TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
VIỆN TRÍ TUỆ NHÂN TẠO
-----*-----**

BÁO CÁO MÔN HỌC KỸ THUẬT VÀ CÔNG NGHỆ DỮ LIỆU LỚN

ĐỀ TÀI

BITCOIN PRICE STREAMING PROCESSING

Nhóm sinh viên thực hiện:

1. Phạm Thành Long - 22022604
2. Nguyễn Đức Minh - 22022533
3. Phan Văn Hiếu - 22022527
4. Trần Tiến Nam - 22022594

Giảng viên hướng dẫn: TS. Trần Hồng Việt
ThS. Ngô Minh Hương
CN. Lương Sơn Bá

HÀ NỘI - 2024

MỞ ĐẦU

Sự tăng trưởng nhanh chóng và biến động của Bitcoin đặt ra nhiều thách thức và cơ hội đáng kể cho việc dự đoán giá, đặc biệt trong kỷ nguyên dữ liệu lớn. Dự án này xây dựng một pipeline có khả năng mở rộng và hoạt động theo thời gian thực để dự đoán giá Bitcoin, đồng thời lưu trữ và xử lý dữ liệu giá trong lịch sử, sử dụng môi trường dữ liệu lớn bao gồm Apache Kafka, Hadoop File System (HDFS), Apache Spark, HBase và MySQL database. Bằng cách tận dụng dữ liệu luồng từ sàn giao dịch tiền mã hóa Binance, chúng tôi thực hiện tiền xử lý và phân tích dữ liệu để huấn luyện một mô hình học máy có khả năng cung cấp dự đoán giá ngắn hạn chính xác.

Các kỹ thuật chính bao gồm xây dựng pipeline để xử lý dữ liệu có khối lượng lớn và tốc độ cao, kỹ thuật trích xuất đặc trưng để nhận diện xu hướng giá, và mô hình dự đoán tiên tiến sử dụng các mô hình Machine Learning hoặc Deep Learning hiện đại. Việc triển khai sử dụng Spark MLlib để huấn luyện mô hình phân tán và HBase để lưu trữ dữ liệu lịch sử, đảm bảo tính khả dụng cao và chịu lỗi tốt.

Kết quả của chúng tôi chứng minh tính hiệu quả của framework được đề xuất, cung cấp dự đoán thời gian thực với độ trễ tối thiểu. Dự án này mang lại những hiểu biết có giá trị về việc xây dựng các pipeline dữ liệu lớn mạnh mẽ cho các ứng dụng tài chính và gợi mở các hướng cải tiến tiềm năng trong dự đoán giá tiền mã hóa.

MỤC LỤC

CHƯƠNG 1: TỔNG QUAN

1.1. Giới thiệu

1.1.1. Tại sao việc dự đoán giá Bitcoin lại quan trọng?

Dự đoán giá Bitcoin và thị trường tài chính nói chung rất quan trọng vì nó giúp nhà đầu tư đưa ra quyết định thông minh, giảm thiểu rủi ro và tối ưu hóa lợi nhuận. Trong một thị trường đầy biến động như Bitcoin, dự đoán chính xác có thể giúp các tổ chức và cá nhân xác định thời điểm mua vào hoặc bán ra, từ đó tăng cường khả năng sinh lời. Ngoài ra, việc dự đoán giá cũng hỗ trợ trong việc quản lý các chiến lược đầu tư dài hạn, ổn định tài chính và đảm bảo an toàn cho các hệ thống giao dịch tự động.

Những thách thức trong môi trường dữ liệu lớn:

- Khối lượng dữ liệu lớn: Dữ liệu khổng lồ và liên tục tăng yêu cầu khả năng mở rộng của các hệ thống xử lý.
- Độ phức tạp của dữ liệu: Dữ liệu không có cấu trúc hoặc bán cấu trúc khó xử lý và phân tích.
- Tốc độ dữ liệu: Dữ liệu đến nhanh chóng và liên tục (real-time), đòi hỏi các hệ thống xử lý dữ liệu phải nhanh chóng và hiệu quả.
- Đảm bảo tính chính xác: Dữ liệu từ nhiều nguồn khác nhau có thể không đồng nhất, ảnh hưởng đến độ chính xác của các phân tích.
- Quản lý dữ liệu: Việc lưu trữ, bảo mật và truy xuất dữ liệu trong môi trường phân tán gặp nhiều khó khăn.
- Chi phí tính toán: Xử lý và lưu trữ dữ liệu lớn tốn kém về chi phí hạ tầng và tài nguyên.
- Tích hợp hệ thống: Kết nối và đồng bộ các công cụ và nền tảng khác nhau trong hệ sinh thái dữ liệu lớn đôi khi gặp phải vấn đề về tương thích.

1.1.2. Mục tiêu cụ thể của dự án

Mục tiêu cụ thể của dự án *Dự đoán giá Bitcoin trong môi trường dữ liệu lớn* bao gồm:

- Phát triển hệ thống dự đoán giá Bitcoin theo thời gian thực: Sử dụng môi trường dữ liệu lớn (Kafka, Spark, HBase) để thu thập, xử lý và phân tích dữ liệu Bitcoin từ các sàn giao dịch.
- Xây dựng pipeline dữ liệu hiệu quả: Tạo ra một hệ thống xử lý dữ liệu mạnh mẽ và có khả năng mở rộng, cho phép thu thập và xử lý dữ liệu trong thời gian thực.

- Ứng dụng các mô hình học máy tiên tiến: Áp dụng thuật toán dự đoán giá XGBoost để dự đoán xu hướng giá Bitcoin trong ngắn hạn, từ đó tối ưu hóa chiến lược đầu tư.
- Đảm bảo khả năng mở rộng và tính khả dụng: Triển khai hệ thống trên môi trường thử nghiệm, đảm bảo có thể xử lý khối lượng dữ liệu lớn và hoạt động ổn định, khả năng mở rộng trong tương lai.

1.1.3. Phạm vi dự án

Dự án này bao gồm:

- Thu thập và xử lý dữ liệu: Dự án sẽ thu thập dữ liệu Bitcoin từ các sàn giao dịch, xử lý dữ liệu thô để chuẩn bị cho việc phân tích, bao gồm làm sạch, chuẩn hóa và trích xuất đặc trưng.
- Xây dựng pipeline dữ liệu thời gian thực: Sử dụng Apache Kafka để thu thập dữ liệu trực tiếp, Apache Spark để xử lý dữ liệu và HBase để lưu trữ dữ liệu lớn, đảm bảo hệ thống có thể mở rộng và chịu lỗi.
- Ứng dụng mô hình học máy: Triển khai các mô hình học máy (XGBoost) để dự đoán giá Bitcoin dựa trên dữ liệu lịch sử và các đặc trưng thị trường.
- Tích hợp và triển khai hệ thống: Triển khai hệ thống dự đoán trong môi trường dữ liệu lớn, có khả năng xử lý và dự đoán giá Bitcoin theo thời gian thực.

Dự án sẽ không đi sâu vào các chủ đề:

- Dự đoán giá dài hạn: Dự án sẽ tập trung vào dự đoán ngắn hạn (từng phút hoặc giờ), không bao gồm các dự đoán dài hạn (ngày, tuần, tháng).
- Phân tích các yếu tố tác động ngoài dữ liệu thị trường: Dự án không xem xét các yếu tố như chính trị, kinh tế vĩ mô, hoặc các yếu tố xã hội tác động đến giá Bitcoin.
- Xử lý các loại tiền mã hóa khác: Dự án chỉ tập trung vào dự đoán giá Bitcoin, không mở rộng sang các loại tiền mã hóa khác như Ethereum hoặc Ripple.
- Đánh giá hiệu suất mô hình: Đánh giá độ chính xác của các mô hình thông qua các chỉ số như RMSE, MAE, và so sánh kết quả dự đoán với các mô hình khác.
- Tối ưu hóa mô hình cho các thị trường khác: Mô hình được xây dựng cho thị trường Bitcoin và không tối ưu cho các thị trường tài chính khác.
- Phân tích chi tiết các chiến lược đầu tư: Dự án không sẽ không đi sâu vào việc phát triển các chiến lược đầu tư hoặc quản lý danh mục đầu tư.

