ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

— *** —

BÁO CÁO MÔN HỌC

LƯU TRỮ VÀ XỬ LÝ DỮ LIỆU LỚN

Tên đề tài: Xây Dựng Hệ Thống Lưu Trữ Phân Tích và Dự Đoán Dữ Liệu Giá Tiền Điện Tử

Giảng viên hướng dẫn: TS. Trần Văn Đặng

Nhóm sinh viên thực hiện: Đỗ Đức Long - 20225034

Bùi Vũ Đức Nghĩa - 20224883

Lê Tiến Khôi - 20225021

Trần Quang Minh - 20225047

Hà Nội, tháng 5 năm 2025

Mục lục

Danh sách hình vẽ

Phân công công việc

Phân công công việc chi tiết của từng thành viên được lưu trữ trong bảng Google Sheets sau:

- https://docs.google.com/spreadsheets/d/14JVTDl5i-NOyU7rRBV1xJPjE6kIaL-sBpVT$_g r2l9j4$

Mã nguồn dự án

Toàn bộ mã nguồn của dự án được quản lý và công khai trên GitHub tại liên kết sau:

- https://github.com/ducnghia0509/Big$_D ata_P r$

1 Giới thiệu

1.1 Bối cảnh và Lý do chọn đề tài

Thị trường tiền điện tử với sự biến động không ngừng và khối lượng giao dịch khổng lồ đã tạo ra một nguồn dữ liệu lớn (Big Data) phong phú. Việc thu thập, lưu trữ, xử lý và phân tích hiệu quả nguồn dữ liệu này mang lại giá trị to lớn cho việc ra quyết định đầu tư, theo dõi xu hướng và hiểu biết thị trường. Nhu cầu về các công cụ hỗ trợ có khả năng xử lý lượng dữ liệu này một cách hiệu quả, đồng thời cung cấp thông tin cập nhật theo thời gian thực ngày càng trở nên cấp thiết.

1.2 Mục tiêu của Đề tài

Đề tài này nhằm mục tiêu xây dựng một "Hệ thống Lưu trữ và Phân tích Dữ liệu Giá Tiền Điện Tử"với các mục tiêu cụ thể sau:

- Xây dựng một pipeline dữ liệu hoàn chỉnh, áp dụng kiến trúc Lambda, để thu thập, lưu trữ, xử lý, phân tích và trực quan hóa dữ liệu giá tiền điện tử (OHLCV).
- Triển khai Batch Layer để xử lý dữ liệu lịch sử, tính toán các chỉ báo kỹ thuật cơ bản (ví dụ: SMA) và tạo ra các "master data views".
- Triển khai Speed Layer để xử lý dữ liệu OHLCV 1 phút theo thời gian thực, cung cấp giá đóng cửa mới nhất của nến 1 phút, các thống kê theo cửa sổ trượt (dựa trên giá đóng cửa của nến 1 phút) và dữ liệu cho biểu đồ real-time.
- Sử dụng Apache Kafka làm message queue cho Speed Layer và HDFS để lưu trữ dữ liệu thô cho Batch Layer cũng như checkpoint cho Spark.
- Sử dụng Apache Spark (Spark SQL và Spark Streaming) làm công cụ xử lý dữ liệu chính cho cả Batch và Speed Layer.
- Sử dụng Elasticsearch làm Serving Layer, lưu trữ dữ liệu đã xử lý từ cả hai lớp để phục vụ truy vấn nhanh.
- Phát triển một ứng dụng web (dashboard) bằng Flask và các công nghệ frontend (HTML, JS, Chart.js) để người dùng có thể tương tác và theo dõi:
    - Dữ liệu lịch sử (khung thời gian 1 giờ) với các chỉ báo và tùy chọn lọc thời gian.
    - Các thống kê real-time và biểu đồ giá 1 phút.

- Kết quả dự đoán giá đóng cửa cho 24 giờ tiếp theo sử dụng mô hình Machine Learning (XGBoost) đã được huấn luyện trước.

- Thiết lập cơ chế cập nhật dữ liệu lịch sử định kỳ (hàng giờ) và tự động hóa các tác vụ bằng `cron`.

- Tích hợp và sử dụng một mô hình Machine Learning (XGBoost) đã được huấn luyện trước để thực hiện dự đoán giá trong tương lai ngắn hạn (24 giờ).

1.3 Phạm vi của Đề tài

- **Dữ liệu:**
  - Tập trung vào dữ liệu OHLCV của các cặp tiền điện tử phổ biến (ví dụ: BTC/USDT, ETH/USDT) từ sàn giao dịch Binance.
  - Khung thời gian cho dữ liệu lịch sử được xử lý bởi Batch Layer là 1 giờ (`1h`).
  - Khung thời gian cho dữ liệu được xử lý bởi Speed Layer và phục vụ cho biểu đồ real-time là 1 phút (`1m`).

- **Phân tích và Dự đoán:**
  - **Batch Layer:** Tính toán các chỉ báo kỹ thuật cơ bản như Simple Moving Average (SMA7, SMA30) trên dữ liệu 1 giờ.
  - **Speed Layer:** Tính toán giá đóng cửa mới nhất của nến 1 phút, giá trung bình, min, max, số lượng nến 1 phút trong một cửa sổ thời gian nhất định (ví dụ: 10 phút) từ dữ liệu 1 phút.
  - **Dự đoán:** Sử dụng mô hình XGBoost đã được huấn luyện trước (dựa trên dữ liệu lịch sử 1 giờ) để dự đoán giá đóng cửa cho 24 giờ tiếp theo. Dự đoán này được thực hiện theo yêu cầu của người dùng thông qua API, không lưu trữ kết quả dự đoán dài hạn vào cơ sở dữ liệu.

- **Công nghệ chính:** Python (bao gồm Flask, ccxt, kafka-python, pandas, numpy, scikit-learn, xgboost, joblib, pandas-ta), Apache Kafka, HDFS, Apache Spark (Spark SQL, Spark Streaming), Elasticsearch, HTML, CSS, JavaScript (Chart.js, Moment.js).

- **Môi trường triển khai:** Máy ảo VMware chạy hệ điều hành Ubuntu, với các thành phần được cài đặt và cấu hình để hoạt động trên một node (chế độ pseudo-distributed cho HDFS/Spark khi thích hợp).

