổng hợp hay còn gọi là Gold là tầng lưu trữ những dữ liệu được thống kê biến đổi đặc trưng dành cho các bài toán cụ thể. Từ những dữ liệu ở tầng dữ liệu sạch, dữ liệu ở tầng này sẽ dùng cho mục đích trực quan và báo cáo, giúp cho người sử dụng có được lượng thông tin hữu ích để có thể ra các quyết định hoặc thiết lập các chiến lược. Nói cách khác, dữ liệu ở tầng này có giá trị về mặt thông tin cao nhất trong 3 tầng.



Hình 3.5: Dữ liệu tam cấp cho hệ thống lưu trữ

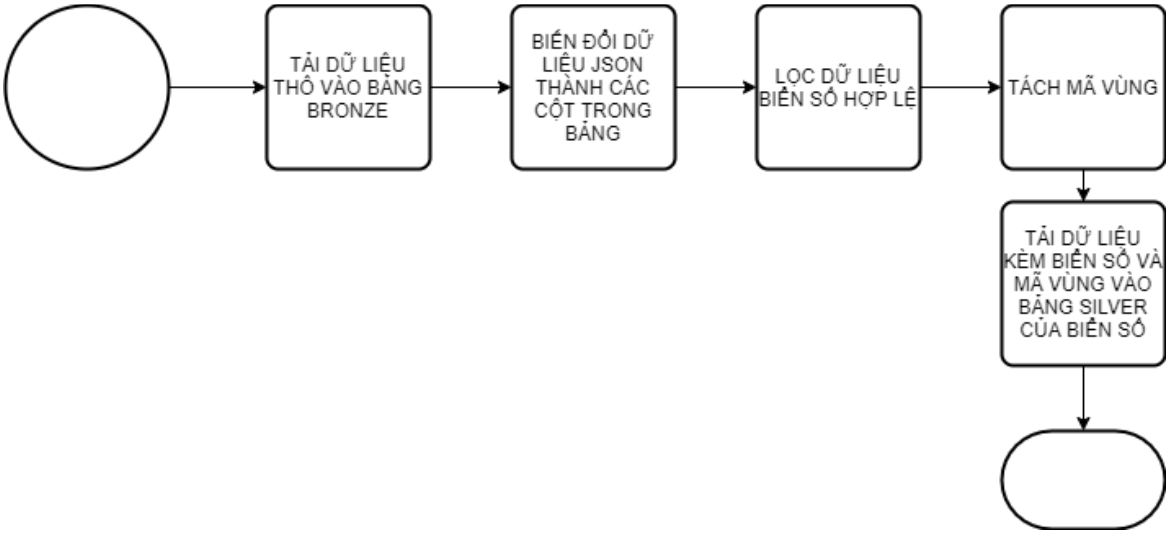
Thiết kế lưu trữ và ETL cho dữ liệu đếm xe và biển số

Đối với bài toán thống kê dữ liệu đếm xe ở phần 2, từ chuỗi JSON ban đầu biến đổi thành cột (Loại xe) sẽ là thông tin về loại phương tiện được ghi nhận và cột (Số lượng xe) là thông tin về số lượng phương tiện ứng với từng loại trong (Loại xe) (Hình 3.5)



Hình 3.6: Lưu đồ biến đổi dữ liệu đếm xe

Đối với bài toán thống kê biển số ở phần 2, từ chuỗi JSON ban đầu, biến đổi thành cột (Loại xe) có biển số được ghi nhận và cột (Biển số) là cột biển số phương tiện, tiếp theo là cột (Mã vùng) trích xuất từ (Biển số). Ngoài ra những biển số không hợp lệ cũng đã được sàng lọc (Hình 3.6)



Hình 3.7: Lưu đồ biến đổi dữ liệu đếm biển số

Kỹ thuật gom file và phân vùng dữ liệu

Cơ chế chung của các Table Format là khi phát sinh tác vụ CRUD đều sinh ra các transaction log và các snapshot của dữ liệu. Nghĩa là, khi ghi n record vào Table có m dòng, table format sẽ sinh ra một snapshot table hiện tại thành một phiên bản có m record và tạo ra một snapshot có m + n record hiện tại và liệt kê bản cũ thành phiên bản Time travel để dành cho các tác vụ truy xuất phiên bản quá khứ cũng như phục hồi dữ liệu nếu sinh ra lỗi.

Khi các dữ liệu dạng bó được tải vào bảng, số lượng snapshot được sinh ra là ít và rời rạc. Tuy nhiên, theo thời gian, khi liên tục ghi dữ liệu vào bảng, Delta sẽ sinh ra một lượng file rất lớn, đặc biệt với các tác vụ streaming data khi data được ghi vào thành từng bó nhỏ. Điều này sẽ ảnh hưởng đến hiệu suất đọc dữ liệu của bảng và hiệu năng hệ thống. Vì khi đọc một số lượng file lớn có dung lượng nhỏ cần được duyệt qua, điều này làm nghẽn IO của hệ thống qua quá trình đọc ghi.

- *Tối ưu gom file* là ghi lại các file nhỏ thành một file dữ liệu lớn hơn một cách tối ưu. Một số lượng file nhỏ dữ liệu của bảng có thể được gom thành file lớn hơn. Đặc biệt, Table Format Delta Lake có thể giúp thực hiện quá trình này mà không làm thay đổi dữ liệu của file, điều này giúp cho việc gộp file có thể thực hiện đồng thời với ghi dữ liệu vào bảng
- *Phân vùng dữ liệu* là chia nhỏ một cơ sở dữ liệu lớn thành các phân vùng. Các phân vùng sẽ được truy xuất trực tiếp bằng các câu truy vấn SQL. Khi dữ liệu đã được phân vùng, các công việc chỉ cần thao tác trên các vùng nhỏ của dữ liệu, thay vì phải scan toàn bộ cơ sở dữ liệu.

