

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ - DHQGHN

VIỆN TRÍ TUỆ NHÂN TẠO



## KỸ THUẬT VÀ CÔNG NGHỆ DỮ LIỆU LỚN

### XÂY DỰNG HỆ THỐNG GIÁM SÁT RỦI RO TÀI CHÍNH VÀ PHÁT HIỆN GIAN LẬN THỜI GIAN THỰC

Giảng viên hướng dẫn: Trần Hồng Việt  
Ngô Minh Hương

Sinh viên: Nguyễn Minh Quân 23020417  
Cao Minh Quang 23020411  
Nguyễn Năng Thịnh 23020439

HÀ NỘI, THÁNG MUÒI MỘT 2025

# Mục lục

<b>1 TỔNG QUAN</b>	<b>2</b>
1.1 Giới thiệu bài toán . . . . .	2
1.1.1 Bối cảnh và Tầm quan trọng . . . . .	2
1.1.2 Vấn đề và Thách thức trong môi trường Big Data . . . . .	2
1.2 Mục tiêu của dự án . . . . .	3
1.3 Phạm vi dự án . . . . .	3
1.3.1 Các nội dung thực hiện (In-scope) . . . . .	3
1.3.2 Các nội dung không thực hiện (Out-of-scope) . . . . .	3
1.4 Một số nghiên cứu liên quan . . . . .	4
1.4.1 Các nghiên cứu về phát hiện gian lận tài chính . . . . .	4
1.4.2 Các nghiên cứu về phân tích rủi ro thị trường . . . . .	4
1.4.3 Vai trò của kiến trúc Lambda trong các hệ thống tài chính . . . . .	4
<b>2 PHƯƠNG PHÁP</b>	<b>5</b>
2.1 Các công nghệ sử dụng . . . . .	5
2.1.1 Apache Kafka và Zookeeper . . . . .	5
2.1.2 Hadoop Distributed File System (HDFS) và YARN . . . . .	7
2.1.3 Apache Spark . . . . .	8
2.1.4 Redis . . . . .	10
2.1.5 Jupyter Notebook . . . . .	11
2.1.6 Nginx . . . . .	11
2.1.7 Docker và Docker Compose . . . . .	12
2.2 Thu thập và xử lý dữ liệu . . . . .	13
2.2.1 Nguồn dữ liệu . . . . .	13
2.2.2 Phân tích và Tiền xử lý dữ liệu (EDA) . . . . .	14
2.3 Mô hình hóa và Phân tích . . . . .	15
2.3.1 Mô hình Phát hiện gian lận . . . . .	15
2.3.2 Phân tích Rủi ro Thị trường theo Thời gian thực . . . . .	16
2.4 Triển khai . . . . .	17
2.4.1 Quy trình thực hiện và Luồng dữ liệu . . . . .	18
<b>3 KẾT QUẢ VÀ THẢO LUẬN</b>	<b>21</b>
3.1 Kết quả thực nghiệm . . . . .	21
3.1.1 Kết quả Mô hình Phát hiện gian lận . . . . .	21
3.1.2 Kết quả Phân tích Rủi ro Thị trường . . . . .	23

3.1.3	Giao diện Dashboard trực quan hóa . . . . .	24
3.2	Dánh giá hệ thống . . . . .	25
3.2.1	Dánh giá hiệu năng . . . . .	25
3.2.2	Dánh giá khả năng mở rộng . . . . .	25
3.3	Thảo luận . . . . .	26
3.3.1	Các ưu điểm đạt được . . . . .	26
3.3.2	Các nhược điểm và thách thức . . . . .	26
<b>4</b>	<b>KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN</b>	<b>27</b>
4.1	Kết luận . . . . .	27
4.2	Hướng phát triển . . . . .	27
<b>A</b>	<b>NHIỆM VỤ CỦA CÁC THÀNH VIÊN</b>	<b>30</b>



# TÓM TẮT

Trong bối cảnh cuộc cách mạng công nghiệp 4.0, ngành tài chính đang đổi mới với sự bùng nổ của dữ liệu giao dịch, dữ liệu thị trường và dữ liệu hành vi người dùng, tạo ra cả cơ hội lẫn thách thức to lớn. Việc phân tích khối dữ liệu khổng lồ (Big Data) này trong thời gian ngắn nhất để trích xuất thông tin giá trị đã trở thành yêu cầu sống còn đối với các tổ chức tài chính.