2 Big Data và Kiến Trúc Lambda

2.1 Big Data và Đặc điểm 3V

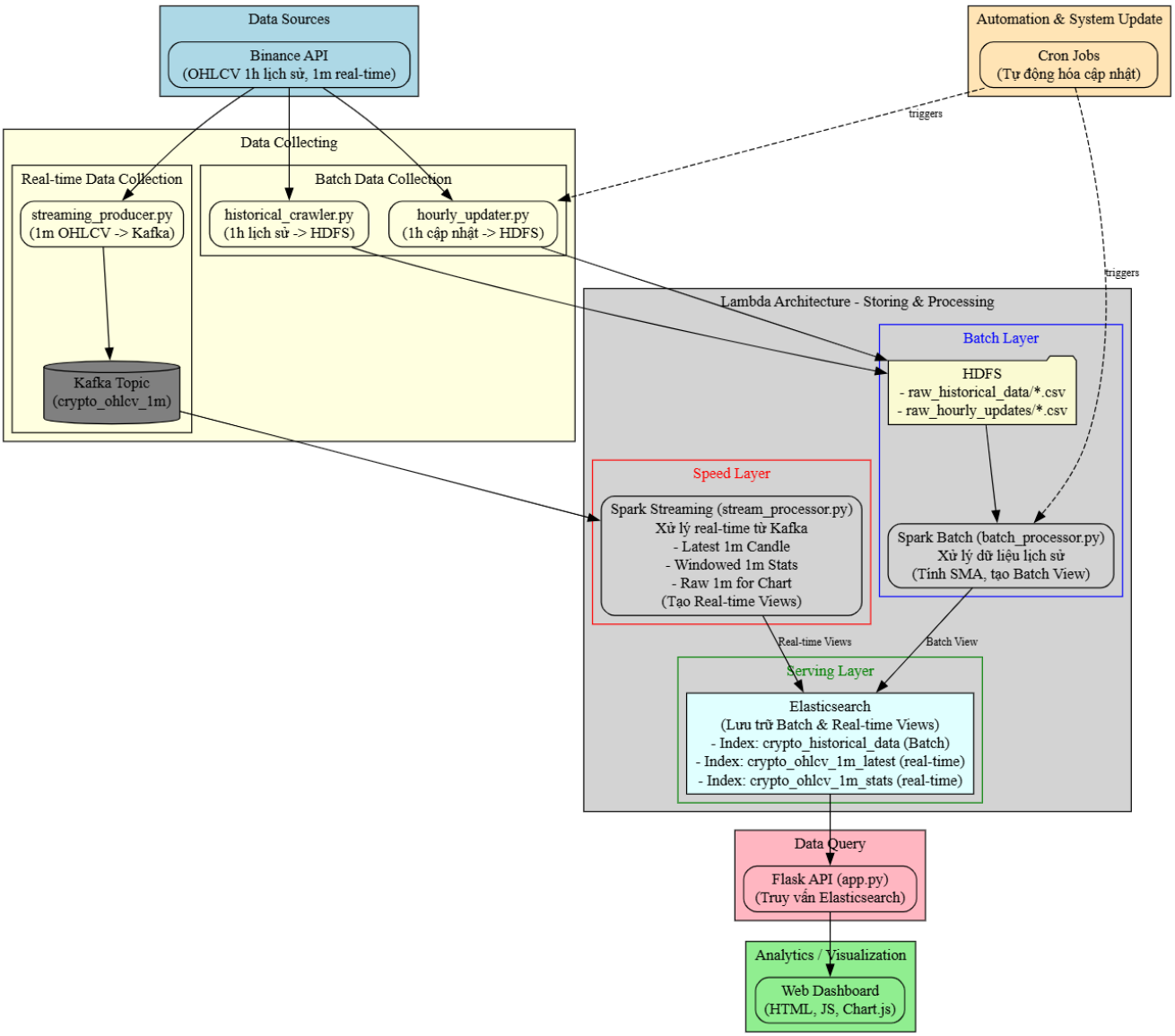
Dữ liệu dự án (giá tiền điện tử lịch sử và real-time) thể hiện các đặc điểm của Big Data:

- **Volume (Khối lượng):** Dữ liệu lịch sử kéo dài nhiều năm với các khung thời gian chi tiết (giờ, phút) tạo ra khối lượng lớn.
- **Velocity (Tốc độ):** Dữ liệu giá biến động liên tục, đặc biệt là dữ liệu 1 phút, đòi hỏi khả năng xử lý tốc độ cao.
- **Variety (Đa dạng):** Hiện tại tập trung vào dữ liệu giá (OHLCV), nhưng kiến trúc có thể mở rộng cho các loại dữ liệu khác (tin tức, sentiment).

2.2 Kiến Trúc Lambda

Được chọn vì khả năng kết hợp xử lý batch (cho độ chính xác và toàn vẹn dữ liệu lịch sử) và xử lý stream (cho độ trễ thấp và thông tin cập nhật). Kiến trúc bao gồm 3 lớp chính:

- **Batch Layer:** Xử lý toàn bộ tập dữ liệu gốc (master dataset) để tạo ra các "batch views"hoàn chỉnh và chính xác.
- **Speed Layer (Streaming Layer):** Xử lý dữ liệu mới đến với độ trễ thấp, bù đắp cho độ trễ của Batch Layer và cung cấp "real-time views".
- **Serving Layer:** Hợp nhất kết quả từ Batch Layer và Speed Layer để phục vụ các truy vấn.



Hình 1: Sơ đồ kiến trúc Lambda áp dụng trong dự án.

3 Các Công Cụ và Nền Tảng Chính

3.1 Apache Kafka

Apache Kafka (phiên bản 3.7.0) được sử dụng làm hệ thống message queue trung gian trong Speed Layer. Producer (`ohlcv_1m_producer.py`) gửi dữ liệu OHLCV 1 phút vào topic `crypto_ohlcv_1m`. Spark Streaming consumer trong `stream_processor.py` đọc dữ liệu từ topic này để xử lý theo thời gian thực.

3.2 HDFS (Hadoop Distributed FileSystem)

Hệ thống file phân tán Hadoop (phiên bản 3.3.6) được dùng để:

- Lưu trữ dữ liệu lịch sử OHLCV thô (định dạng CSV) được thu thập bởi `historical_crawler.py` tại thư mục `/user/username/crypto_project/raw_historical_data/`.
- Lưu trữ các file CSV cập nhật hàng giờ được thu thập bởi `hourly_updater.py` tại thưmuc `/user/username/crypto_project/raw_hourly_updates/`.
- Lưu trữ checkpoint cho các stateful operations của Spark Streaming (ví dụ: thống kê cửa sổ) tại `/user/username/crypto_project/checkpoint/`.

3.3 Apache Spark (v3.4.3)

Apache Spark là nền tảng xử lý dữ liệu chính của hệ thống.

- **Spark SQL & DataFrame API:** Được sử dụng trong `batch_processor.py` để đọc dữ liệu CSV từ HDFS, thực hiện các phép biến đổi (trích xuất symbol/timeframe, chuyển đổi timestamp), tính toán các chỉ báo kỹ thuật (SMA 7, SMA 30), và ghi/upsert kết quả vào Elasticsearch.
- **Spark Streaming (Structured Streaming):** Được sử dụng trong `stream_processor.py` để:
  - Đọc dữ liệu OHLCV 1 phút từ Kafka topic `crypto_ohlcv_1m`.
  - Xử lý và ghi nến OHLCV 1 phút mới nhất vào index `crypto_ohlcv_1m_latest` của Elasticsearch.
  - Tính toán các thống kê theo cửa sổ trượt (ví dụ: 10 phút trượt mỗi 1 phút) trên giá đóng cửa của dữ liệu 1 phút (avg, min, max, count) và ghi vào index `crypto_ohlcv_1m_stats`.
  - Ghi dữ liệu OHLCV 1 phút thô (đã parse) vào các index Elasticsearch hàng ngày (`crypto_ohlcv_1m_chartdata-YYYY-MM-DD`) để phục vụ cho biểu đồ real-time.