Ngoài ra, việc phân vùng dữ liệu còn hỗ trợ tốt cho tác vụ gộp file, khi mà các cơ chế gộp file sẽ phân biệt các phân vùng dữ liệu với nhau và tránh được việc xung đột khi cùng một lúc ghi vào một phân vùng. Các key để phân vùng là các dữ liệu trong một column. Để vừa phân biệt tốt các phân vùng dữ liệu, vừa không khiến cho dữ liệu được lưu trong bảng bị phân mảnh quá nhỏ cần chọn đúng các cột để phân vùng dữ liệu.

Hiệu năng truy vấn không chỉ phụ thuộc vào công cụ mà còn phụ thuộc việc tổ chức lưu trữ dữ liệu trên hệ thống lưu trữ. Các nội dung này sẽ được phân tích để tìm giải pháp tổ chức lưu trữ hợp lý cho hệ thống lưu trữ dữ liệu đo đếm phương tiện giao thông. Bài toán về kỹ thuật nâng cao hiệu năng truy vấn sẽ được xem xét và giải quyết trong phần này. Ngoài ra, sự tiện lợi và tính sẵn sàng của dữ liệu cho dữ liệu đo đếm phương tiện giao thông cũng được xét đến.

3.3.3. Giải thuật Support Vector Regression

Ý tưởng cơ bản của SVR là ánh xạ phi tuyến tập dữ liệu $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \subset R^n \times R$ sang một không gian đặc trưng nhiều chiều mà ở đó có thể sử dụng phương pháp hồi qui tuyến tính. Đặc điểm của SVR là khi xây dựng hàm hồi qui ta không cần sử dụng hết tất cả các điểm dữ liệu trong tập huấn luyện. Những điểm dữ liệu có đóng góp vào việc xây dựng hàm hồi qui được gọi là những véc tơ hỗ trợ.

Hàm hồi qui của SVR như sau:

$$f(x) = w^T \phi(x) + b \qquad (1)$$

Trong đó: $w \in R^m$ là véc tơ trọng số, $b \in R$ là hằng số, $x \in R^n$ là véc tơ đầu vào , $\phi(x) \in R^m$ là véc tơ đặc trưng.

Để tìm w và b, SVR giải quyết bài toán tối ưu hóa sau:

Cực tiểu hóa hàm:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \qquad (2)$$

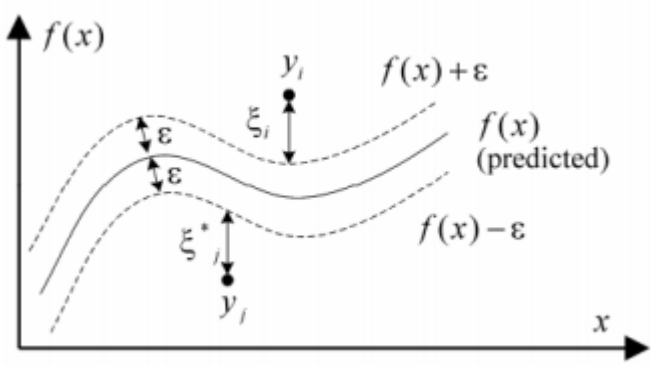
Với các ràng buộc:

$$\begin{cases} y_i - f(x_i) \le \varepsilon + \xi_i \\ f(x_i) - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases}$$

Với i = 1, 2, …, N

Trong đó, C là hằng số chuẩn hóa đóng vai trò cân bằng giữa độ lỗi huấn luyện và độ phức tạp mô hình.

Hình 3.7 minh họa SVR với hàm lỗi ε-insensitive . Đường nét liền ở giữa ứng với đường dự đoán. Giá trị Ɛ xác định độ rộng của ống bao quanh đường dự đoán. Nếu giá trị đích y_i nằm trong ống này thì coi như độ lỗi bằng 0. Nếu giá trị đích y_i nằm ngoài ống này thì độ lỗi bằng ξ_i (nếu y_i nằm ngoài phía trên ống) hoặc ξ_i^* (nếu y_i nằm ngoài phía dưới ống).



Hình 3.8: Minh họa hàm lỗi của thuật toán SVR

Từ (2) dùng hàm Lagrange và điều kiện Karush-Kuhn-Tucker, ta có bài toán tối ưu hóa tương đương:

Cực đại hóa:

$$-\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(x_i, x_j) + \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \varepsilon\sum_{i=1}^{N}(\alpha_i^* + \alpha_i) \qquad (3)$$

Với các ràng buộc:

$$\begin{cases} \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \\ 0 \le \alpha_i, \alpha_i^* \le C, i = 1, \dots, N \end{cases}$$

Trong đó, các nhân tử Lagrange α_i, α_i^* phải thỏa $\alpha_i\alpha_i^* = 0$. Véc tơ trọng tối ưu sẽ có dạng: $w = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\,\phi(x_i)$.

Từ đây, ta có hàm hồi qui của SVR:

$$f(x) = \sum_{i=1}^{N}(\alpha_i^* - \alpha_i)K(x, x_i) + b \qquad (4)$$

Trong đó, $K(x_i , x_j)$ được gọi là hàm nhân và có giá trị bằng tích vô hướng của hai véc tơ đặc trưng $\phi(x_i), \phi(x_j)$.