1.2. Một số nghiên cứu liên quan

1.2.1. Tổng quan về các nghiên cứu

Nghiên cứu về dự đoán giá Bitcoin đã thu hút sự chú ý lớn nhờ vào tính biến động của tiền điện tử này. Nhiều nghiên cứu đã khám phá các mô hình học máy (ML) và học sâu (DL) khác nhau để dự đoán giá Bitcoin, tận dụng nhiều nguồn dữ liệu khác nhau. Các phương pháp truyền thống như hồi quy tuyến tính và ARIMA đã được áp dụng rộng rãi, nhưng các kỹ thuật tiên tiến hơn như mạng nơ-ron (neural networks), mạng nơ-ron hồi quy (RNNs), và mạng LSTM (Long Short-Term Memory) đã cho thấy hiệu quả vượt trội trong việc nhận diện các mô hình phi tuyến tính phức tạp của biến động giá Bitcoin.

1.2.2. Các kỹ thuật phổ biến

Một số kỹ thuật học máy và học sâu được sử dụng phổ biến trong dự đoán giá Bitcoin bao gồm:

- Mạng nơ-ron sâu (DNNs): Các mô hình này hiệu quả trong việc dự đoán cả mức độ thay đổi giá và xu hướng giá (Omole Oluwadamilare and Enke David 2024)[1]
- Mạng nơ-ron hồi quy (RNNs) và LSTM: LSTM, một loại mạng nơ-ron hồi quy, đặc biệt phù hợp với dữ liệu chuỗi thời gian như giá Bitcoin, vì chúng có thể nắm bắt được các phụ thuộc dài hạn trong dữ liệu (Ji Suhwan, Kim Jongmin, and Im Hyeonseung 2019)[2]
- Máy vectơ hỗ trợ (SVM) và Rừng ngẫu nhiên (Random Forests): Các mô hình này cũng được sử dụng trong các tác vụ phân loại và hồi quy, với máy vectơ hỗ trợ có hiệu suất tốt trong dự đoán biến động giá (Jaquart Patrick, Dann David, and Martin Carl 2020)[3]

1.2.3. Các đặc trưng dùng cho dự đoán

Dự đoán giá Bitcoin sử dụng nhiều loại đặc trưng khác nhau, bao gồm:

- Chỉ báo kỹ thuật: Bao gồm dữ liệu giá lịch sử, khối lượng giao dịch và các chỉ số thị trường khác.
- Đặc trưng từ blockchain: Dữ liệu từ blockchain Bitcoin, như độ khó khai thác hoặc số giao dịch mỗi khối, cũng được sử dụng để dự đoán biến động giá.
- Phân tích cảm xúc: Thông tin từ mạng xã hội và các nguồn tin tức, bao gồm cảm xúc trên Twitter và dữ liệu từ Google Trends, đã được tích hợp vào các mô hình dự đoán.
- Các yếu tố kinh tế vĩ mô: Bao gồm các yếu tố như lãi suất, chỉ số thị trường chứng khoán và giá hàng hóa có thể ảnh hưởng đến giá trị của Bitcoin.

1.2.4. Vai trò của BigData trong dự đoán giá Bitcoin

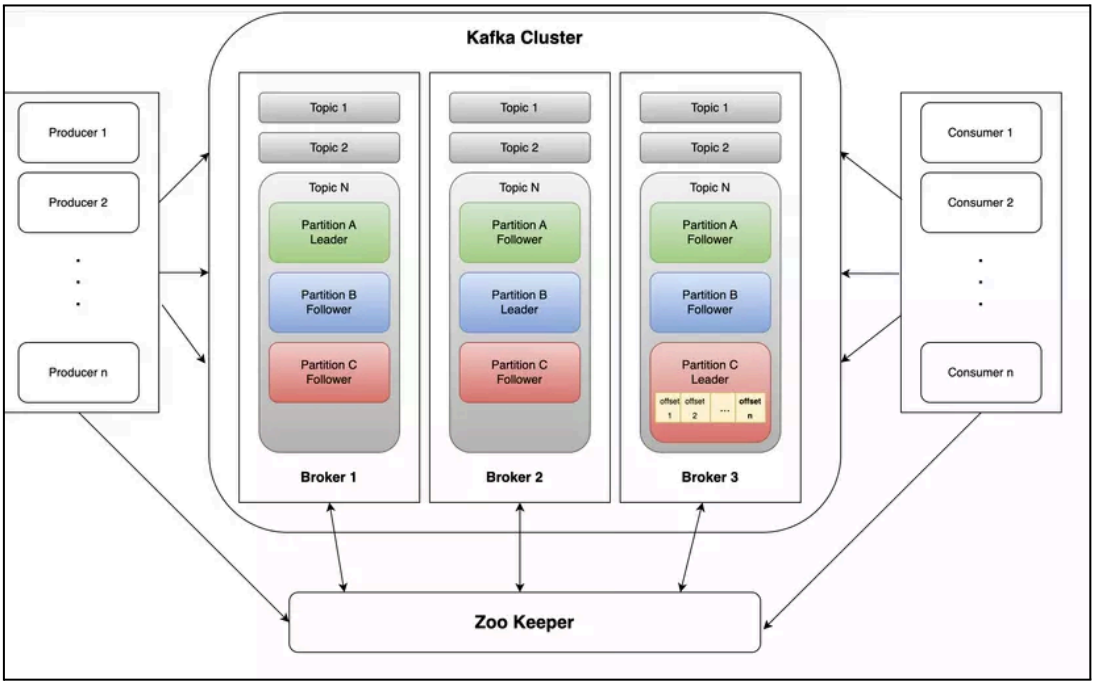
Việc sử dụng Big Data đã trở thành một lợi thế quan trọng trong việc cải thiện độ chính xác và độ tin cậy của dự đoán giá Bitcoin. Big Data cho phép kết hợp khối lượng lớn thông tin thời gian thực từ nhiều nguồn khác nhau, bao gồm dữ liệu blockchain, tín hiệu thị trường và xu hướng trên mạng xã hội. Bằng cách tận dụng các công cụ Big Data như Apache Spark và Kafka, các nhà nghiên cứu có thể xử lý và phân tích dữ liệu lớn một cách hiệu quả, giúp dự đoán chính xác hơn và tối ưu hóa các chiến lược giao dịch kịp thời.

Tóm lại, mặc dù các mô hình truyền thống vẫn còn giá trị, nhưng việc sử dụng Big Data và các kỹ thuật học sâu tiên tiến đang nâng cao khả năng dự đoán giá Bitcoin chính xác hơn. Tuy nhiên, lĩnh vực này vẫn đang phát triển, với nhu cầu tiếp tục cải tiến việc lựa chọn đặc trưng và tối ưu hóa mô hình để vượt qua các thách thức như hiện tượng overfitting và sự biến động cao.

CHƯƠNG 2: PHƯƠNG PHÁP

2.1. Các công nghệ sử dụng

2.1.1. Kafka



Môi trường triển khai Apache Kafka thường bao gồm các thành phần sau:

- Zookeeper: Trước phiên bản Kafka 2.8, Zookeeper là thành phần quan trọng để quản lý thông tin về cluster, phân phối và đồng bộ hóa các thông tin giữa các node Kafka. Zookeeper giúp quản lý các broker, lưu trữ thông tin về topic và các phân vùng (partitions), và xử lý các thay đổi trong cluster.

- Kafka Brokers: Các broker là các server chịu trách nhiệm lưu trữ và phân phối dữ liệu. Mỗi broker trong một Kafka cluster có thể xử lý một phần dữ liệu (partition), và dữ liệu trong các partition này có thể được sao chép giữa các broker để tăng độ sẵn sàng và khôi phục dữ liệu khi gặp sự cố.
- Producers: Là các ứng dụng hoặc thành phần gửi dữ liệu vào Kafka. Dữ liệu này được gửi dưới dạng các message, thường là các sự kiện, vào một topic cụ thể trong Kafka.
- Consumers: Các ứng dụng hoặc thành phần đọc dữ liệu từ Kafka. Các consumer sẽ đăng ký để nhận các message từ một hoặc nhiều topic. Mỗi consumer có thể làm việc độc lập hoặc nằm trong một consumer group để chia sẻ tải công việc.
- Kafka Topics: Một topic trong Kafka là một kênh mà qua đó dữ liệu được gửi và nhận. Producers ghi vào topic và Consumers đọc từ topic. Kafka hỗ trợ chia nhỏ (partition) topic thành nhiều phần để xử lý song song và nâng cao hiệu suất.
- Kafka Connect: Đây là một công cụ để kết nối Kafka với các hệ thống bên ngoài (như cơ sở dữ liệu, HDFS, hoặc các hệ thống messaging khác). Nó giúp việc tích hợp dữ liệu trở nên đơn giản hơn.
- Kafka Streams: Là một thư viện Java giúp xử lý dữ liệu trong Kafka. Kafka Streams cho phép xử lý dữ liệu thời gian thực trực tiếp trong Kafka mà không cần phải sử dụng các công cụ xử lý như Spark hoặc Flink.
- Kafka Schema Registry: Được sử dụng để quản lý và bảo vệ cấu trúc dữ liệu (schema) cho các message trong Kafka, giúp dễ dàng xử lý và kiểm tra tính hợp lệ của dữ liệu.