Kết nối giữa Spark và Elasticsearch được thực hiện thông qua connector `elasticsearch-spark-30_2.12-8.14.0.jar`.

3.4 Elasticsearch (v8.14.0)

Elasticsearch đóng vai trò là Serving Layer, cung cấp khả năng lưu trữ và truy vấn nhanh chóng cho dữ liệu đã được xử lý:

- **Index `crypto_historical_data`:** Lưu trữ dữ liệu lịch sử OHLCV (khung thời gian 1 giờ) cùng với các chỉ báo SMA đã được tính toán bởi Batch Layer. Dữ liệu được cập nhật hàng giờ.
- **Index `crypto_ohlcv_1m_latest`:** Lưu trữ thông tin đầy đủ (O,H,L,C,V, timestamp) của nến 1 phút mới nhất cho mỗi symbol, được cập nhật bởi Speed Layer.
- **Index `crypto_ohlcv_1m_stats`:** Lưu trữ các thống kê theo cửa sổ (ví dụ: giá trung bình, min, max trong 10 phút qua) được tính từ dữ liệu OHLCV 1 phút, cập nhật bởi Speed Layer.
- **Indices `crypto_ohlcv_1m_chartdata-YYYY-MM-DD`:** Lưu trữ chi tiết dữ liệu OHLCV 1 phút thô, được ghi theo ngày và quản lý vòng đời bằng ILM (ví dụ: tự động xóa sau 1-2 ngày) để phục vụ biểu đồ real-time 1 phút.

3.5 Python (v3.8.10) và Các Thư Viện Chính

Ngôn ngữ lập trình Python được sử dụng rộng rãi trong dự án:

- `ccxt`: Thu thập dữ liệu OHLCV từ API sàn Binance.
- `kafka-python`: Gửi dữ liệu vào Kafka trong script `ohlcv_1m_producer.py`.
- `elasticsearch-py`: Tương tác với Elasticsearch từ ứng dụng Flask.
- `Flask`: Xây dựng backend API cho ứng dụng web.
- `pandas`: Xử lý dữ liệu dạng bảng trong các script thu thập.
- `python-dotenv`: Quản lý các biến môi trường cấu hình.
- `schedule`: (Nếu sử dụng cho producer thay vì cron) Lập lịch chạy tác vụ.

Toàn bộ danh sách thư viện được quản lý trong file `requirements.txt`.

3.6 Frontend và Trực Quan Hóa

Giao diện người dùng được xây dựng bằng HTML, CSS và JavaScript.

- **Chart.js:** Thư viện JavaScript để vẽ các biểu đồ đường (line chart) cho dữ liệu lịch sử và real-time.
- **Moment.js:** Thư viện JavaScript để xử lý và định dạng thời gian trên frontend.
- **chartjs-plugin-zoom:** Plugin cho Chart.js để hỗ trợ zoom và pan trên biểu đồ lịch sử.

3.7 Các Công Cụ Khác

- **VMware và Ubuntu:** Hệ thống được triển khai và thử nghiệm trên máy ảo VMware chạy hệ điều hành Ubuntu.
- **Crontab:** Được sử dụng để lập lịch chạy tự động các script cập nhật dữ liệu hàng giờ (`hourly_updater.py` và `run_batch_processor.sh`).

4 Luồng Dữ Liệu Tổng Quan

Hệ thống xử lý dữ liệu theo hai luồng chính song song của kiến trúc Lambda:

1. **Luồng Batch (Dữ liệu Lịch sử & Cập nhật hàng giờ):**
   - `historical_crawler.py` (chạy một lần) thu thập dữ liệu lịch sử OHLCV (1h) từ Binance, lưu file CSV vào HDFS (`raw_historical_data/`).
   - `hourly_updater.py` (chạy mỗi giờ) thu thập dữ liệu OHLCV (1h) của giờ trước đó, lưu file CSV mới vào HDFS (`raw_hourly_updates/`).
   - `batch_processor.py` (Spark job, chạy mỗi giờ) đọc tất cả dữ liệu CSV từ cả hai thư mục trên HDFS, tính toán SMA, và upsert kết quả vào index `crypto_historical_data` trên Elasticsearch.
2. **Luồng Speed (Dữ liệu Real-time 1 phút):**
   - `ohlcv_1m_producer.py` (chạy liên tục) thu thập dữ liệu OHLCV (1m) mới nhất từ Binance, gửi vào Kafka topic `crypto_ohlcv_1m`.
   - `stream_processor.py` (Spark Streaming job, chạy liên tục) đọc từ Kafka:
     - Ghi nến 1 phút mới nhất (O, H, L, C, V) vào chỉ mục Elasticsearch `crypto_ohlcv_1m_latest`.
     - Tính toán thống kê (giá đóng cửa trung bình, nhỏ nhất, lớn nhất) trong cửa sổ 10 phút trượt theo từng phút, và ghi kết quả vào chỉ mục `crypto_ohlcv_1m_stats`.
     - Ghi dữ liệu OHLCV 1 phút thô vào chỉ mục Elasticsearch theo ngày `crypto_ohlcv_1m_chartdata-YYYY-MM-DD` để phục vụ hiển thị biểu đồ.
3. **Phục vụ truy vấn:** Ứng dụng Flask (`app.py`) cung cấp API, đọc dữ liệu từ các index Elasticsearch tương ứng để trả về cho frontend.
4. **Trực quan hóa:** Frontend (`realtime_dashboard.html`, `historical_data.html`) gọi API và sử dụng Chart.js để hiển thị.

5 Triển Khai Chi Tiết Các Thành Phần

5.1 Batch Layer

5.1.1 Thu thập dữ liệu lịch sử (`historical_crawler.py` và `hourly_updater.py`)

Cả hai script đều sử dụng thư viện `ccxt` để kết nối tới API của sàn Binance và lấy dữ liệu OHLCV.

- `historical_crawler.py`: Lấy dữ liệu trong một khoảng thời gian dài được định nghĩa bởi `CRYPTO_START_DATE` và `CRYPTO_END_DATE` trong file `.env`. Dữ liệu được lưu dưới dạng các file CSV (ví dụ: `BTC_USDT_1h.csv`) vào thư mục `raw_historical_data/` trên HDFS.
- `hourly_updater.py`: Được thiết kế để chạy mỗi giờ. Script này đầu tiên sẽ truy vấn Elasticsearch (index `crypto_historical_data`) để tìm timestamp của nến cuối cùng đã được xử lý cho mỗi cặp symbol/timeframe. Sau đó, nó sẽ lấy dữ liệu từ nến tiếp theo đó cho đến đầu giờ hiện tại. Dữ liệu mới được lưu vào các file CSV riêng biệt (ví dụ: `BTC_USDT_1h_update_20250521_1500.csv`) trong thư mục `raw_hourly_updates/` trên HDFS.

Cả hai script đều đảm bảo dữ liệu CSV có các cột: `timestamp` (milliseconds), `open`, `high`, `low`, `close`, `volume`, `datetime_str`.