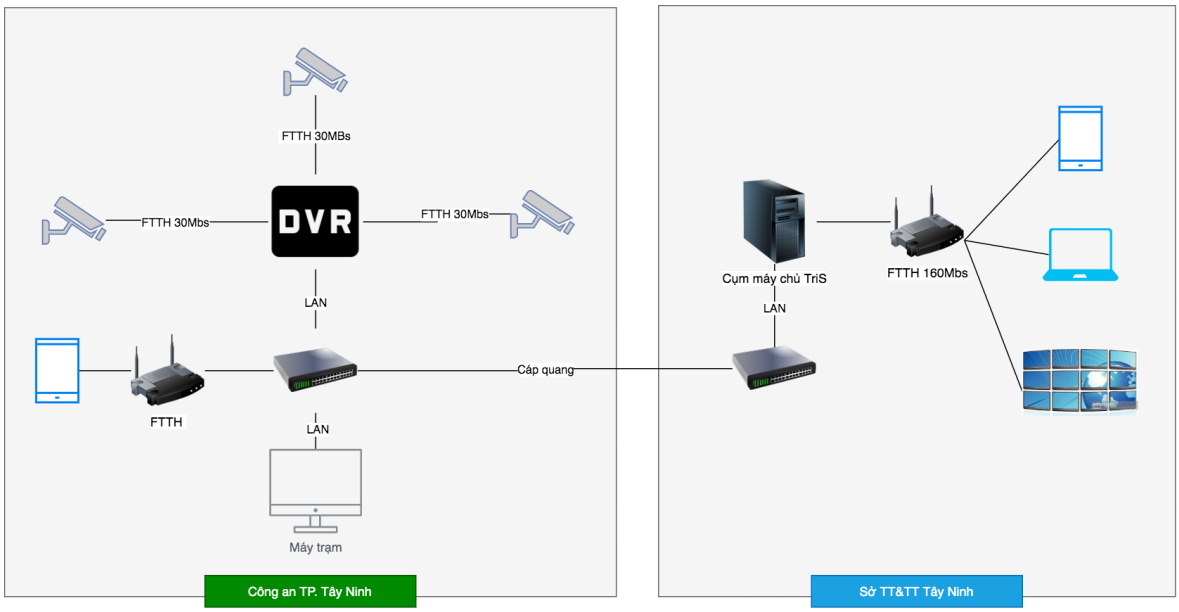
Bất kỳ một hàm nào thỏa điều kiện Mercer thì đều có thể được dùng làm hàm nhân. Hàm nhân được sử dụng phổ biến nhất là hàm Gaussian:

$$K(x_i, x_j) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$$

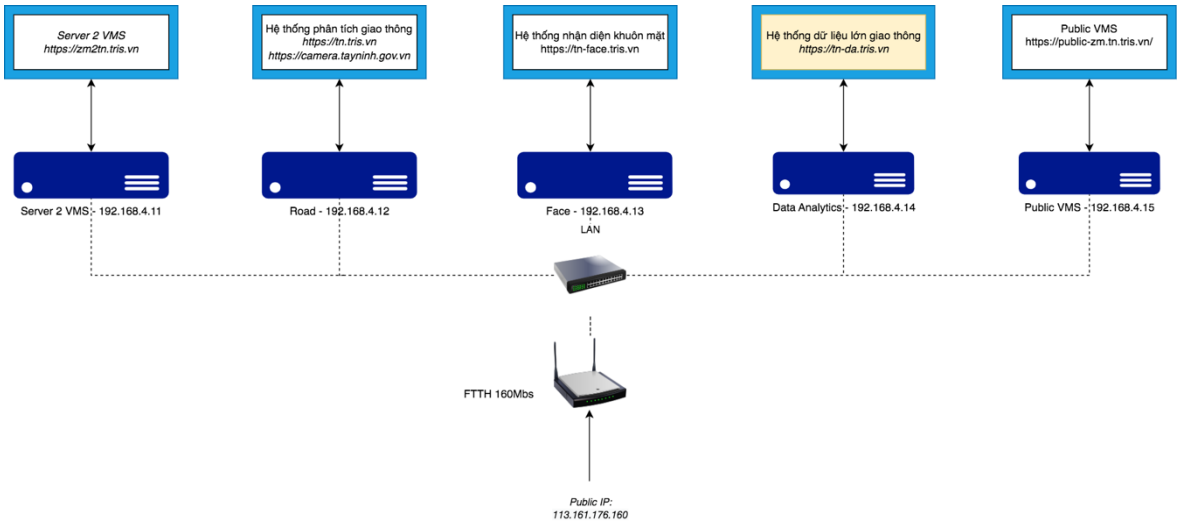
Như vậy, với SVR sử dụng hàm lỗi ε-insensitive . và hàm nhân Gaussian ta có ba tham số cần tìm: hệ số chuẩn hóa C, tham số γ của hàm nhân Gaussian và độ rộng của ống Ɛ . Cả bat ham số này đều ảnh hưởng đến độ chính xác dự đoán của mô hình và cần phải chọn lựa kỹ càng. Nếu C quá lớn thì sẽ ưu tiên vào phần độ lỗi huấn luyện, dẫn đến mô hình phức tạp, dễ bị quá khớp. Còn nếu C quá nhỏ thì lại ưu tiên vào phần độ phức tạp mô hình, dẫn đến mô hình quá đơn giản, giảm độ chính xác dự đoán. Ý nghĩa của Ɛ cũng tương tự C. Nếu Ɛ quá lớn thì ít có véc tơ hỗ trợ, làm cho mô hình quá đơn giản. Ngược lại, nếu Ɛ quá nhỏ thì có nhiều véc tơ hỗ trợ, dẫn đến mô hình phức tạp, dễ bị quá khớp. Tham số γ phản ánh mối tương quan giữa các véc tơ hỗ trợ nên cũng ảnh hưởng đến độ chính xác dự đoán của mô hình.