Hai trong số những bài toán quan trọng nhất là **phát hiện các hành vi gian lận giao dịch (fraud detection)** ngay khi chúng vừa xảy ra để ngăn chặn thất thoát tài chính, và **đo lường, phân tích các chỉ số rủi ro thị trường (market risk)** dựa trên dữ liệu lịch sử để đưa ra các quyết định đầu tư chiến lược.

Dự án tập trung xây dựng một **pipeline Big Data hoàn chỉnh** dựa trên kiến trúc **Lambda tinh chỉnh**, kết hợp xử lý dữ liệu thời gian thực và dữ liệu batch để giải quyết hai bài toán quan trọng trong lĩnh vực tài chính: **phát hiện gian lận giao dịch (fraud detection)** và **phân tích rủi ro thị trường (market risk)**.

Hệ thống sử dụng các công nghệ chủ đạo trong hệ sinh thái Big Data gồm **Apache Kafka** để thu thập và truyền dữ liệu thời gian thực, **Hadoop HDFS** để lưu trữ dữ liệu quy mô lớn, **Apache Spark** (Spark Streaming, Spark SQL, Spark MLlib) để xử lý và phân tích dữ liệu, cùng với **Redis** để truy xuất nhanh các kết quả tính toán.

Bằng cách kết hợp linh hoạt giữa xử lý thời gian thực và xử lý batch, pipeline mang lại khả năng mở rộng, độ tin cậy và hiệu suất cao, giúp các tổ chức tài chính nhận diện rủi ro kịp thời và đưa ra quyết định chính xác trong môi trường thị trường luôn biến động.

# Chương 1

## TỔNG QUAN

### 1.1 Giới thiệu bài toán

#### 1.1.1 Bối cảnh và Tầm quan trọng

Ngành tài chính hiện đại vận hành dựa trên dữ liệu. Mỗi giây trôi qua, hàng triệu giao dịch được thực hiện, hàng tỷ điểm dữ liệu thị trường được tạo ra. Khả năng khai thác dòng dữ liệu này quyết định lợi thế cạnh tranh của một tổ chức.

- **Phát hiện gian lận (Fraud Detection):** Các giao dịch gian lận ngày càng tinh vi, từ việc sử dụng thông tin thẻ bị đánh cắp đến các kỹ thuật rửa tiền phức tạp. Thiệt hại tài chính từ gian lận là vô cùng lớn. Việc phát hiện chúng sau khi đã xảy ra (post-transaction) là quá muộn. Thách thức đặt ra là phải xây dựng được mô hình có khả năng "chặn" hoặc gắn cờ các giao dịch đáng ngờ ngay lập tức (real-time), đòi hỏi một hệ thống xử lý có độ trễ cực thấp (low latency).
- **Phân tích rủi ro thị trường (Market Risk Analysis):** Sự biến động của thị trường tài chính (chứng khoán, tiền tệ, hàng hóa) đòi hỏi các tổ chức phải liên tục đánh giá lại rủi ro của danh mục đầu tư. Việc tính toán các chỉ số rủi ro phức tạp (như Value at Risk, Sharpe Ratio) trên một lượng lớn dữ liệu lịch sử (historical data) là một bài toán xử lý dữ liệu lớn theo lô (batch processing) điển hình, cần thông lượng (throughput) cao.

#### 1.1.2 Vấn đề và Thách thức trong môi trường Big Data

Việc giải quyết đồng thời hai bài toán trên đặt ra những thách thức kỹ thuật lớn, đặc trưng của Big Data (3V):

- **Volume (Khối lượng):** Phải lưu trữ và xử lý hàng Terabyte dữ liệu lịch sử cho phân tích rủi ro.
- **Velocity (Tốc độ):** Phải xử lý hàng ngàn giao dịch mỗi giây cho bài toán phát hiện gian lận.



- **Variety (Tính đa dạng):** Dữ liệu có thể đến từ nhiều nguồn khác nhau (giao dịch, log hệ thống, dữ liệu thị trường, tin tức).

Do đó, một hệ thống tài chính hiện đại không thể chỉ dựa vào cơ sở dữ liệu quan hệ truyền thống mà bắt buộc phải áp dụng các kiến trúc và công nghệ dữ liệu lớn phân tán.