Môi trường triển khai Kafka có thể hoạt động trên một hệ thống phân tán, cho phép mở rộng quy mô hệ thống với các broker và Zookeeper được triển khai trên nhiều máy chủ khác nhau. Các thành phần này hoạt động cùng nhau để đảm bảo tính sẵn sàng cao, chịu lỗi tốt và khả năng xử lý dữ liệu lớn.

Thông qua việc triển khai Kafka trong môi trường phân tán, hệ thống có thể xử lý hàng triệu message mỗi giây với độ trễ thấp, là lựa chọn phổ biến cho các ứng dụng cần xử lý và phân phối dữ liệu thời gian thực.

2.1.2. Hadoop File System

HDFS (Hadoop Distributed File System) là một hệ thống lưu trữ phân tán được thiết kế để chạy trên các phần cứng thương mại thông thường, thường được sử dụng trong môi trường Big Data. HDFS là một thành phần cốt lõi của Hadoop Framework và được tối ưu hóa cho việc lưu trữ và xử lý dữ liệu lớn với các đặc điểm sau:

a) Các đặc điểm chính của HDFS

Lưu trữ dữ liệu phân tán:

- Dữ liệu được chia thành các khối (block) và phân phối trên nhiều máy tính (nodes) trong một cụm Hadoop.
- Kích thước mặc định của mỗi block là 128 MB hoặc 256 MB, cho phép xử lý các tệp rất lớn.
- Khả năng chịu lỗi cao (Fault Tolerance):
  HDFS tạo ra nhiều bản sao (replicas) của mỗi block dữ liệu và lưu trữ chúng trên các nodes khác nhau.
- Mặc định, HDFS tạo 3 bản sao, giúp hệ thống vẫn hoạt động bình thường ngay cả khi một hoặc nhiều nodes gặp sự cố.

Thiết kế cho dữ liệu lớn:

- HDFS được tối ưu hóa cho việc lưu trữ và xử lý các tệp lớn (hàng trăm GB đến vài PB), thay vì tệp nhỏ.
- Dữ liệu trong HDFS chủ yếu được ghi một lần và đọc nhiều lần, phù hợp với các ứng dụng phân tích dữ liệu lớn.

Khả năng mở rộng (Scalability):

- HDFS có thể mở rộng bằng cách thêm các nodes mới vào cụm Hadoop mà không làm gián đoạn hoạt động của hệ thống.

Chi phí thấp:

- HDFS được thiết kế để hoạt động trên các phần cứng thương mại (commodity hardware), giảm chi phí so với việc sử dụng các hệ thống lưu trữ cao cấp.

b) Kiến trúc của HDFS

HDFS hoạt động theo mô hình Client-Server, với hai thành phần chính:

- NameNode:
  - Là nút quản lý trung tâm, chịu trách nhiệm theo dõi metadata của hệ thống (cấu trúc thư mục, thông tin về block, và vị trí lưu trữ của chúng).
  - Không lưu trữ dữ liệu thực tế, chỉ giữ thông tin về cách dữ liệu được phân phối trên các DataNodes.
- DataNode:
  - Là nơi lưu trữ thực tế các khối dữ liệu.
  - Thực hiện các nhiệm vụ như đọc/ghi dữ liệu theo yêu cầu từ NameNode.
- Secondary NameNode:

- Không phải bản sao dự phòng của NameNode, mà là một thành phần hỗ trợ, giúp lưu trữ các bản sao metadata định kỳ để giảm tải cho NameNode.

c) Quy trình hoạt động

Viết dữ liệu:

- Khi người dùng upload một tệp vào HDFS, tệp sẽ được chia thành các block và phân phối tới các DataNodes khác nhau.
- NameNode lưu trữ metadata, bao gồm vị trí của từng block.

Đọc dữ liệu:

- Người dùng gửi yêu cầu tới NameNode, nhận thông tin về vị trí lưu trữ block, sau đó truy cập trực tiếp các DataNodes để đọc dữ liệu.

Chịu lỗi:

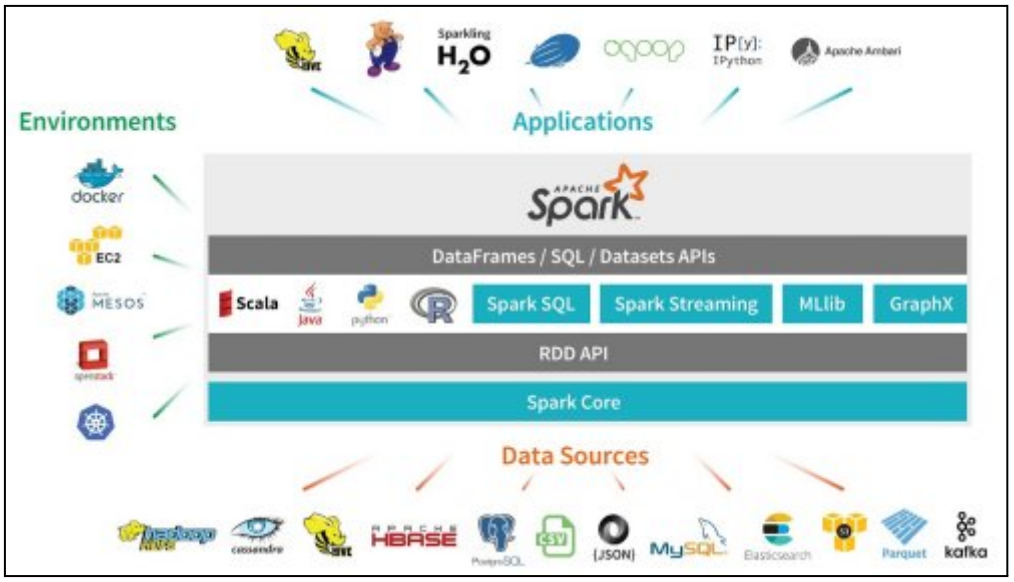
- Nếu một DataNode gặp sự cố, NameNode sẽ chỉ định bản sao trên các DataNode khác để phục hồi dữ liệu.

d) Ứng dụng của HDFS trong dự án

- Lưu trữ dữ liệu lịch sử lớn: HDFS là nơi lưu trữ dữ liệu lịch sử được thu thập từ Binance API.
- Kết hợp với Spark: Dữ liệu trên HDFS có thể được phân tích và xử lý hiệu quả thông qua Apache Spark để tạo ra các đặc trưng cho mô hình học máy.
- Tối ưu hóa hiệu suất: Cơ chế phân tán và khả năng chịu lỗi của HDFS đảm bảo hệ thống hoạt động liên tục, bất kể số lượng dữ liệu lớn hay sự cố phần cứng.

HDFS đóng vai trò trung tâm trong các kiến trúc xử lý dữ liệu lớn, hỗ trợ hiệu quả cho các ứng dụng phân tích và dự đoán.

2.1.3. Apache Spark



Môi trường triển khai Apache Spark thường bao gồm các thành phần sau:

- Spark Cluster: Spark chạy trên môi trường phân tán, bao gồm các node hoặc máy chủ thực thi các tác vụ. Một cluster Spark gồm nhiều thành phần như:
  - Driver: Đây là nơi bắt đầu và điều khiển ứng dụng Spark. Driver gửi các công việc (jobs) đến các node trong cluster và thu thập kết quả.
  - Cluster Manager: Quản lý tài nguyên của Spark cluster, phân phối công việc đến các worker node. Có thể sử dụng các công cụ cluster manager như YARN (Hadoop), Mesos, hoặc Kubernetes để quản lý và phân bổ tài nguyên.
  - Worker Nodes: Đây là các máy chủ thực thi các task (nhiệm vụ) của Spark. Mỗi worker node sẽ chạy các executor.
- Executor: Mỗi worker node sẽ chứa một hoặc nhiều executor. Executor là tiến trình thực thi các tác vụ (tasks) của Spark, bao gồm việc thực hiện các công việc tính toán và lưu trữ dữ liệu tạm thời (caching). Mỗi executor sẽ xử lý các phân vùng dữ liệu trong RDD (Resilient Distributed Datasets) hoặc DataFrame và trả kết quả về driver.
- Resilient Distributed Datasets (RDDs): Là cấu trúc dữ liệu phân tán trong Spark, giúp xử lý dữ liệu lớn theo cách hiệu quả và chịu lỗi. Dữ liệu trong RDD được phân phối trên các node trong cluster và Spark hỗ trợ các phép toán phân tán, như map, reduce, filter.
- Spark SQL: Cung cấp một ngôn ngữ truy vấn tương tự SQL để làm việc với dữ liệu dạng bảng. Spark SQL có thể sử dụng các dữ liệu trong HDFS, HBase,

Cassandra, và nhiều hệ thống khác. Nó hỗ trợ khả năng truy vấn, lọc và nhóm dữ liệu trên quy mô lớn.