CHƯƠNG 4: THỰC NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

4.1. Mô hình triển khai



Hình 4.1: Mô hình kết nối camera



Hình 4.2: Sơ đồ kết nối máy chủ

Hình 4.3: Hình ảnh một số camera nhận diện bảng số xe

4.2. Kết quả thực nghiệm và đánh giá

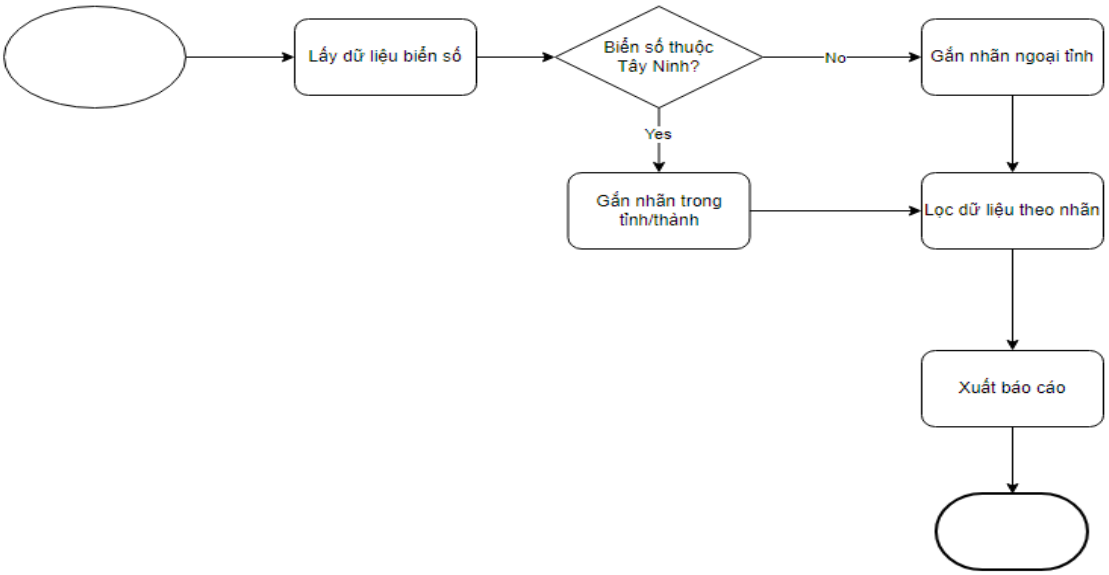
4.2.1. Tóm tắt dữ liệu

- Số lượng camera nhận diện bảng số: 24
- Số lượng camera đếm lưu lượng xe: 21
- Số lượng bảng số xe trung bình theo ngày: 47.828
- Số lượng xe trung bình theo ngày:
    - Xe máy: 25.000
    - Xe hơi: 6.233
    - Xe ô tô: 368
    - Xe tải: 4.119
    - Xe bus: 366
    - Xe container: 559

4.2.2. Một số tính năng phân tích dữ liệu dòng giao thông

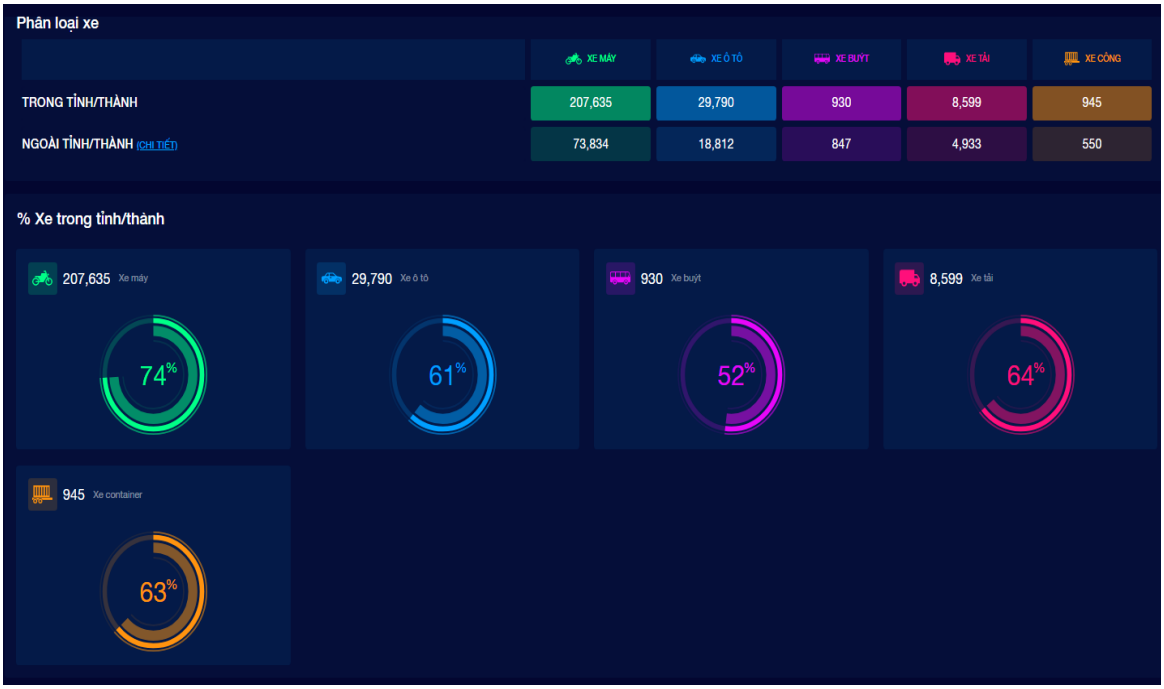
- Phân tích xe trong và ngoài tỉnh:

- Lưu đồ:



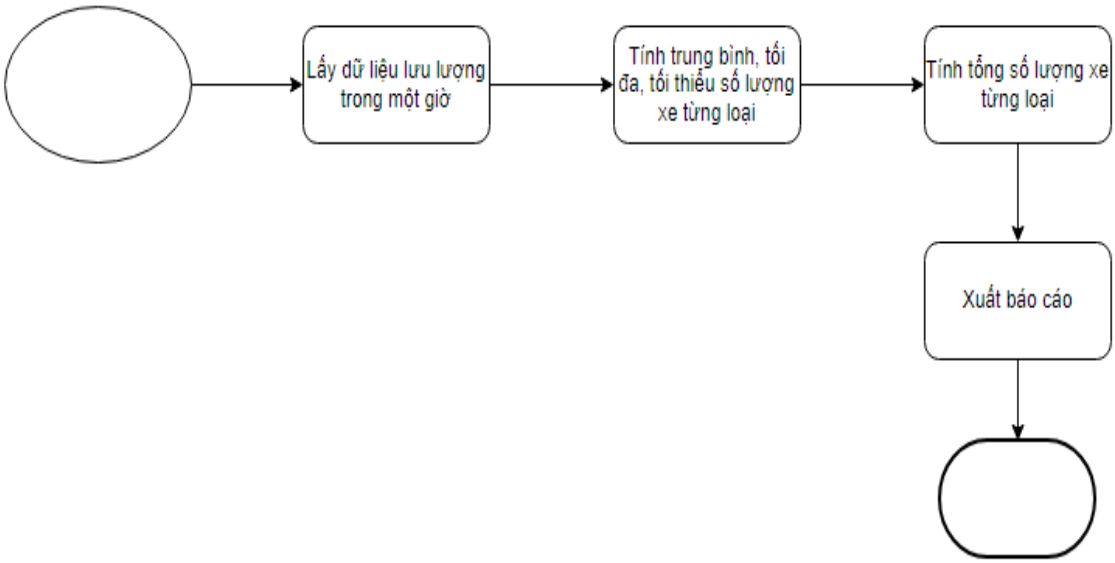
Hình 4.4: Lưu đồ phân tích xe trong và ngoài tỉnh

- Hình ảnh phân tích:



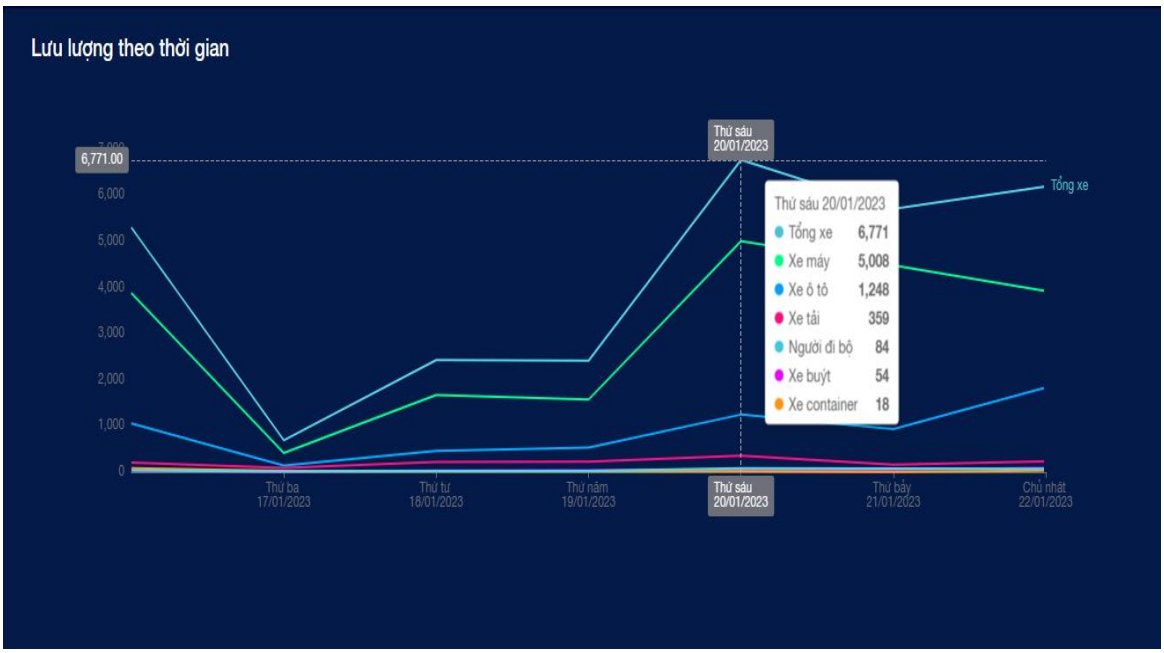
Hình 4.5: Hình ảnh phân tích xe trong và ngoài tỉnh

- Phân tích lưu lượng xe:
  - Lưu đồ:

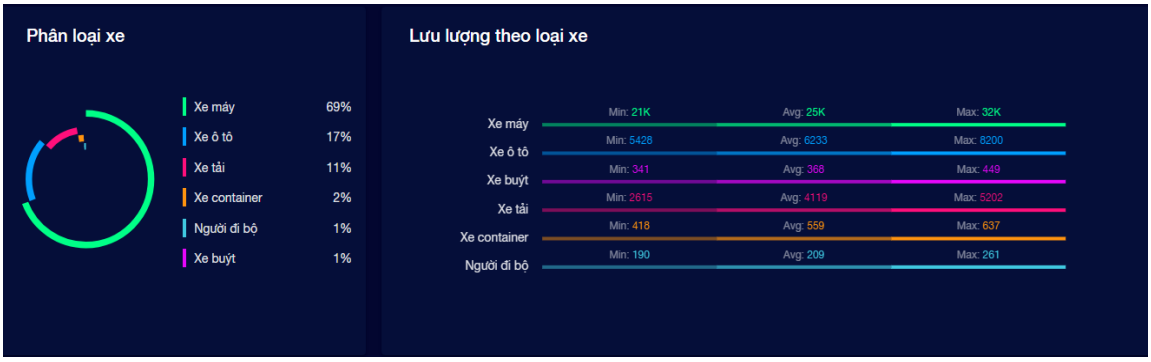


Hình 4.6: Lưu đồ phân tích lưu lượng xe

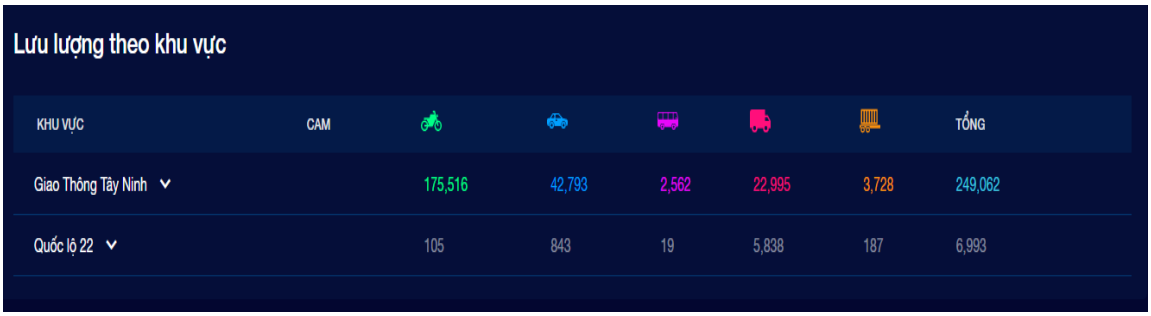
  - Hình ảnh phân tích lưu lượng xe:



Hình 4.7: Hình ảnh phân tích lưu lượng xe theo thời gian

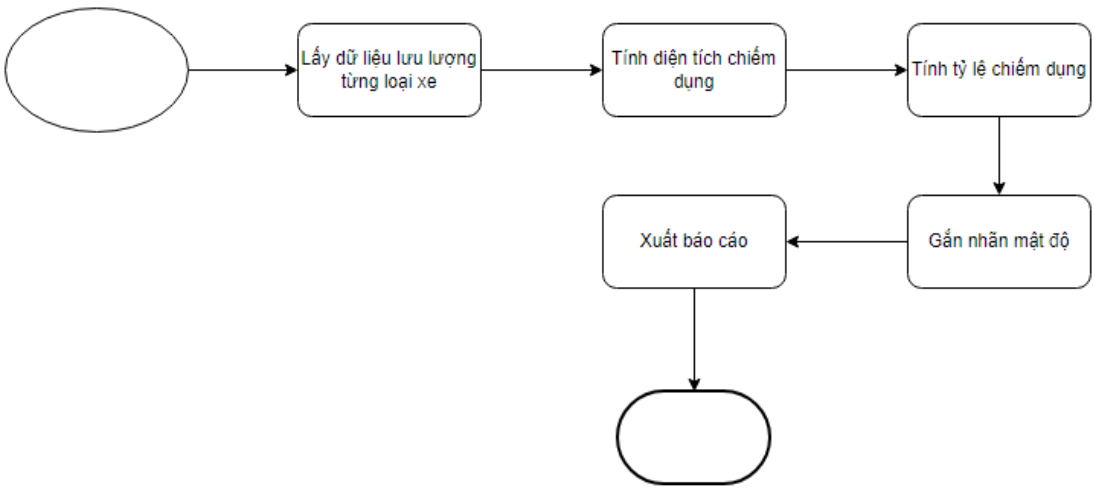


Hình 4.8: Hình ảnh phân tích lưu lượng xe theo loại xe



Hình 4.9: Hình ảnh phân tích lưu lượng xe theo khu vực

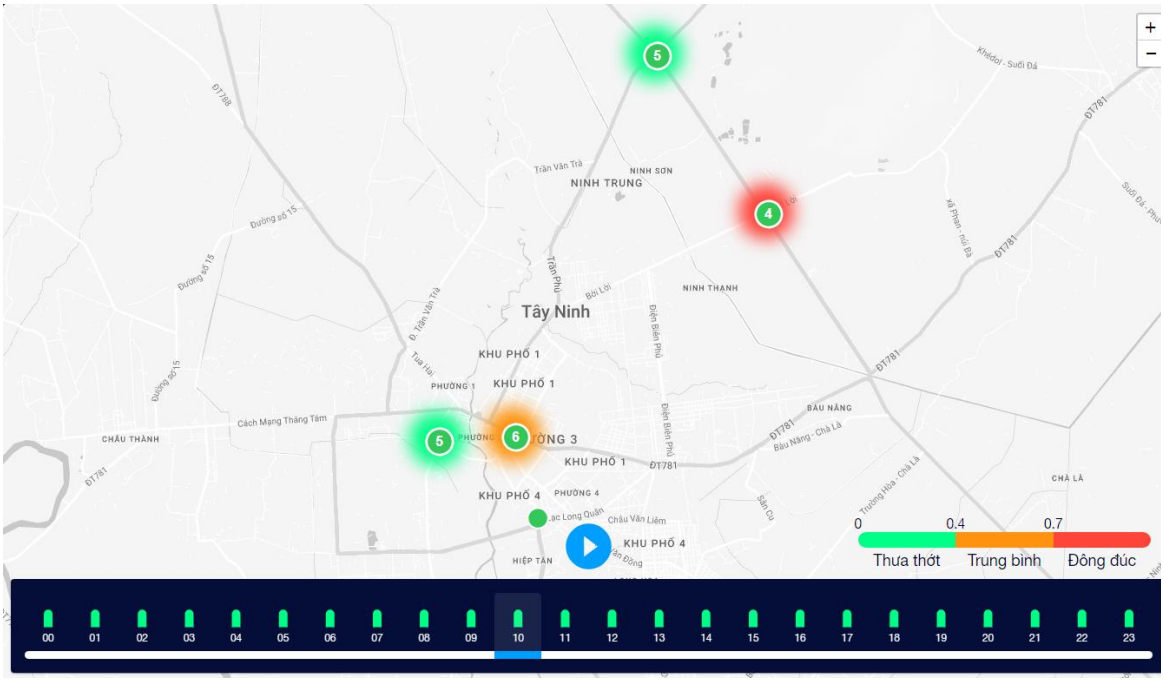
- Phân tích mật độ xe:
  - Lưu đồ:



Hình 4.10: Lưu đồ phân tích mật độ xe

Tỷ lệ chiếm dụng được tính theo công thức: $R_{\text{chiếm dụng}} = S_{\text{chiếm dụng}} / S_{\text{mặt đường}}$

- Hình ảnh phân tích mật độ xe:



Hình 4.11: Hình ảnh phân tích mật độ xe

4.2.3. Mô hình dự báo lưu lượng giao thông

Thuật giải

Support-Vector Networks, Support Vector Machines (SVM) là một giải thuật học máy được dùng để giải bài toán phân loại hai nhóm dữ liệu khác nhau. Ý tưởng của cơ bản thuật toán như sau: các giá trị vector đặc trưng đầu vào là phi tuyến tính, chúng sẽ được ánh xạ lên một không gian đặc trưng có số chiều cao hơn và từ đó các quyết định tuyến tính sẽ được đưa ra dựa trên không gian đặc trưng mới này.

Giải thuật Support Vector Regression Machines (SVR) là một phần mở rộng của Support-Vector Networks khi thay vì đưa ra các quyết định cho bài toán phân loại thì SVR được sử dụng để học và dự báo dữ liệu trong các bài toán hồi quy. SVR tận dụng được ưu điểm của SVM khi không phụ thuộc vào số chiều không gian của vector dữ liệu đầu vào và xử lý ở một chiều không gian đặc trưng cao hơn của riêng mình.

Do đó, giải thuật SVR được chọn để tiến hành thực nghiệm trên bộ dữ liệu thu thập được từ mô hình đề xuất trên để dự báo dữ liệu lưu lượng trong tương lai gần. Cụ thể hơn là lưu lượng xe máy tại một thiết bị camera quan sát trong tương lai gần.

Training

Dữ liệu thu thập được là lưu lượng xe máy tại một camera trong khoảng thời gian từ 05/05/2022 đến 23/05/2022. Kích thước dữ liệu gồm 26.199 mẫu dữ liệu, mỗi mẫu dữ liệu cách nhau một phút. Các dữ liệu ở các thời điểm bị thiếu sẽ được nội suy bằng cách gán bằng với giá trị dữ liệu gần kề trong quá khứ. Việc chuẩn hóa dữ liệu sẽ được chuẩn hóa bằng chuẩn hóa min-max. Bộ dữ liệu sẽ được phân chia làm 3 phần: 60% dùng để training, 20% dùng để validating và 20% dùng để testing.

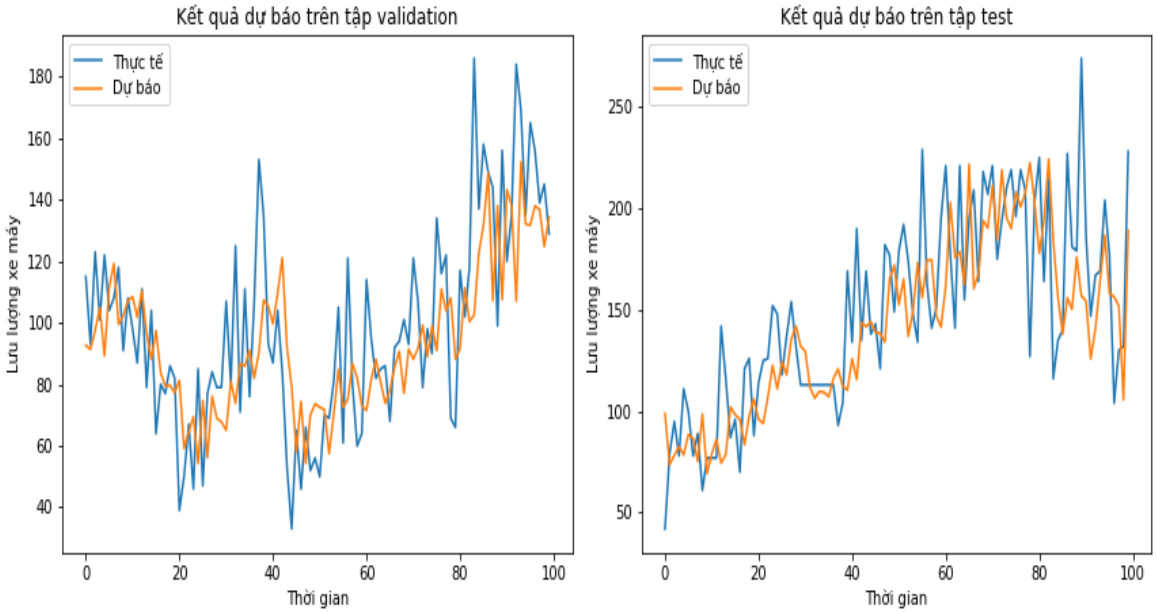
Testing

Quá trình kiểm tra chất lượng dự báo được dựa trên chỉ số R2 (R-squared) để đánh giá kết quả. Chỉ số càng gần 1 thì kết quả dự báo càng tốt. Nếu chỉ số càng gần 0 thì kết quả dự báo không tốt.