## 1.2 Mục tiêu của dự án

Dự án này đặt ra các mục tiêu cụ thể như sau:

1. **Thiết kế và xây dựng một pipeline dữ liệu lớn hoàn chỉnh,** áp dụng kiến trúc Lambda Layer (hybrid architecture), có khả năng xử lý song song cả hai luồng dữ liệu thời gian thực và dữ liệu theo lô.
2. **Hiện thực luồng xử lý thời gian thực (Speed Layer)** cho bài toán phát hiện gian lận giao dịch. Luồng này phải đảm bảo độ trễ thấp, sử dụng Apache Kafka và Spark Streaming.
3. **Hiện thực luồng xử lý thời gian thực (Speed Layer) thứ hai** cho bài toán phân tích rủi ro thị trường, tính toán hơn 30 chỉ số và đẩy kết quả vào Redis để dashboard hiển thị tức thì.
4. **Hiện thực luồng xử lý theo lô (Batch Layer)** với nhiệm vụ duy nhất là huấn luyện lại (re-train) mô hình phát hiện gian lận định kỳ từ dữ liệu lịch sử trên HDFS.
5. **Trực quan hóa kết quả** phân tích từ cả hai luồng thông qua một dashboard tương tác, hiện đại, cung cấp cái nhìn tổng quan cho người ra quyết định.

## 1.3 Phạm vi dự án

### 1.3.1 Các nội dung thực hiện (In-scope)

- Triển khai toàn bộ hệ thống trên môi trường giả lập đa node sử dụng Docker và Docker Compose để đảm bảo tính nhất quán và dễ tái tạo.
- Sử dụng bộ dữ liệu giả lập PaySim cho bài toán gian lận (do tính nhạy cảm của dữ liệu thật) và phát lại (replay) dữ liệu này để giả lập luồng thời gian thực.
- Sử dụng thư viện vnstock để thu thập dữ liệu chứng khoán Việt Nam làm đầu vào cho bài toán phân tích rủi ro.
- Xây dựng các mô hình Machine Learning và tính toán tài chính cơ bản để chứng minh tính khả thi.

### 1.3.2 Các nội dung không thực hiện (Out-of-scope)

- Tối ưu hóa hiệu năng sâu (deep performance tuning) cho từng thành phần (ví dụ: tinh chỉnh JVM, cấu hình bộ nhớ của Spark).
- Triển khai trên cụm máy chủ vật lý (physical cluster) quy mô lớn.



- Xây dựng các mô hình dự đoán có độ chính xác ở mức sản phẩm thương mại (commercial-grade).
- Đảm bảo các vấn đề về an ninh, bảo mật (security, authentication) ở mức độ sản xuất.

## 1.4 Một số nghiên cứu liên quan

### 1.4.1 Các nghiên cứu về phát hiện gian lận tài chính

Phát hiện gian lận là một bài toán phân loại (classification) kinh điển. Tuy nhiên, nó có một đặc thù là dữ liệu cực kỳ mất cân bằng (highly imbalanced), tức là số lượng giao dịch hợp lệ chiếm áp đảo (ví dụ: 99.9%) so với giao dịch gian lận (0.1%). Các nghiên cứu trong lĩnh vực này tập trung vào hai hướng chính:

- **Kỹ thuật xử lý dữ liệu:** Sử dụng các phương pháp lấy mẫu như Over-sampling (ví dụ: SMOTE - Synthetic Minority Over-sampling Technique) để tăng số lượng mẫu của lớp thiểu số, hoặc Under-sampling (ví dụ: Random Undersampling) để giảm số lượng mẫu của lớp đa số.
- **Lựa chọn thuật toán:** Các thuật toán dạng "ensemble" như Random Forest và Gradient Boosting (XGBoost, LightGBM) thường cho kết quả rất tốt do khả năng xử lý dữ liệu nhiều chiều và tính phi tuyến cao. Các mô hình học sâu (Deep Learning) như mạng Nơ-ron (Neural Networks) hoặc Graph Neural Networks (GNN) cũng đang được nghiên cứu để phát hiện các mẫu gian lận tinh vi dựa trên mối quan hệ giữa các tài khoản.

### 1.4.2 Các nghiên cứu về phân tích rủi ro thị trường

Phân tích rủi ro thị trường tập trung vào việc lượng hóa các tổn thất tiềm ẩn.