5.1.2 Xử lý Batch (`batch_processor.py`)

Đây là một Spark job được viết bằng PySpark, thực hiện các công việc sau:

1. **Đọc dữ liệu:** Đọc tất cả các file CSV từ cả hai thư mục `raw_historical_data/` và `raw_hourly_updates/` trên HDFS. Schema được định nghĩa sẵn để đảm bảo tính nhất quán.
2. **Trích xuất metadata:** Từ tên file đầu vào, trích xuất thông tin `symbol` (ví dụ: "BTC_USDT") và `timeframe` (ví dụ: "1h"). Cần có logic regex phù hợp để xử lý các định dạng tên file khác nhau.
3. **Xử lý timestamp:** Chuyển đổi cột `timestamp` (milliseconds từ CSV) sang kiểu `TimestampType` của Spark (để phục vụ windowing) và sang kiểu `LongType` (số giây kể từ epoch) để lưu trữ vào Elasticsearch và tạo `doc_id`.
4. **Tính toán Feature Engineering:** Tính toán các đường trung bình động SMA 7 và SMA 30 dựa trên giá đóng cửa (`close`) cho mỗi `symbol` và `timeframe`.

5. **Chuẩn bị ghi vào Elasticsearch:**
    - Tạo cột `doc_id` duy nhất cho mỗi nến dựa trên công thức: `symbol + "_"+ timeframe + "_hist_"+ timestamp_seconds`.
    - Chọn các cột cần thiết để lưu: `timestamp` (số giây), `symbol`, `timeframe`, `open`, `high`, `low`, `close`, `volume`, `sma_7`, `sma_30`.
6. **Ghi/Upsert vào Elasticsearch:** Sử dụng Spark-Elasticsearch connector để ghi DataFrame kết quả vào index `crypto_historical_data`.
    - `mode("append")`: Để không xóa toàn bộ index mỗi lần chạy.
    - `option("es.mapping.id", "doc_id")`: Chỉ định cột `doc_id` làm ID của document trong Elasticsearch.
    - `option("es.write.operation", "upsert")`: Nếu document với `doc_id` đó đã tồn tại, nó sẽ được cập nhật; nếu chưa, một document mới sẽ được tạo. Điều này đảm bảo dữ liệu luôn là mới nhất và không bị trùng lặp sau các lần cập nhật.

5.2 Speed Layer

5.2.1 Thu thập dữ liệu OHLCV 1 phút (`ohlcv_1m_producer.py`)

Script Python này chạy liên tục (hoặc theo lịch trình mỗi phút):

- Sử dụng `ccxt` để fetch `N` (ví dụ: 2) nến OHLCV 1 phút gần nhất từ Binance cho các symbol trong `.env`.
- Chọn nến cuối cùng (được cho là đã đóng hoặc gần đóng nhất).
- Tạo một message JSON chứa: `timestamp` (milliseconds), `symbol`, `timeframe` ("1m"), `open`, `high`, `low`, `close`, `volume`, `datetime_str`.
- Gửi message này vào Kafka topic `crypto_ohlcv_1m` bằng thư viện `kafka-python`.

5.2.2 Xử lý Streaming (`stream_processor.py`)

Spark Streaming job này xử lý dữ liệu OHLCV 1 phút:

1. **Đọc từ Kafka:** Tạo một input DataFrame đọc từ topic `crypto_ohlcv_1m`.
2. **Parse dữ liệu:** Chuyển đổi message JSON từ Kafka thành các cột DataFrame theo schema đã định nghĩa (`kafka_ohlcv_1m_schema`). Cột `timestamp` (milliseconds) được chuyển thành `event_timestamp` (Spark `TimestampType`) và `timestamp_ms` (LongType).

3. **Watermarking:** Áp dụng watermark (ví dụ: 2 phút) cho cột `event_timestamp` để xử lý dữ liệu trễ cho các phép toán stateful.

4. **Luồng 1 - Nến 1 Phút Mới Nhất (cho thẻ "Latest Price/Volume"):**

   - Từ `watermarked_ohlcv_df`, thực hiện `groupBy("symbol")` và `agg()` để lấy `F.last()` của các trường `close_price` (đặt tên là `current_price`), `volume` (đặt tên là `current_volume`), `event_timestamp` (đặt tên là `latest_event_timestamp`), `open`, `high`, `low`, `timestamp_ms`.
   - Stream query này sử dụng `outputMode("complete")` và `trigger` thường xuyên (ví dụ: 15 giây).
   - Trong `foreachBatch`, dữ liệu được ghi/upsert vào Elasticsearch index `crypto_ohlcv_1m_latest` với `doc_id` là `symbol`.

5. **Luồng 2 - Thống Kê Cửa Sổ (cho các thẻ Avg/Min/Max):**

   - Từ `watermarked_ohlcv_df`, thực hiện `groupBy(col("symbol"), window(col("event_timestamp"), WINDOW_DURATION, SLIDE_DURATION))`.
   - Tính toán `avg("close_price")`, `min("close_price")`, `max("close_price")`, và `F.count("close_price")`.
   - Stream query này sử dụng `outputMode("update")` và `trigger` ít thường xuyên hơn (ví dụ: 1 phút).
   - Trong `foreachBatch`, dữ liệu được ghi/upsert vào Elasticsearch index `crypto_ohlcv_1m_stats` với `doc_id` là `symbol + "_stats_"+ window_end_timestamp`.

6. **Luồng 3 - Dữ liệu Chart 1 Phút Thô:**

   - Từ `parsed_ohlcv_df` (DataFrame đã parse từ Kafka, không cần watermark nếu chỉ append), chọn các cột cần thiết (`symbol`, `@timestamp` (là `event_timestamp`), `timestamp_ms`, `open`, `high`, `low`, `close`, `volume`).
   - Stream query này sử dụng `outputMode("append")` và `trigger` thường xuyên (ví dụ: 10 giây).
   - Trong `foreachBatch`, dữ liệu được ghi vào Elasticsearch index hàng ngày `crypto_ohlcv_1m_chartdata-YYYY-MM-DD` với `doc_id` là `symbol_1m_timestamp_ms`.

7. Tất cả các stream query đều sử dụng HDFS cho checkpoint.

5.3 Serving Layer và API (app.py)

Ứng dụng Flask cung cấp các API endpoint:

- Lấy danh sách symbol/timeframe cho dropdowns từ các index Elasticsearch tương ứng.
- /api/realtime_stats/<encoded_symbol>: Query đồng thời vào crypto_ohlcv_1m_latest (lấy theo id=symbol) và crypto_ohlcv_1m_stats (lấy thống kê cửa sổ mới nhất cho symbol) và trả về JSON.
- /api/chart_data_1m/<encoded_symbol>: Query vào các index crypto_ohlcv_1m_chartdata-* để lấy N điểm dữ liệu OHLCV 1 phút gần nhất (ví dụ 60 điểm), trả về dưới dạng mảng các cặp [timestamp_ms, close_price].
- /api/historical_data/<symbol_timeframe>: Query vào crypto_historical_data, hỗ trợ lọc theo tham số range (1m, 3m, all,...) dựa trên thời gian hiện tại.

Các symbol chứa '/' (ví dụ "BTC/USDT") được encode thành "BTC-USDT"trong URL API và decode ngược lại ở backend.