- Spark Streaming: Cho phép xử lý dữ liệu thời gian thực bằng cách chia nhỏ dữ liệu thành các micro-batches. Spark Streaming sử dụng cùng một API với Spark core, điều này giúp dễ dàng triển khai các công việc xử lý cả batch lẫn thời gian thực.
- MLlib: Là thư viện học máy của Spark, cung cấp các thuật toán học máy phân tán. MLlib hỗ trợ các phương pháp như hồi quy, phân loại, clustering và giảm chiều dữ liệu, phù hợp với các mô hình học máy quy mô lớn.
  GraphX: Là thư viện xử lý đồ thị trong Spark, hỗ trợ việc xử lý và phân tích dữ liệu dạng đồ thị (graph). Các tác vụ đồ thị trong Spark có thể áp dụng trên các dữ liệu lớn với tính năng phân tán của Spark.
- Hadoop Ecosystem Integration: Spark thường được triển khai với Hadoop để tận dụng HDFS (Hadoop Distributed File System) cho lưu trữ và YARN cho quản lý tài nguyên. Spark có thể truy cập dữ liệu trong HDFS, HBase, Cassandra, hoặc các kho dữ liệu khác, làm cho Spark trở thành một công cụ mạnh mẽ cho xử lý dữ liệu lớn.

Môi trường triển khai Spark có thể được cấu hình linh hoạt, từ các cluster nhỏ đến các hệ thống quy mô lớn, và hỗ trợ nhiều phương thức lưu trữ dữ liệu (local, HDFS, hoặc cloud storage). Với khả năng tích hợp mạnh mẽ với các hệ thống như HDFS, Cassandra, và S3, Spark là lựa chọn phổ biến cho các ứng dụng xử lý dữ liệu lớn và phân tích dữ liệu trong thời gian thực.

2.1.4. Apache Airflow

Apache Airflow là một nền tảng mã nguồn mở được sử dụng để lập lịch, giám sát, và quản lý quy trình xử lý công việc (workflows) dưới dạng biểu đồ có hướng (DAG - Directed Acyclic Graph). Airflow hỗ trợ xây dựng và tự động hóa các pipeline dữ liệu phức tạp, rất phổ biến trong môi trường Big Data.

a) Đặc điểm chính của Apache Airflow

Quản lý workflows linh hoạt:

- Workflows được định nghĩa bằng mã Python, giúp người dùng dễ dàng mô hình hóa các công việc xử lý dữ liệu như ETL, batch processing hoặc trigger scripts.

Khả năng mở rộng (Scalability):

- Airflow có thể mở rộng để chạy trên nhiều máy chủ, phân phối công việc trên nhiều worker nodes bằng cách sử dụng các executor như Celery Executor, Kubernetes Executor.

Khả năng tích hợp mạnh mẽ:

- Hỗ trợ tích hợp với nhiều công nghệ và dịch vụ như Hadoop, Spark, HDFS, Kafka, AWS, Google Cloud, MySQL, và hơn thế nữa.

Giao diện người dùng trực quan (UI):

- Airflow cung cấp một giao diện web để người dùng có thể dễ dàng:
  - Theo dõi tiến trình của các tasks.
  - Quản lý và khởi chạy workflows.
  - Truy vấn logs để debug khi có lỗi xảy ra.

Điều phối dựa trên DAG:

- Công việc được tổ chức dưới dạng DAG, trong đó mỗi node đại diện cho một task, và các task được thực hiện theo thứ tự dựa trên mối quan hệ phụ thuộc (dependencies).

b) Kiến trúc của Apache Airflow

Airflow có kiến trúc module, bao gồm các thành phần chính:

Scheduler:

- Lập lịch và phân phối các task đến các worker dựa trên DAG đã định nghĩa.

Executor:

- Thực hiện các task theo lịch. Một số executor phổ biến:
  - Local Executor: Chạy trên một máy chủ duy nhất.
  - Celery Executor: Phân phối task trên nhiều worker thông qua message queue.
  - Kubernetes Executor: Tích hợp với Kubernetes để chạy task trên các pod.

Web Server:

- Cung cấp giao diện web để quản lý và theo dõi workflows.

Metadata Database:

- Lưu trữ thông tin về DAGs, trạng thái các task, và lịch sử thực hiện. Các cơ sở dữ liệu phổ biến như MySQL hoặc MySQL thường được sử dụng.

c) Ứng dụng của Apache Airflow trong dự án

Trong hệ thống dự đoán giá Bitcoin, Apache Airflow có vai trò quan trọng trong việc điều phối các pipeline xử lý dữ liệu batch:

- Thu thập dữ liệu lịch sử:
  - Airflow tự động lên lịch để gọi API Binance, thu thập dữ liệu lịch sử về giá và khối lượng giao dịch.

- Xử lý và lưu trữ:
  - Dữ liệu được xử lý qua các bước ETL (Extract, Transform, Load), lưu trữ vào HDFS hoặc các cơ sở dữ liệu như MySQL.
- Huấn luyện mô hình định kỳ:
  - Airflow lên lịch cho việc huấn luyện mô hình XGBoost trên dữ liệu lịch sử, đảm bảo mô hình luôn cập nhật với dữ liệu mới nhất.
- Quản lý lỗi và giám sát:
  - Trong trường hợp xảy ra lỗi (ví dụ: API không phản hồi), Airflow có thể tự động retry hoặc gửi thông báo qua email hoặc Slack.

d) Lợi ích khi sử dụng Apache Airflow

Tự động hóa quy trình:

- Airflow giảm thiểu công việc thủ công bằng cách tự động hóa toàn bộ pipeline xử lý dữ liệu.

Dễ bảo trì:

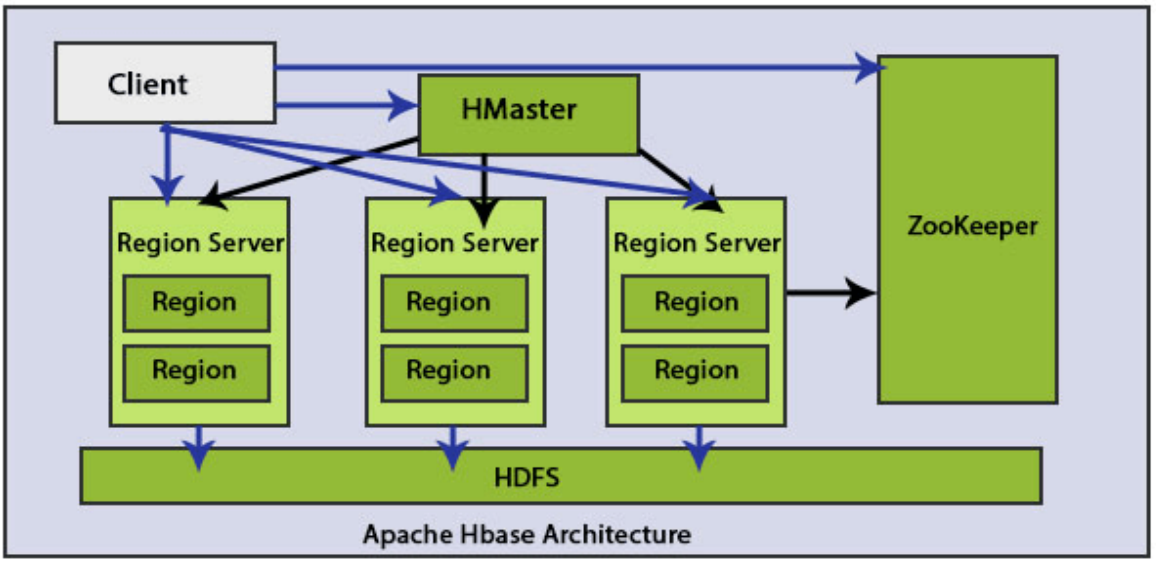
- Workflow được định nghĩa bằng mã Python nên dễ dàng bảo trì và mở rộng.