- **Value at Risk (VaR):** Là chỉ số phổ biến nhất, được định nghĩa là "khoản lỗ tối đa có thể xảy ra trong một khoảng thời gian nhất định với một mức độ tin cậy cho trước". Các phương pháp tính VaR bao gồm phương pháp tham số (Parametric, giả định phân phối chuẩn), phương pháp lịch sử (Historical Simulation), và phương pháp Monte Carlo Simulation.
- **Sharpe Ratio:** Được phát triển bởi William F. Sharpe (người đoạt giải Nobel), chỉ số này đo lường lợi nhuận thu được (đã điều chỉnh theo rủi ro) trên một đơn vị rủi ro (đo bằng độ lệch chuẩn). Tỷ lệ này càng cao, hiệu suất đầu tư càng tốt.

### 1.4.3 Vai trò của kiến trúc Lambda trong các hệ thống tài chính

Kiến trúc Lambda, được giới thiệu bởi Nathan Marz, được thiết kế để xử lý khối lượng dữ liệu lớn bằng cách cung cấp một giải pháp cân bằng giữa độ trễ và tính chính xác. Nó chia hệ thống thành ba tầng: Batch Layer (ưu tiên tính chính xác, tính toán trên toàn bộ dữ liệu), Speed Layer (ưu tiên tốc độ, tính toán trên dữ liệu mới nhất), và Serving Layer (hợp nhất kết quả). Kiến trúc này đặc biệt phù hợp với ngành tài chính, nơi vừa cần các báo cáo chính xác tuyệt đối (batch) vừa cần các cảnh báo tức thời (real-time).

# Chương 2

## PHƯƠNG PHÁP

Chương này trình bày chi tiết về các công nghệ được sử dụng, phương pháp thu thập và xử lý dữ liệu, các mô hình phân tích và kiến trúc hệ thống tổng thể của dự án.

### 2.1 Các công nghệ sử dụng

Để xây dựng một hệ thống phức tạp với các yêu cầu cao về thông lượng và độ trễ, dự án đã sử dụng một tổ hợp các công nghệ mã nguồn mở hàng đầu trong hệ sinh thái Big Data.

#### 2.1.1 Apache Kafka và Zookeeper

##### Tổng quan về Apache Kafka

Apache Kafka là một nền tảng streaming sự kiện phân tán (distributed event streaming platform), ban đầu được phát triển tại LinkedIn và hiện là một dự án mã nguồn mở của Apache. Kafka được thiết kế để xử lý các luồng dữ liệu (data streams) theo thời gian thực với thông lượng cao, khả năng chịu lỗi tốt và có thể mở rộng theo chiều ngang.

Kiến trúc cốt lõi của Kafka bao gồm các thành phần sau:

- **Broker:** Một máy chủ Kafka (server) trong cụm (cluster). Nhiều broker làm việc cùng nhau để tạo thành một cụm Kafka, phân tán việc lưu trữ và xử lý dữ liệu.
- **Topic:** Một danh mục (category) hoặc tên luồng (feed name) mà các bản ghi (records) được đẩy vào. Dữ liệu trong Kafka được tổ chức theo các topic.
- **Partition:** Mỗi topic được chia thành nhiều "partition" (phân vùng). Partition là một bản ghi (log) bất biến, chỉ cho phép ghi tiếp vào cuối (append-only). Dữ liệu trong các partition được sắp xếp theo thứ tự và được gán một ID duy nhất gọi là "offset". Việc chia partition cho phép dữ liệu của một topic được phân tán trên nhiều broker, giúp tăng khả năng song song hóa.
- **Producer:** Các ứng dụng client gửi (ghi) dữ liệu vào các topic của Kafka.