5.4 Tự Động Hóa bằng Cron

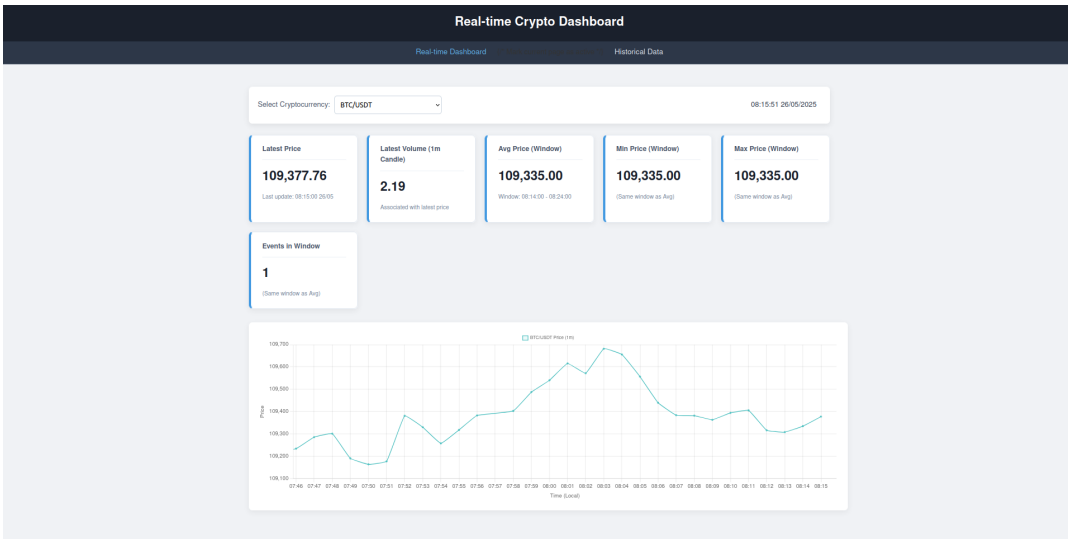
Các tác vụ cập nhật dữ liệu lịch sử hàng giờ được lập lịch bằng crontab:

- hourly_updater.py chạy vào phút thứ 5 của mỗi giờ.
- run_batch_processor.sh (kích hoạt batch_processor.py) chạy vào phút thứ 10 (hoặc 15, 20) của mỗi giờ.

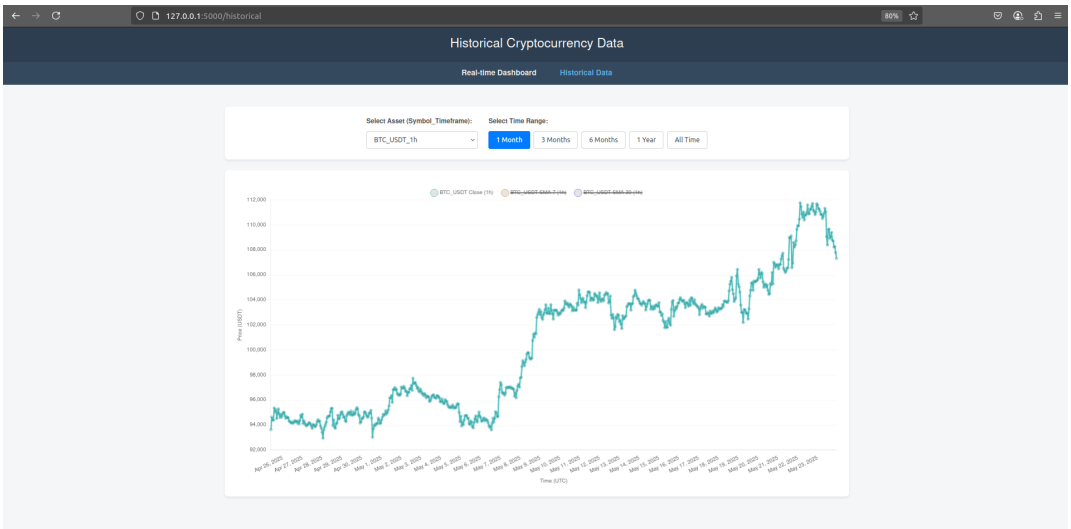
Log của các cron job được ghi vào thư mục logs/.

6 Giao Diện Người Dùng

Hệ thống cung cấp hai giao diện chính cho người dùng, được minh họa trong Hình 2.



(a) Giao diện Dashboard Real-time.



(b) Giao diện Dashboard Dữ liệu Lịch sử.

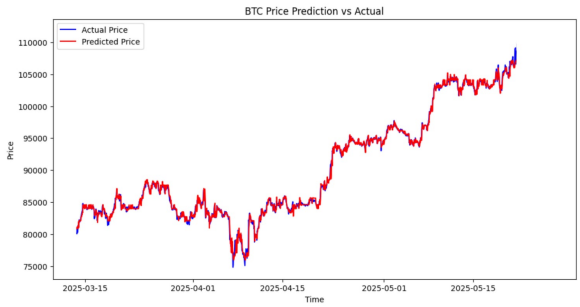
Hình 2: Các giao diện chính của hệ thống.

Mô tả các chức năng hiển thị trên mỗi trang:

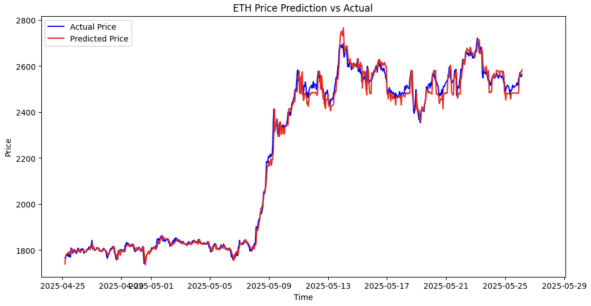
- **Dashboard Real-time (Hình 2a):** Hiển thị giá đóng cửa mới nhất (từ nến 1 phút), volume của nến 1 phút đó, các thống kê theo cửa sổ (trung bình, min, max giá đóng cửa của các nến 1 phút trong khoảng 10 phút qua, được cập nhật mỗi phút), số lượng nến 1 phút trong cửa sổ, và biểu đồ giá 1 phút cho khoảng 30-60 phút gần nhất. Người dùng có thể chọn các cặp tiền điện tử khác nhau để theo dõi.
- **Dashboard Dữ liệu Lịch sử (Hình 2b):** Hiển thị biểu đồ giá lịch sử (khung thời gian 1 giờ) với các đường trung bình động SMA7 và SMA30. Cho phép người dùng chọn cặp tiền điện tử/khung thời gian (ví dụ: BTC/USDT_1h) và các khoảng thời

gian hiển thị khác nhau (1 Tháng, 3 Tháng, ..., Tất cả). Biểu đồ hỗ trợ các tính năng zoom và pan để khám phá dữ liệu chi tiết.