Khả năng theo dõi và khắc phục sự cố:

- Giao diện web trực quan giúp theo dõi tiến trình và log chi tiết để xử lý lỗi nhanh chóng.

Apache Airflow là công cụ mạnh mẽ cho quản lý workflows, đặc biệt trong các dự án xử lý dữ liệu lớn và dự đoán giá Bitcoin, nơi yêu cầu tính tự động hóa và khả năng giám sát cao.

2.1.5. Apache Hbase



Môi trường triển khai HBase thường bao gồm các thành phần và cấu hình sau:

- HBase Cluster: HBase hoạt động trong môi trường phân tán, và một cluster HBase thường bao gồm ba thành phần chính:
  - HMaster: Là thành phần điều phối, quản lý và giám sát các RegionServer trong cluster. HMaster chịu trách nhiệm phân phối các Region và theo dõi trạng thái của các RegionServer.
  - RegionServer: Mỗi RegionServer lưu trữ một hoặc nhiều "region". RegionServer chịu trách nhiệm thực hiện các thao tác đọc và ghi dữ liệu vào HBase. Mỗi RegionServer có thể chạy trên một hoặc nhiều máy chủ vật lý trong môi trường phân tán.
  - Regions: Dữ liệu trong HBase được chia thành các vùng (regions). Mỗi region chứa một phần của bảng và được phân phối đến các RegionServer khác nhau. Các region được phân phối lại khi cần để duy trì sự cân bằng tải trong cluster.
- Zookeeper: HBase sử dụng Apache Zookeeper để quản lý thông tin cấu hình, kiểm soát và đồng bộ hóa giữa các thành phần trong cluster. Zookeeper giúp duy trì thông tin trạng thái của các RegionServer, giám sát tình trạng hoạt động của các node trong cluster, và giúp phân phối lại region khi một RegionServer gặp sự cố.
- HBase Tables: Các bảng (tables) trong HBase được tổ chức theo kiểu NoSQL với các cột (column families) và mỗi bảng có thể chứa một lượng dữ liệu lớn. HBase lưu trữ dữ liệu dưới dạng cặp key-value với khả năng mở rộng theo chiều ngang, nghĩa là có thể thêm nhiều RegionServer để xử lý thêm dữ liệu.
  HFile: Dữ liệu trong HBase được lưu trữ dưới dạng HFiles. HFiles là định dạng

tệp trên đĩa mà HBase sử dụng để lưu trữ dữ liệu. Khi dữ liệu được ghi vào HBase, nó được ghi vào memstore (bộ nhớ tạm), và sau đó được đẩy vào HFile khi đạt một kích thước nhất định.

- HBase Clients: Các ứng dụng hoặc người dùng có thể truy cập vào HBase thông qua các client APIs. HBase hỗ trợ các API bằng Java, REST, và Thrift, giúp các ứng dụng có thể tương tác với dữ liệu lưu trữ trong HBase.
  HBase và Hadoop Integration: HBase thường được triển khai cùng với Hadoop để tận dụng HDFS (Hadoop Distributed File System) cho việc lưu trữ dữ liệu phân tán. Dữ liệu trong HBase được lưu trữ trên HDFS, và việc sử dụng HDFS giúp đảm bảo tính sẵn sàng cao và khả năng mở rộng của HBase.
- Backup and Replication: HBase hỗ trợ tính năng sao lưu (backup) và sao chép (replication) giữa các cluster. Việc sao chép giúp tăng tính sẵn sàng và khả năng phục hồi dữ liệu trong trường hợp cluster gặp sự cố.
  Security: HBase hỗ trợ các tính năng bảo mật như xác thực (authentication) và phân quyền (authorization) qua Kerberos hoặc các cơ chế bảo mật khác, giúp bảo vệ dữ liệu khỏi các truy cập trái phép.

Môi trường triển khai HBase cần có một cấu hình phần cứng đủ mạnh để đáp ứng nhu cầu lưu trữ và xử lý dữ liệu lớn, và việc triển khai với một hệ thống phân tán như Hadoop sẽ giúp mở rộng khả năng lưu trữ và xử lý dữ liệu theo chiều ngang.

2.1.6. MySQL

MySQL là một hệ quản trị cơ sở dữ liệu quan hệ mã nguồn mở, được thiết kế với tính năng mạnh mẽ, mở rộng tốt và khả năng tuân thủ các tiêu chuẩn SQL. MySQL rất phổ biến trong các ứng dụng xử lý dữ liệu lớn nhờ khả năng tích hợp tốt và hỗ trợ nhiều loại dữ liệu phức tạp.

a) Đặc điểm chính

Khả năng mở rộng cao:

- Hỗ trợ các bảng có kích thước lớn, dữ liệu phân tán, và có thể mở rộng thông qua các công cụ như mặc dù không sử dụng trong dự án này.

Tích hợp tốt với Big Data:

- Hỗ trợ JSON, XML, và nhiều loại dữ liệu phi cấu trúc khác, giúp dễ dàng tích hợp với các pipeline dữ liệu hiện đại.

Tính năng mạnh mẽ:

- Giao dịch ACID (Atomicity, Consistency, Isolation, Durability) đảm bảo tính toàn vẹn dữ liệu.
- Chức năng indexing nâng cao như B-Tree, GIN, và GiST giúp tối ưu hiệu suất.

Mã nguồn mở và cộng đồng lớn:

- Được cập nhật và hỗ trợ liên tục từ cộng đồng toàn cầu.

b) Ứng dụng trong dự án

Lưu trữ và quản lý dữ liệu đã xử lý từ lớp Batch.

Hỗ trợ các công cụ phân tích như Power BI để tạo báo cáo chi tiết.

2.1.7. Flask

Flask là một micro-framework web viết bằng Python, được thiết kế để xây dựng các ứng dụng web đơn giản và linh hoạt. Flask được sử dụng phổ biến để xây dựng API hoặc các ứng dụng web nhỏ gọn.

a) Đặc điểm chính

Nhẹ và linh hoạt:

- Flask chỉ cung cấp các thành phần cốt lõi, cho phép người dùng dễ dàng thêm các module cần thiết.

Hỗ trợ RESTful API:

- Dễ dàng triển khai các API để tương tác với dữ liệu từ các nguồn khác nhau.

Mở rộng dễ dàng:

- Tích hợp tốt với nhiều thư viện Python như SQLAlchemy, Pandas, hoặc NumPy.

Giao diện đơn giản:

- Hỗ trợ Jinja2 để xây dựng giao diện động.

b) Ứng dụng trong dự án

Cung cấp giao diện API để người dùng truy cập kết quả dự đoán thời gian thực từ HBase.

Hiển thị kết quả qua giao diện web hoặc tích hợp với các ứng dụng khác.

2.1.8. Docker

Docker là một nền tảng mã nguồn mở được sử dụng để xây dựng, đóng gói và chạy các ứng dụng trong các container. Container giúp tạo môi trường cô lập cho từng ứng dụng, đảm bảo tính nhất quán giữa các môi trường phát triển, kiểm thử và triển khai.

a) Đặc điểm chính

Khả năng cô lập môi trường:

- Mỗi container có đầy đủ các thành phần cần thiết (thư viện, công cụ) để chạy ứng dụng độc lập với hệ thống host.

Nhẹ và nhanh:

- Container nhẹ hơn máy ảo (VM), khởi động nhanh và không tiêu tốn quá nhiều tài nguyên.

Khả năng mở rộng và di động:

- Container được đóng gói sẵn, dễ dàng triển khai trên bất kỳ hệ thống nào hỗ trợ Docker.

Quản lý hiệu quả với Docker Compose:

- Cho phép định nghĩa và quản lý nhiều container trong một hệ thống.

b) Ứng dụng trong dự án

Đóng gói các thành phần (Kafka, Spark, HBase, Flask, v.v.) thành các container riêng biệt.

Đảm bảo tính đồng nhất và dễ triển khai hệ thống trên các môi trường khác nhau.

2.2. Thu thập và xử lý dữ liệu

2.2.1. Nguồn dữ liệu

Dữ liệu cho dự án này được lấy từ **Binance API**, một nền tảng giao dịch tiền điện tử phổ biến cung cấp API công khai cho việc truy xuất dữ liệu thị trường. Các loại dữ liệu được thu thập bao gồm:

- Lịch sử giá giao dịch (Price History): Bao gồm giá mở cửa, giá đóng cửa, giá cao nhất, giá thấp nhất (OHLC), và khối lượng giao dịch theo khoảng thời gian cụ thể (nến, tức là candlestick data).
- Khối lượng giao dịch (Trading Volume): Tổng lượng giao dịch trong một khoảng thời gian.
- Order Book: Dữ liệu sổ lệnh (order book) gồm các mức giá mua và bán.
- Thông tin thời gian thực (Real-time Data): Giá và khối lượng giao dịch hiện tại để phân tích ngắn hạn.