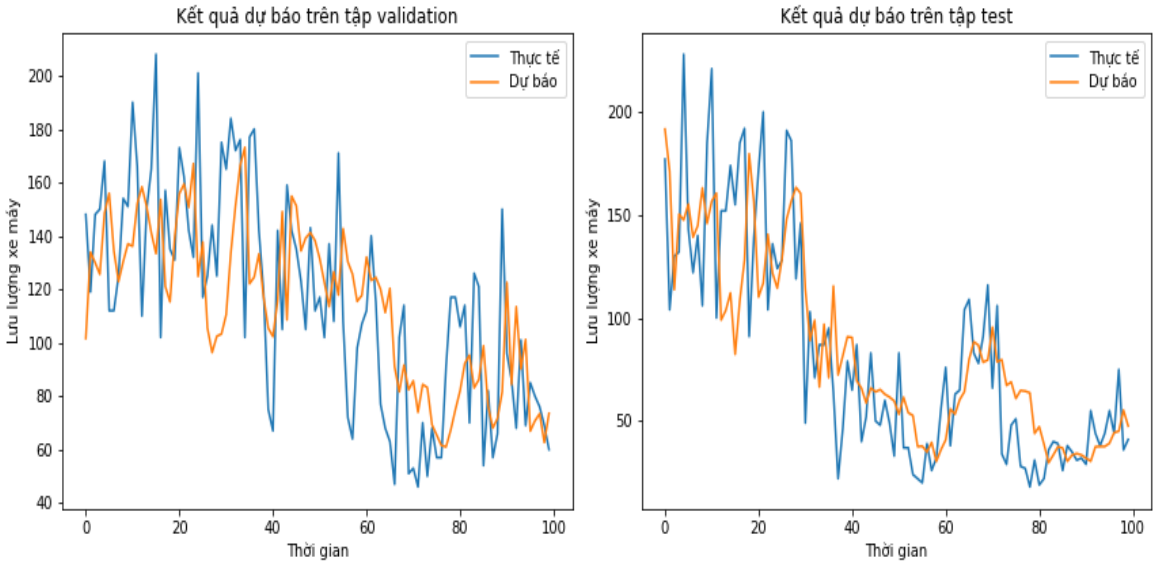
Bảng 4.1: Kiểm tra chất lượng dự báo

Tập dữ liệu	**Số bước vào tương lai**	**R2**
Validation	1	0.859
Test	1	0.851
Validation	5	0.822
Test	5	0.807

Hình biểu đồ dự báo



Hình 4.12: Dự báo lưu lượng xe máy ở 1 bước vào tương lai



Hình 4.13: Dự báo lưu lượng xe máy ở 5 bước vào tương lai

Kết quả dự báo bước đầu tương đối hiệu quả khi hệ số tương quan Pearson R2 có giá trị lớn hơn 0.8. Tuy nhiên nhìn biểu đồ dự báo (Hình 4.13) cho thấy sai lệch

giữa thực tế và kết quả dự báo còn cao, đặc biệt là những đỉnh (peak) của biểu đồ. Các cải tiến sẽ tiếp tục được nghiên cứu trong đó học viên sẽ tập trung vào thu thập thêm dữ liệu mẫu và thử nghiệm dữ liệu mẫu này với các thuật giải học sâu như Deep Belief Network, Long-S (DBN), Long-Short Term Memory (LSTM), …

CHƯƠNG 5: KẾT LUẬN

5.1. Kết quả nghiên cứu của đề tài

Trong giải pháp giám sát giao thông đô thị, bài toán về đo đếm phương tiện giao thông là một bài toán cần thiết cho việc phân tích đánh giá mật độ giao thông. Lưu trữ được dữ liệu đo đếm phương tiện sẽ hỗ trợ nhiều báo cáo, phân tích, từ đó các nhà lãnh đạo có thể đưa ra những chính sách hợp lý để điều tiết phân luồng giao thông cũng như huy động lực lượng điều tiết tại các điểm nóng ùn tắc giao thông. Bài toán liên quan đến dữ liệu phương tiện giao thông đã được nghiên cứu và giải quyết trong luận văn này:

- Giải pháp lưu trữ và truy vấn dữ liệu đo đếm phương tiện giao thông với các kiến trúc đề xuất bao gồm: Delta + HDFS, Delta + MinIO, Iceberg + MinIO + Trino;
- Kỹ thuật nâng cao hiệu năng truy vấn: Gom file và Phân vùng dữ liệu.
- Ứng dụng học máy vào trong công tác dự báo dữ liệu lưu lượng giao thông.

Các giải pháp này cũng đã được triển khai đánh giá trên tập dữ liệu đo đếm và nhận diện biển số phương tiện giao thông thực với: 717.420 record dữ liệu biển số phương tiện và 549.675 record dữ liệu đo đếm số lượng phương tiện.

Kết quả nghiên cứu trong luận văn là cơ sở để triển khai hệ lưu trữ, truy vấn dữ liệu giao thông nhằm phục vụ cho mục đích dự báo lưu lượng phương tiện giao thông, quy hoạch giao thông cho các đô thị.

5.2. Hạn chế luận văn

Số lượng mẫu dữ liệu còn hạn chế và thời gian nghiên cứu luận văn chưa đủ dài để đánh giá tính tối ưu nhất của giải pháp.

5.3. Hướng phát triển tiếp theo của đề tài nghiên cứu

- Tiếp tục nghiên cứu triển khai đánh giá hiệu năng trên hệ thống phân tán nhiều máy chủ.
- Cải tiến hiệu năng mô hình dự báo trong đó sẽ tiếp tục thu thập thêm dữ liệu mẫu và thử nghiệm dữ liệu mẫu này với các thuật giải học sâu như Deep Belief Network, Long-S (DBN), Long-Short Term Memory (LSTM), …

DANH MỤC TÀI LIỆU THAM KHẢO

[1] M. Chowdhury, A. Apon, and K. Dey, *Data analytics for intelligent transportation systems*. Elsevier, 2017.

[2] J. Dorsey, "Big data in the driver's seat of connected car technological advances," 2013.

[3] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: challenges and opportunities," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 1986–1989, 2019.

[4] C. Paschalidi, "Data governance: A conceptual framework in order to prevent your data lake from becoming a data swamp," 2015.

[5] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics," in *Proceedings of CIDR*, 2021.

[6] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. van Hovell, A. Ionescu, A. Łuszczak, *et al.*, "Delta lake: high-performance acid table storage over cloud object stores," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020.

[7] D. Tovarňák, M. Raček, and P. Velan, "Cloud native data platform for network telemetry and analytics," in *2021 17th International Conference on Network and Service Management (CNSM)*, pp. 394–396, IEEE, 2021.