- **Giao diện dự báo giá (Hình 3a và 3b):** Được thiết kế để hỗ trợ người dùng theo dõi xu hướng thị trường theo thời gian thực. Phần bên trái (Hình 3a) hiển thị đồ thị giá đóng cửa dự đoán của BTC trong khung thời gian 1 phút, kèm các chỉ báo kỹ thuật như đường trung bình động EMA5 và EMA20, cùng thông tin biến động phần trăm so với phiên trước. Phía bên phải (Hình 3b) tương tự nhưng dành cho ETH, cho phép so sánh song song giữa hai loại tài sản. Người dùng có thể chuyển đổi nhanh giữa các khung thời gian dự báo (1m, 5m, 15m) và tuỳ chỉnh ngưỡng cảnh báo biến động để nhận thông báo sớm khi giá chạm các mức quan trọng.



(a) Giao diện dự báo giá BTC theo thời gian thực.



(b) Giao diện dự báo giá ETH theo thời gian thực.

Hình 3: Các giao diện dự báo giá BTC và ETH theo thời gian thực.

Time (UTC)	Predicted Close Price
2025-05-27 00:00	2577.96
2025-05-27 01:00	2577.96
2025-05-27 02:00	2577.96
2025-05-27 03:00	2577.96
2025-05-27 04:00	2589.25
2025-05-27 05:00	2602.51
2025-05-27 06:00	2602.51
2025-05-27 07:00	2602.92
2025-05-27 08:00	2602.92
2025-05-27 09:00	2602.51
2025-05-27 10:00	2602.51
2025-05-27 11:00	2614.56
2025-05-27 12:00	2619.77
2025-05-27 13:00	2628.60
2025-05-27 14:00	2627.54
2025-05-27 15:00	2627.54
2025-05-27 16:00	2628.57
2025-05-27 17:00	2628.33
2025-05-27 18:00	2627.95
2025-05-27 19:00	2630.84
2025-05-27 20:00	2633.57
2025-05-27 21:00	2643.44
2025-05-27 22:00	2649.83
2025-05-27 23:00	2652.59

Hình 4: Dự đoán giá theo giờ của BTC ngày 27/05/2025 theo giờ UTC.

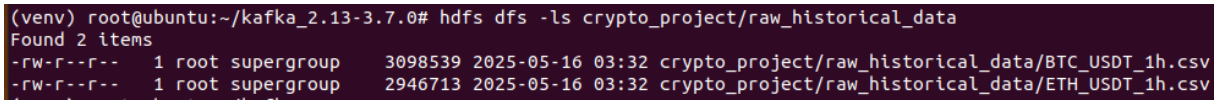
7 Hoạt Động Của Các Thành Phần Backend

7.1 Thu thập và Lưu trữ Dữ liệu Lịch sử (Batch Layer)

Luồng xử lý batch layer chịu trách nhiệm thu thập và xử lý dữ liệu lịch sử ban đầu, cũng như cập nhật dữ liệu mới định kỳ hàng giờ.

7.1.1 Thu thập dữ liệu lịch sử ban đầu

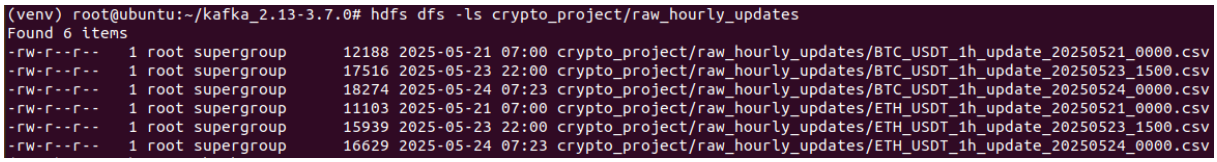
Script `historical_crawler.py` được thực thi một lần để thu thập dữ liệu OHLCV lịch sử từ sàn giao dịch Binance cho các cặp tiền BTC/USDT và ETH/USDT với khung thời gian 1 giờ, ví dụ từ ngày 01/01/2021 đến 15/05/2025. Dữ liệu thô được lưu dưới dạng file CSV trên HDFS tại thư mục `/user/root/crypto_project/raw_historical_data/`. Hình 5 minh họa kết quả kiểm tra thư mục này trên HDFS.



Hình 5: Kết quả liệt kê file trong thư mục `raw_historical_data` trên HDFS.

7.1.2 Cập nhật dữ liệu lịch sử hàng giờ

Script `hourly_updater.py` được lập lịch chạy mỗi giờ để thu thập dữ liệu OHLCV của giờ trước đó. Dữ liệu này được lưu vào thư mục `/user/root/crypto_project/raw_hourly_updates/` trên HDFS. Hình 6 cho thấy các file cập nhật được tạo ra vào các thời điểm khác nhau.



Hình 6: Kết quả liệt kê file trong thư mục `raw_hourly_updates` trên HDFS.

7.1.3 Xử lý Batch và Lưu trữ vào Serving Layer

Script Spark `batch_processor.py` đọc dữ liệu từ cả hai thư mục trên HDFS, tính toán các chỉ số SMA7, SMA30 và ghi/cập nhật (upsert) kết quả vào index `crypto_historical_data` trên Elasticsearch. Một document mẫu từ index này cho thấy dữ liệu đã được xử lý được thể hiện trong Hình 7.

```
huster@ubuntu:~$ curl -X GET "http://192.168.30.128:9200/crypto_historical_dat
a/_doc/BTC_USDT_1h_hist_1657040400?pretty"
{
  "_index" : "crypto_historical_data",
  "_id" : "BTC_USDT_1h_hist_1657040400",
  "_version" : 1,
  "_seq_no" : 13204,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "timestamp" : 1657040400,
    "symbol" : "BTC_USDT",
    "timeframe" : "1h",
    "open" : 19552.41,
    "high" : 19788.0,
    "low" : 19543.79,
    "close" : 19703.85,
    "volume" : 3647.64295,
    "sma_7" : 19505.464285714286,
    "sma_30" : 19883.189666666665,
    "doc_id" : "BTC_USDT_1h_hist_1657040400"
  }
}
```

Hình 7: Một document mẫu từ index `crypto_historical_data` trên Elasticsearch, hiển thị dữ liệu OHLCV 1 giờ và các chỉ số SMA.

7.2 Thu thập và Xử lý Dữ liệu Streaming (Speed Layer)

Luồng xử lý speed layer thu thập và xử lý dữ liệu OHLCV 1 phút theo thời gian thực từ sàn Binance.

7.2.1 Thu thập dữ liệu OHLCV 1 phút vào Kafka

Script `streaming_producer.py` chạy liên tục, lấy dữ liệu nến 1 phút mới nhất và gửi vào Kafka topic `crypto_ohlcv_1m`. Hình 8 cho thấy một message mẫu được lấy từ topic này bằng công cụ `kafka-console-consumer.sh`.

```
(venv) root@ubuntu:~/crypto_project# kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic crypto_ohlcv_1m --from-beginning --max-messages 1
{"timestamp": 1748237340000, "symbol": "BTC/USDT", "timeframe": "1m", "open": 1095
90.86, "high": 109608.7, "low": 109590.86, "close": 109608.69, "volume": 6.18994,
"datetime_str": "2025-05-26T05:29:00+00:00"}
```

Hình 8: Một message mẫu trong Kafka topic `crypto_ohlcv_1m`, chứa dữ liệu OHLCV 1 phút.