Dữ liệu được truy xuất bằng cách sử dụng các phương thức từ Binance REST API hoặc WebSocket API. Dữ liệu nến (candlestick) được sử dụng chính vì nó phù hợp để phân tích xu hướng giá và dễ dàng áp dụng các chỉ số kỹ thuật.

2.2.2. Xử lý dữ liệu

a) Thu thập dữ liệu

- Kết nối với Binance API bằng thư viện Python như **ccxt** hoặc **binance** để tải dữ liệu.
- Lựa chọn khung thời gian (interval) phù hợp, ví dụ: 1 phút, 15 phút, hoặc 1 giờ.

- Lưu trữ dữ liệu trong các tệp CSV hoặc cơ sở dữ liệu (như MongoDB hoặc MySQL) để dễ dàng xử lý sau này.

b) Làm sạch dữ liệu

- Xử lý dữ liệu thiếu: Loại bỏ hoặc điền giá trị vào các trường hợp thiếu dữ liệu (ví dụ: sử dụng giá trị trung bình của các điểm liền kề).
- Loại bỏ outliers: Phát hiện và loại bỏ các điểm bất thường trong dữ liệu giá hoặc khối lượng.
- Chuyển đổi kiểu dữ liệu: Đảm bảo các cột như thời gian (timestamp) được chuyển đổi sang định dạng thời gian chuẩn (datetime).

c) Chuyển đổi dữ liệu

- Chuẩn hóa dữ liệu: Nếu sử dụng nhiều nguồn dữ liệu, cần đưa các giá trị về cùng một thang đo để dễ dàng phân tích.
- Tạo các đặc trưng mới (Feature Engineering):
  - Tính toán các chỉ số kỹ thuật như **Moving Average (MA)**, **Relative Strength Index (RSI)**, **Bollinger Bands**, hoặc **MACD** để cung cấp thông tin về xu hướng giá.
  - Tạo các cột cho biến mục tiêu (Target Variable), ví dụ: thay đổi giá trong 1 giờ tiếp theo.
  - Đổi định dạng dữ liệu: Sắp xếp lại dữ liệu thành các chuỗi thời gian (time series) để phù hợp với mô hình học máy.

d) Xử lý thời gian

- Chuyển đổi thời gian từ UNIX timestamp (do API cung cấp) sang định dạng thời gian dễ đọc (UTC).
- Sử dụng các khung thời gian phù hợp với bài toán (short-term, medium-term, hoặc long-term).

e) Phân chia dữ liệu

- Dữ liệu được chia thành các phần:
  - Training set: Để huấn luyện mô hình.
  - Validation set: Để tinh chỉnh tham số.
  - Test set: Để đánh giá hiệu suất của mô hình.

Với quy trình này, dữ liệu sau khi thu thập và xử lý sẽ đảm bảo chất lượng, sẵn sàng để sử dụng trong các bước tiếp theo, như xây dựng mô hình dự đoán giá Bitcoin.

2.3. Mô hình hóa và dự đoán

Thuật toán: XGBoost (Extreme Gradient Boosting) là một thuật toán học máy rất mạnh mẽ, được sử dụng chủ yếu cho các bài toán phân loại và hồi quy. Đây là một dạng của Gradient Boosting Machine (GBM) nhưng tối ưu hơn với nhiều cải tiến về hiệu suất và tốc độ.

Cách hoạt động và các đặc điểm chính của thuật toán XGBoost:

2.3.1. Cơ chế hoạt động của XGBoost

XGBoost hoạt động theo nguyên lý gradient boosting, trong đó các cây quyết định (decision trees) được xây dựng tuần tự, mỗi cây sau được tạo ra để sửa các sai số của cây trước đó. Cơ chế này giúp cải thiện dự đoán và giảm thiểu sai sót. Quá trình này bao gồm các bước chính:

- Boosting: Là quá trình học tuần tự, mỗi mô hình mới được học nhằm khắc phục sai số của mô hình trước đó. Mỗi bước trong thuật toán cố gắng tối ưu hóa hàm mất mát (loss function).
- Gradient Descent: XGBoost sử dụng gradient descent để tìm ra hướng tối ưu cho việc cải thiện mô hình qua từng bước.
- Decision Trees: Mỗi lần học, XGBoost xây dựng một cây quyết định mới để cải thiện dự đoán dựa trên sai số của các cây trước.

2.3.2. Các đặc điểm nổi bật của XGBoost

- Regularization: Một trong những cải tiến quan trọng của XGBoost là khả năng điều chỉnh overfitting thông qua việc sử dụng các thuật toán regularization (L1 và L2). Điều này giúp mô hình không bị quá phức tạp, giảm thiểu khả năng overfitting.
- Tree Pruning: XGBoost sử dụng thuật toán "depth-first" để cắt tỉa cây quyết định. Điều này giúp tránh việc tạo ra những cây quyết định quá phức tạp và giúp tiết kiệm tài nguyên tính toán.
- Parallelization: XGBoost được tối ưu để thực thi song song, giúp tăng tốc quá trình huấn luyện trên dữ liệu lớn bằng cách tận dụng tài nguyên tính toán hiệu quả hơn.
- Handling Missing Data: XGBoost có thể tự động xử lý dữ liệu thiếu trong quá trình huấn luyện mà không cần phải làm sạch dữ liệu trước.
- Custom Loss Functions: Ngoài các hàm mất mát phổ biến như squared error, XGBoost còn hỗ trợ các hàm mất mát tùy chỉnh, giúp tăng cường tính linh hoạt trong việc tối ưu mô hình cho các bài toán khác nhau.

2.3.3. Các bước thực thi trong XGBoost

- Bước 1: Dự đoán ban đầu từ mô hình (thường là trung bình hoặc xác suất).
- Bước 2: Tính toán gradient của hàm mất mát để xác định hướng tối ưu cho việc cải thiện mô hình.
- Bước 3: Tạo một cây quyết định mới để điều chỉnh sai sót của mô hình hiện tại.
- Bước 4: Lặp lại quá trình này cho đến khi đạt được độ chính xác tối ưu.

2.3.4. Ứng dụng của XGBoost

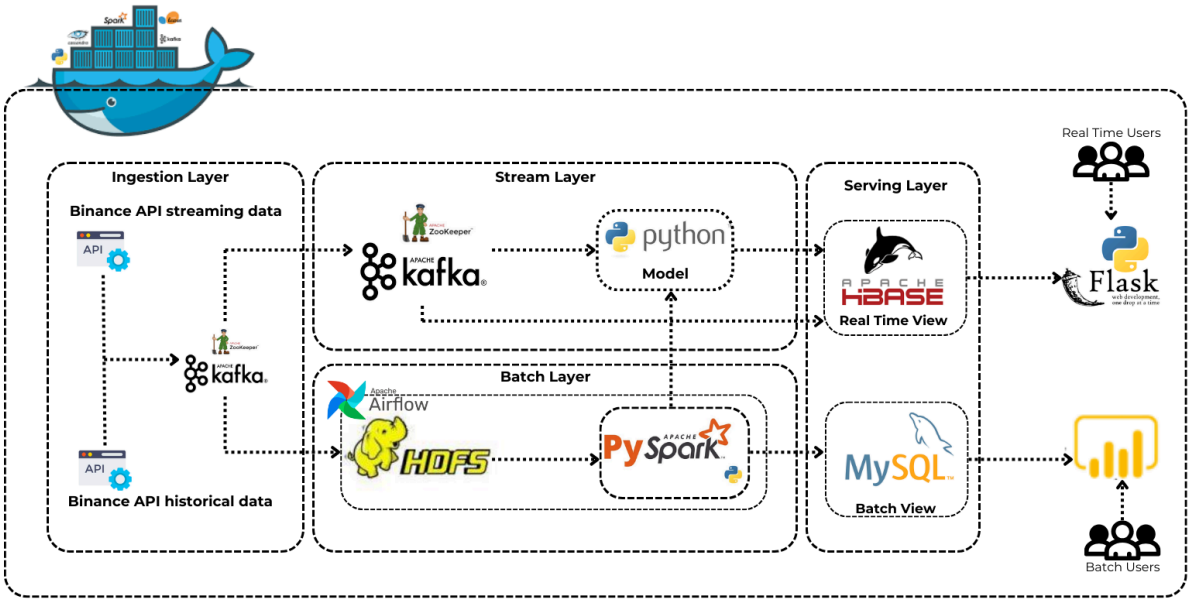
XGBoost được sử dụng trong nhiều bài toán học máy khác nhau như:

- Dự đoán giá trị (hồi quy).
- Phân loại (classification).
- Phân tích dữ liệu tín hiệu lớn, như phân tích các mẫu dữ liệu trong các bài toán kinh doanh, tài chính, và khoa học xã hội.
- Các cuộc thi machine learning (như Kaggle), nơi XGBoost được đánh giá là một trong những thuật toán mạnh mẽ nhất.