7.2.2 Xử lý Streaming và Lưu trữ vào Serving Layer

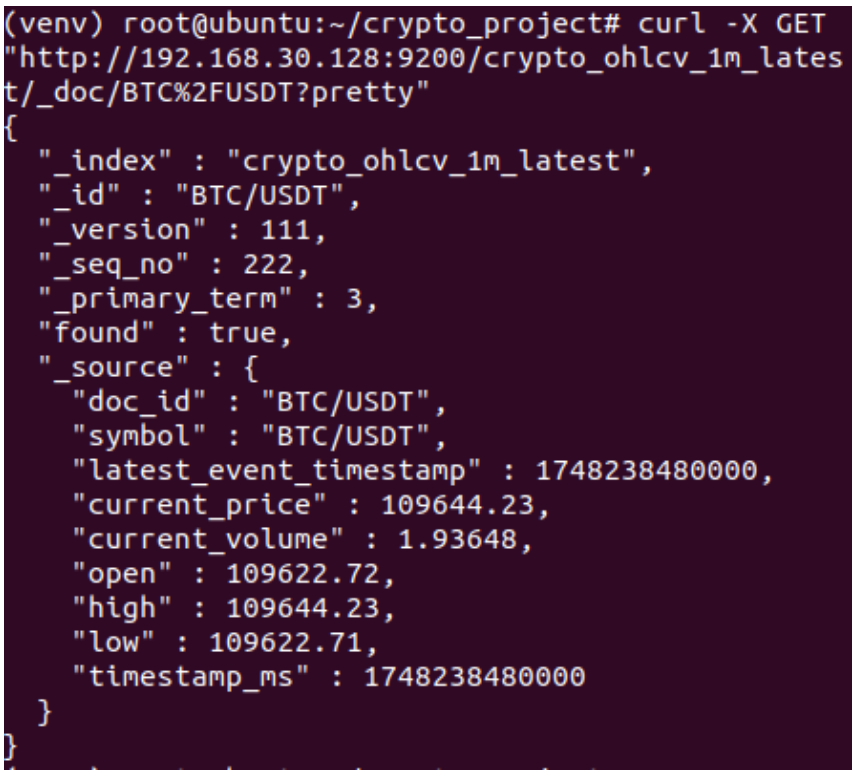
Script Spark Streaming `stream_processor.py` đọc dữ liệu từ topic `crypto_ohlcv_1m`. Nó thực hiện ba luồng xử lý chính song song, ghi kết quả vào các index Elasticsearch

riêng biệt:

1. **Lưu nến 1 phút mới nhất:** Thông tin của nến 1 phút mới nhất cho mỗi symbol được ghi/cập nhật vào index `crypto_ohlcv_1m_latest`. Hình 9 minh họa một document mẫu.

2. **Tính toán thống kê theo cửa sổ:** Các chỉ số thống kê (trung bình, min, max giá đóng cửa, số lượng nến) được tính toán từ giá đóng cửa của các nến 1 phút trong một cửa sổ trượt và được ghi/cập nhật vào index `crypto_ohlcv_1m_stats`. Hình 10 minh họa một document mẫu.

3. **Lưu dữ liệu thô cho biểu đồ 1 phút:** Toàn bộ thông tin của mỗi nến 1 phút được ghi vào các index theo ngày (ví dụ: `crypto_ohlcv_1m_chartdata-YYYY-MM-DD`). Hình 11 minh họa một document mẫu.



Hình 9: Document mẫu trong `crypto_ohlcv_1m_latest`.

Hình 10: Document mẫu trong `crypto_ohlcv_1m_stats`.

Hình 11: Document mẫu trong `crypto_ohlcv_1m_chartdata-YYYY-MM-DD`.

8 Đánh Giá Hệ Thống

8.1 Ưu điểm

- Hệ thống đã triển khai thành công kiến trúc Lambda, cho phép xử lý hiệu quả cả dữ liệu lịch sử quy mô lớn (Batch Layer) và dữ liệu OHLCV 1 phút theo thời gian thực (Speed Layer) cho thị trường tiền điện tử.
- Khả năng thu thập và cập nhật dữ liệu tự động: Dữ liệu lịch sử được bổ sung hàng giờ, và dữ liệu real-time được cập nhật liên tục, giúp duy trì tính thời sự của thông tin.
- Cung cấp giao diện người dùng trực quan với hai dashboard riêng biệt: một cho phân tích dữ liệu lịch sử với các chỉ báo kỹ thuật (SMA) và khả năng lọc theo nhiều khoảng thời gian; một cho theo dõi dữ liệu real-time với các thống kê cửa sổ và biểu đồ giá 1 phút cập nhật.
- Sử dụng các công nghệ Big Data phổ biến và mạnh mẽ như Apache Spark (Spark SQL, Spark Streaming), Apache Kafka, HDFS, và Elasticsearch, tạo nền tảng vững chắc cho các phát triển và mở rộng trong tương lai.
- Kiến trúc module hóa, tách biệt các lớp xử lý (thu thập, lưu trữ, xử lý batch, xử lý stream, phục vụ, hiển thị) giúp dễ dàng bảo trì, gỡ lỗi và nâng cấp từng thành phần.

8.2 Nhược điểm và Hạn chế

- Hiệu năng trên môi trường máy ảo (VMware) đơn lẻ còn hạn chế khi tất cả các thành phần (HDFS, Kafka, Spark, Elasticsearch, các script Python) chạy đồng thời, đặc biệt với các job Spark Streaming yêu cầu tài nguyên CPU và RAM đáng kể. Cần tối ưu thêm về trigger interval và tài nguyên cấp phát.
- Logic trích xuất symbol và timeframe từ tên file trong `batch_processor.py` dựa trên regex có thể cần cải thiện để linh hoạt và chính xác hơn với nhiều định dạng tên file đầu vào khác nhau, đặc biệt khi xử lý đồng thời file lịch sử gốc và file cập nhật.
- Việc quản lý Index Lifecycle Management (ILM) cho các index Elasticsearch chứa dữ liệu real-time chi tiết (như dữ liệu chart 1 phút được ghi theo ngày) cần được cấu hình và theo dõi cẩn thận để tối ưu dung lượng lưu trữ và tránh mất dữ liệu không mong muốn.

- Hệ thống hiện tại chủ yếu tập trung vào thu thập, xử lý và hiển thị dữ liệu thống kê cơ bản và giá cả, chưa triển khai các mô hình phân tích sâu hơn như dự đoán giá sử dụng Machine Learning/AI hoặc phát hiện bất thường.

- Chưa có cơ chế giám sát (monitoring) chi tiết và tự động cho sức khỏe của từng thành phần hệ thống (Kafka, Spark jobs, Elasticsearch) và các luồng dữ liệu.

8.3 Khó khăn gặp phải trong quá trình triển khai

Trong quá trình xây dựng và triển khai hệ thống, một số khó khăn và thách thức đã phát sinh, đòi hỏi sự tìm hiểu và gỡ lỗi đáng kể:

- **Tương thích phiên bản và quản lý dependencies:** Việc đảm bảo các phiên bản của Spark, Hadoop, Kafka, Elasticsearch và các thư viện connector liên quan (ví dụ: `elasticsearch-spark`, `spark-sql-kafka`) tương thích với nhau và với môi trường Java đã cài đặt là một thách thức ban đầu. Đã gặp một số trường hợp lỗi do xung đột phiên bản JAR hoặc thiếu class.