2.3.5. Ưu điểm của XGBoost

- Tốc độ và hiệu suất cao: XGBoost nhanh và có thể xử lý khối lượng dữ liệu rất lớn.
- Khả năng tổng quát tốt: XGBoost hoạt động rất hiệu quả trên nhiều loại dữ liệu khác nhau và có thể dễ dàng kết hợp với các kỹ thuật khác để cải thiện kết quả.
- Dễ dàng sử dụng: Thư viện XGBoost có API mạnh mẽ, hỗ trợ nhiều ngôn ngữ như Python, R, Java, và Scala, giúp người dùng dễ dàng triển khai và sử dụng.

2.4. Triển khai



2.4.1. Kiến trúc

Kiến trúc của hệ thống dự đoán giá Bitcoin được xây dựng theo mô hình Lambda Architecture, giúp xử lý đồng thời dữ liệu thời gian thực và dữ liệu batch (lô). Dưới đây là mô tả chi tiết từng thành phần trong hệ thống:

a) Ingestion Layer

Mục tiêu: Thu thập dữ liệu thời gian thực và dữ liệu lịch sử từ Binance API.

- Dữ liệu thời gian thực: Thu thập từ WebSocket API để lấy thông tin giá, khối lượng giao dịch ngay khi có thay đổi.
- Dữ liệu lịch sử: Sử dụng REST API của Binance để lấy dữ liệu quá khứ phục vụ huấn luyện mô hình và phân tích ban đầu.

Công nghệ sử dụng:

- Apache Kafka: Đóng vai trò làm message broker, quản lý và phân phối dữ liệu streaming từ Binance API.
- Apache Zookeeper: Điều phối và quản lý trạng thái của các phiên Kafka, đảm bảo độ tin cậy.

b) Stream Layer

Mục tiêu: Xử lý dữ liệu streaming từ Kafka để dự đoán giá Bitcoin theo thời gian thực.

- Luồng dữ liệu: Dữ liệu từ Kafka được chuyển đến các mô hình dự đoán được triển khai bằng Python.

- Thuật toán chính:
  - XGBoost: Mô hình gradient boosting được sử dụng để dự đoán giá Bitcoin dựa trên các đặc trưng tính toán từ dữ liệu streaming.

Đầu ra:

- Các kết quả dự đoán được lưu trữ trong HBase, hỗ trợ truy vấn thời gian thực để hiển thị nhanh chóng cho người dùng.

c) Batch Layer

Mục tiêu: Xử lý, phân tích dữ liệu lịch sử và huấn luyện mô hình dự đoán để cải thiện độ chính xác.

- Luồng dữ liệu:
  - Dữ liệu thô từ Kafka sẽ được nhận và lưu trữ vào DATALAKE trên HDFS và Database.
  - Sau đó, sử dụng PySpark để xử lý dữ liệu và đưa vào DATA WAREHOUSE.
  - PySpark cũng được dùng để huấn luyện lại mô hình cho phù hợp với phân bố mới của dữ liệu.
  - Apache Airflow: Lập lịch và điều phối các tác vụ xử lý dữ liệu và huấn luyện lại mô hình.

Kết quả đầu ra: Dữ liệu xử lý theo lô được lưu trữ trong HDFS và Database, phục vụ cho các báo cáo và phân tích sâu.

d) Serving Layer

Mục tiêu: Cung cấp giao diện cho người dùng truy vấn kết quả theo thời gian thực và dữ liệu batch.

- Hiển thị thời gian thực:
  - Kết quả dự đoán từ HBase được truy vấn và hiển thị qua Flask API.
  - Người dùng có thể xem dự đoán giá Bitcoin ngay lập tức qua các ứng dụng hoặc giao diện web.
- Hiển thị dữ liệu batch:
  - MySQL cung cấp dữ liệu cho các công cụ phân tích như Power BI, giúp người dùng truy vấn và xem các báo cáo chi tiết.

e) Lợi ích của kiến trúc

- Khả năng mở rộng: Sử dụng Kafka, PySpark và HDFS giúp xử lý lượng dữ liệu lớn với hiệu suất cao.

- Kết hợp thời gian thực và phân tích lô: Mô hình Lambda Architecture cung cấp cả dự đoán tức thời và báo cáo dài hạn.
- Tính linh hoạt: Kiến trúc có thể dễ dàng nâng cấp, thay đổi mô hình học máy hoặc mở rộng dung lượng lưu trữ.

Hệ thống được thiết kế để đảm bảo tính mạnh mẽ, hiệu quả và phù hợp với các bài toán xử lý dữ liệu lớn trong dự đoán giá Bitcoin.

2.4.2. Quy trình thực hiện

a) Lập kế hoạch và phân tích dự án

Xác định mục tiêu:

- Thu thập và phân tích dữ liệu giá Bitcoin thời gian thực và lịch sử giá Bitcoin.
- Dự đoán xu hướng giá Bitcoin bằng mô hình Machine Learning (XGBoost).
- Cung cấp giao diện để trực quan hóa dữ liệu cho người dùng, báo cáo.

Phân tích yêu cầu:

- Tốc độ xử lý: Luồng dữ liệu streaming cần được xử lý nhanh chóng (dưới 1 giây).
- Quy mô dữ liệu: Dữ liệu từ API Binance có thể sẽ rất lớn nếu thu thập dữ liệu liên tục.
- Khả năng mở rộng: Hệ thống cần hỗ trợ xử lý cả thời gian thực và dữ liệu batch.

b) Xây dựng từng lớp của kiến trúc

Ingestion Layer:

- Cấu hình Apache Kafka
  - Khởi tạo các topic: real-time-data, batch-data.
  - cài đặt Kafka broker và Zookeeper qua Docker.

Stream processing layer

- Phát triển Consumer để đọc dữ liệu từ Kafka
  - Tích hợp XGBoost để dự đoán giá Bitcoin dựa trên dữ liệu streaming.
- Tích hợp HBase
  - Lưu kết quả dự đoán thời gian thực vào HBase để truy xuất nhanh.

Batch processing layer:

- Nhận dữ liệu của sàn đã được gửi lên Kafka
- Xử lý dữ liệu với PySpark

- Phân tích và chuẩn bị dữ liệu.
- Huấn luyện mô hình bằng Spark MLLib

- Airflow
  - Lập lịch các task theo ngày để tải dữ liệu lịch sử và xử lý dữ liệu.
  - Lập lịch huấn luyện lại mô hình hàng tháng

Serving layer

- Cấu hình Flask API
  - Cung cấp API cho người dùng xem giá Bitcoin và kết quả dự đoán.
- Tích hợp Power BI
  - Kết nối Power BI với MySQL để hiển thị dashboard.

c) Triển khai dự án với Docker

Cấu hình Docker Compose và các Dockerfile cho từng thành phần cần thiết, bao gồm:

- Dockerfile cho Ingestion layer (API)
- Dockerfile cho Stream layer
- Dockerfile cho Batch layer
- Dockerfile cho Database
- Dockerfile cho Airflow

e) Kiểm thử và tối ưu hóa

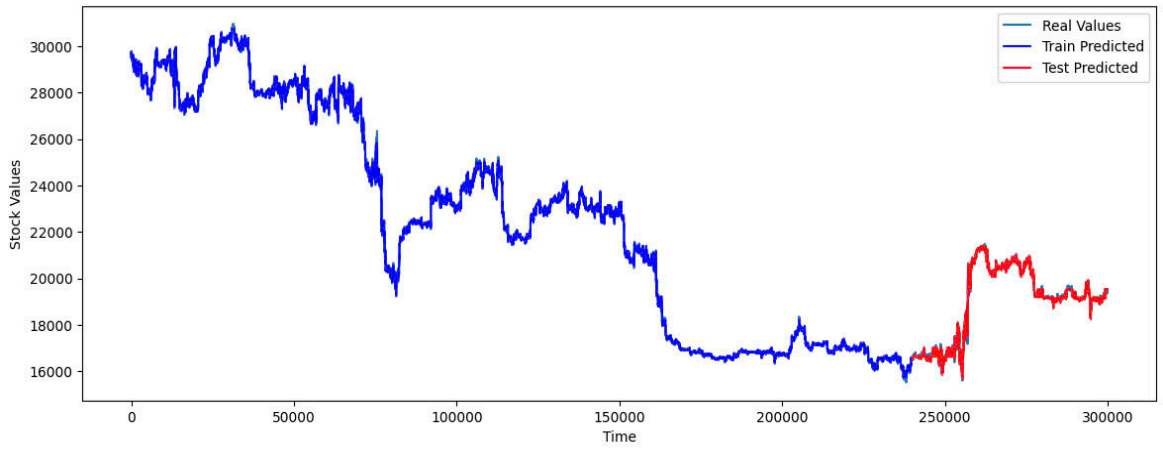
Kiểm thử hệ thống: Kiểm tra dữ liệu từ API→Kafka→Consumer→HBase.