- **Cấu hình kết nối mạng giữa các thành phần:** Đặc biệt là kết nối từ Spark đến Elasticsearch. Lỗi `No data nodes with HTTP-enabled available` xuất hiện nhiều lần, ngay cả khi Elasticsearch có thể truy cập được từ bên ngoài. Vấn đề này yêu cầu kiểm tra kỹ lưỡng các cấu hình `es.nodes`, `es.port`, `es.nodes.wan.only` cả trong code Spark (`SparkSession.builder` hoặc `write.option`) và các tham số `-conf` của `spark-submit`, đồng thời đảm bảo cấu hình `network.host` và `http.host` của Elasticsearch cho phép kết nối từ địa chỉ IP của Spark driver/executor.

- **Quản lý tài nguyên trên môi trường máy ảo đơn lẻ:** Chạy đồng thời nhiều dịch vụ Big Data (HDFS NameNode, DataNode, ZooKeeper, Kafka Broker, Elasticsearch node) cùng các Spark job (batch và streaming) và các script Python producer trên một máy ảo VMware với tài nguyên CPU và RAM hạn chế thường xuyên dẫn đến tình trạng hệ thống chạy chậm, lag, hoặc thậm chí treo. Điều này đòi hỏi phải tối ưu hóa các tham số của Spark Streaming (như `trigger interval`), giảm số core sử dụng (`local[N]`), và theo dõi sát sao việc sử dụng tài nguyên.

- **Xử lý Timestamp và Múi giờ:** Đảm bảo tính nhất quán của dữ liệu timestamp (đơn vị milliseconds hay seconds, múi giờ UTC hay local time) qua các khâu từ thu thập dữ liệu (CCXT), truyền qua Kafka, xử lý trong Spark, lưu trữ trong Elasticsearch (dưới dạng `long` cho timestamp số hoặc `date` cho @timestamp), và cuối cùng là hiển thị trên frontend (sử dụng Moment.js) đòi hỏi sự cẩn thận và kiểm tra kỹ lưỡng để tránh sai lệch và hiển thị thời gian không chính xác.

- **Lập lịch Cron Job và Môi trường Thực thi:** Khi tự động hóa các tác vụ bằng cron, việc đảm bảo cron job chạy trong một môi trường có đầy đủ các biến cần thiết (PATH, JAVA_HOME, SPARK_HOME, các biến trong `.env`) và có đủ quyền để thực thi các script, đọc/ghi file là một điểm cần lưu ý để tránh các lỗi "âm thầm"khi cron chạy.

9 Kết Luận

Đồ án đã thành công trong việc xây dựng và triển khai một hệ thống thu thập, lưu trữ, xử lý và trực quan hóa dữ liệu giá tiền điện tử dựa trên kiến trúc Lambda. Hệ thống đã chứng minh khả năng xử lý cả dữ liệu lịch sử theo lô và dữ liệu OHLCV 1 phút theo thời gian thực, sử dụng các công nghệ Big Data cốt lõi như Apache Kafka, HDFS, Apache Spark và Elasticsearch. Giao diện người dùng cung cấp cái nhìn trực quan về dữ liệu lịch sử cũng như các thông tin cập nhật theo thời gian thực, bao gồm biểu đồ giá 1 phút.

10 Hướng Phát Triển

- **Tối ưu hiệu năng:** Triển khai trên môi trường cluster cho Spark, Kafka, Elasticsearch. Tối ưu hóa các Spark job, cấu hình ILM chi tiết hơn.
- **Mở rộng Phân tích Dữ liệu:**
  - Tích hợp các mô hình Machine Learning/Deep Learning (ví dụ: LSTM, Prophet) để dự đoán xu hướng giá.
  - Phân tích tâm lý thị trường từ các nguồn tin tức hoặc mạng xã hội.
  - Phát hiện các mẫu bất thường trong dữ liệu giá.
- **Cải thiện Giao diện Người dùng:** Thêm nhiều tùy chọn tùy chỉnh, cảnh báo, và các loại biểu đồ khác.
- **Tăng cường Độ tin cậy và Khả năng Giám sát:** Xây dựng cơ chế logging và monitoring chi tiết cho tất cả các thành phần.
- **Bảo mật:** Áp dụng các biện pháp bảo mật cho API và các dịch vụ dữ liệu.
- **Mở rộng Nguồn dữ liệu:** Tích hợp dữ liệu từ nhiều sàn giao dịch khác, dữ liệu on-chain.

Tài liệu

[1] Apache Spark Documentation. https://spark.apache.org/docs/latest/

[2] Apache Kafka Documentation. https://kafka.apache.org/documentation/

[3] Elasticsearch Guide. https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

[4] Apache Hadoop Documentation. https://hadoop.apache.org/docs/current/

[5] CCXT Library. https://github.com/ccxt/ccxt

[6] Flask Documentation. https://flask.palletsprojects.com/

[7] Chart.js Documentation. https://www.chartjs.org/docs/latest/

[8] Marz, N., & Warren, J. (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications.

A Nội dung File Cấu Hình và Script Phụ Trợ

A.1 File .env

```
# === .env ===
# --- Historical Crawler Configurations ---
CRYPTO_SYMBOLS="BTC/USDT,ETH/USDT"
CRYPTO_TIMEFRAME="1h"
CRYPTO_START_DATE="2020-01-01T00:00:00Z"
CRYPTO_END_DATE="2025-05-26T00:00:00Z"
CRYPTO_EXCHANGE="binance"

# --- Kafka Configuration ---
KAFKA_BROKER="localhost:9092"
KAFKA_OHLCV_1M_TOPIC="crypto_ohlcv_1m"
# KAFKA_TICKER_TOPIC="crypto_prices" # If using separate ticker stream

# --- Elasticsearch Configuration ---
ELASTICSEARCH_HOST="192.168.30.128"
ELASTICSEARCH_PORT="9200"
ES_NODES_WAN_ONLY="false"

# --- OHLCV 1-Minute Chart Producer Configuration ---
MINUTE_CHART_FETCH_INTERVAL=60
```

Listing 1: Nội dung file .env.example

A.2 Script run_batch_processor.sh

```
#!/bin/bash
PROJECT_DIR="/root/crypto_project"
echo "Starting batch_processor.py via cron at $(date)"
cd "$PROJECT_DIR" || exit
# export SPARK_HOME=/path/to/spark
spark-submit \
  --master local[*] \
  --jars ./elasticsearch-spark-30_2.12-8.14.0.jar \
  ./batch_processor.py
echo "Finished batch_processor.py via cron at $(date)"
```

Listing 2: Nội dung script run_batch_processor.sh

A.3 Cấu hình crontab

```
# 5 * * * * /path/to/venv/bin/python /path/to/project/hourly_updater.py
    >> /path/to/project/logs/hourly_updater.log 2>&1
# 10 * * * * /bin/bash /path/to/project/run_batch_processor.sh >> /path/
    to/project/logs/batch_processor_cron.log 2>&1
```

Listing 3: Nội dung crontab -l