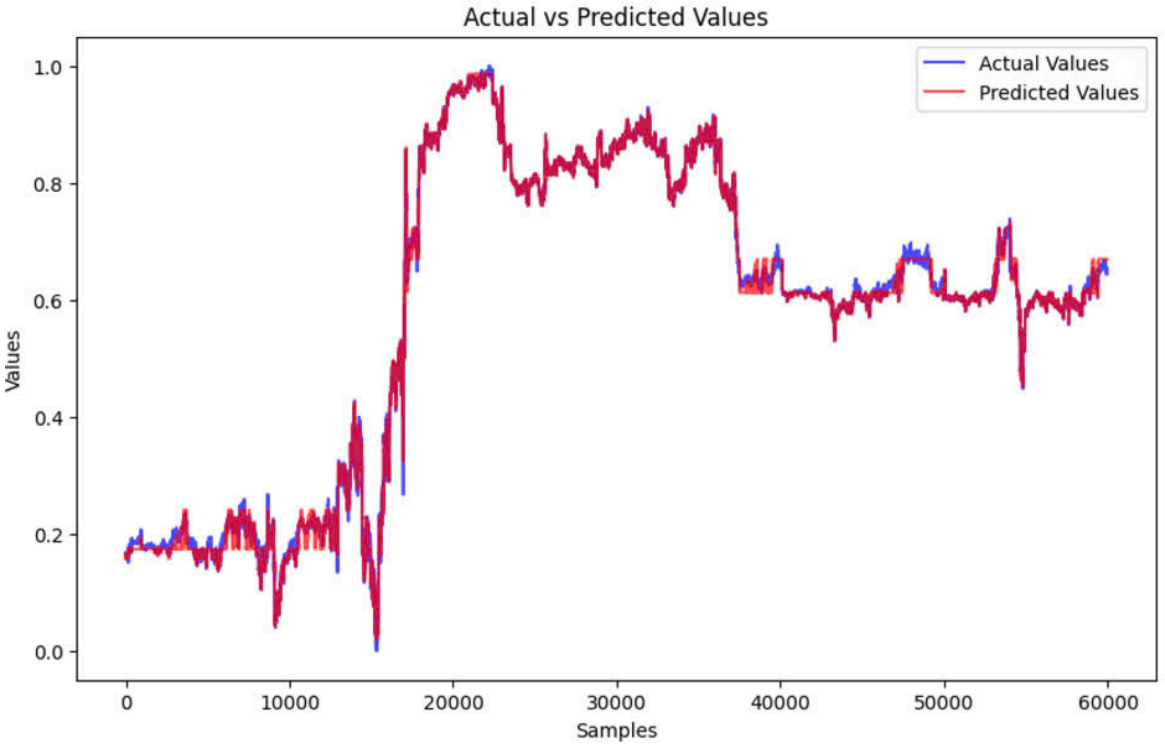
Tối ưu hóa:

- Cải thiện hiệu năng các thành phần.
- Tối ưu mô hình XGBoost với dữ liệu thực tế.

2.5. Kết quả

- Thu thập được dữ liệu real-time thành công: Dữ liệu Bitcoin từ Binance API được cập nhật mỗi giây qua Apache Kafka, được đảm bảo với độ trễ 100ms.
- Xử lý dữ liệu streaming hiệu quả: Mô hình dự đoán giá Bitcoin với RMSE khoảng 5-15$.





- Hệ thống hoạt động ổn định: Với thiết kế microservices (Kafka, HBase, Flask), hệ thống có thể xử lý đồng thời nhiều sự kiện mỗi phút mà không xảy ra tình trạng tắc nghẽn.
- Dashboard trực quan: Tích hợp Power BI giúp mang đến trải nghiệm dễ hiểu với các biểu đồ xu hướng giá theo thời gian và lịch sử.
- Khả năng mở rộng: Hệ thống có thể sẵn sàng mở rộng với với nhu cầu xử lý nhiều loại crypto khác nhau.

CHƯƠNG 3: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

3.1. Kết luận

Dự án xây dựng hệ thống dự đoán giá Bitcoin dựa trên dữ liệu lớn đã đạt được nhiều kết quả đáng kể, thể hiện sự phối hợp hiệu quả giữa công nghệ Big Data, Machine Learning, và các công cụ trực quan hóa dữ liệu. Hệ thống không chỉ xử lý dữ liệu theo thời gian thực một cách ổn định mà còn cung cấp các dự đoán giá có độ chính xác cao, mang lại giá trị thực tiễn cho các nhà đầu tư và người dùng cá nhân.

3.1.1. Tóm tắt thành tựu

- Hiệu suất cao: Hệ thống streaming với Apache Kafka và Spark đã chứng minh khả năng xử lý dữ liệu lớn một cách nhanh chóng và ổn định.
- Trực quan hóa rõ ràng: Dashboard Power BI giúp người dùng dễ dàng theo dõi và phân tích xu hướng giá.
- Tiềm năng mở rộng: Nền tảng công nghệ sẵn sàng hỗ trợ nhiều loại tài sản tài chính khác.

3.1.2. Ý nghĩa thực tiễn

Dự án không chỉ mang lại lợi ích ngắn hạn trong việc dự đoán giá Bitcoin mà còn mở ra tiềm năng phát triển các ứng dụng trong các lĩnh vực tài chính khác như phân tích cổ phiếu, dự đoán xu hướng thị trường, hoặc tối ưu hóa danh mục đầu tư.

3.1.3. Thách thức và hạn chế

Thách thức:

- Xử lý các outliers.
- Kiến thức chuyên môn còn chưa cao.
- Thời gian còn khá gấp rút để hoàn thiện toàn bộ tiến trình đề ra.

Hạn chế:

- Phạm vi dự án còn giới hạn ở dữ liệu Crypto và các loại dữ liệu tương tự (như dữ liệu chứng khoán,...)
- Mới chỉ dự đoán ngắn hạn.

3.2. Hướng phát triển

- Nắm bắt, dự đoán với độ chính xác cao.
- Phát triển thành sản phẩm hữu ích cho người đầu tư, giúp xác định thời điểm.
- Nâng cấp, sử dụng các mô hình học sâu khác để dự đoán được dài hạn.

TÀI LIỆU THAM KHẢO

[1]Omole Oluwadamilare, and Enke David. 2024. “Deep learning for Bitcoin price direction prediction: models and trading strategies empirically compared - Financial Innovation.” *Financial Innovation*, August 5, 2024.

[2]Ji Suhwan, Kim Jongmin, and Im Hyeonseung. 2019. “A comparative study of bitcoin price prediction using deep learning.” *Mathematics*, 2019.

[3]Jaquart Patrick, Dann David, and Martin Carl. 2020. “Machine Learning for Bitcoin Pricing—A Structured.” 2020.

NHIỆM VỤ CỦA CÁC THÀNH VIÊN

Họ và tên	Công việc
Phạm Thành Long	- Tìm hiểu chủ đề - Refactor và Docker hóa dự án - Thuyết trình - Testing - Viết report phần kết quả - Làm slide
Nguyễn Đức Minh	- Tìm hiểu chủ đề - Xử lý batch layer - QA - Viết báo cáo phần Batch - Làm slide
Phan Văn Hiếu	- Tìm hiểu chủ đề - Xử lý stream layer - QA - Viết báo cáo phần Stream - Làm slide
Trần Tiến Nam	- Tìm hiểu chủ đề - Xử lý phần ingestion - QA - Viết báo cáo phần bài toán, dữ liệu - Làm slide