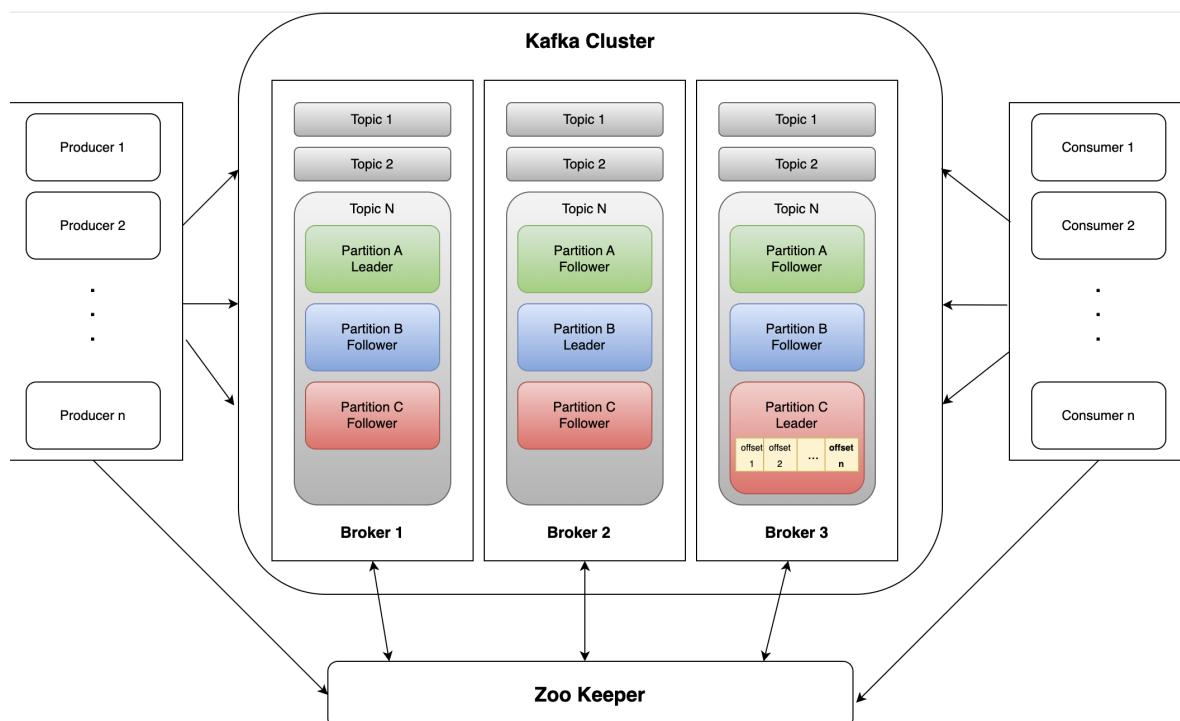
- Consumer:** Các ứng dụng client đọc (đăng ký) dữ liệu từ các topic. Kafka hỗ trợ mô hình "Consumer Group", nơi nhiều consumer có thể cùng đọc từ một topic, và mỗi partition chỉ được gán cho một consumer trong nhóm, cho phép xử lý song song.

## Tổng quan về Apache Zookeeper

Apache Zookeeper là một dịch vụ tập trung dùng để duy trì thông tin cấu hình, đặt tên, cung cấp khả năng đồng bộ hóa phân tán và cung cấp các dịch vụ nhóm. Trong bối cảnh của Kafka (các phiên bản trước 2.8), Zookeeper đóng vai trò cực kỳ quan trọng:

- Quản lý và điều phối các broker trong cụm.
- Theo dõi trạng thái của các node (broker nào đang sống/chết).
- Lưu trữ metadata của các topic, partition và offset.
- Thực hiện việc bầu chọn "controller" cho cụm Kafka.

\*(Lưu ý: Các phiên bản Kafka mới hơn đang dần loại bỏ Zookeeper bằng cách sử dụng cơ chế đồng thuận nội bộ gọi là KRaft, nhưng trong nhiều hệ thống sản xuất, Zookeeper vẫn được sử dụng rộng rãi).\*



Hình 2.1: Kiến trúc tổng quan của Apache Kafka.

## Vai trò trong dự án

Trong dự án này, Kafka đóng vai trò là "xương sống" (backbone) hay hệ thống thần kinh trung ương của toàn bộ pipeline dữ liệu:



- **Tiếp nhận dữ liệu đầu vào:** Kafka là điểm tiếp nhận duy nhất cho tất cả các luồng dữ liệu, bao gồm dữ liệu giao dịch (từ `realtime_producer.py`) và dữ liệu thị trường chứng khoán (từ `get_finance_data.py`).
- **Bộ đệm (Buffer) và Khớp nối (Decoupling):** Kafka đóng vai trò là một bộ đệm khổng lồ, cho phép các hệ thống producer (ghi dữ liệu) và consumer (đọc dữ liệu) hoạt động độc lập với tốc độ khác nhau. Producer có thể đẩy dữ liệu vào Kafka với tốc độ rất cao mà không cần quan tâm consumer có đang xử lý kịp hay không, và ngược lại. Điều này giúp hệ thống chống chịu được các đợt tải đột biến (load spike).
- **Phân phối dữ liệu:** Kafka cung cấp dữ liệu cho cả Speed Layer (Spark Streaming đọc topic `fraud_transactions`) và Batch Layer (Spark Batch có thể đọc từ topic dữ liệu thị trường hoặc dữ liệu được lưu trữ xuống HDFS).

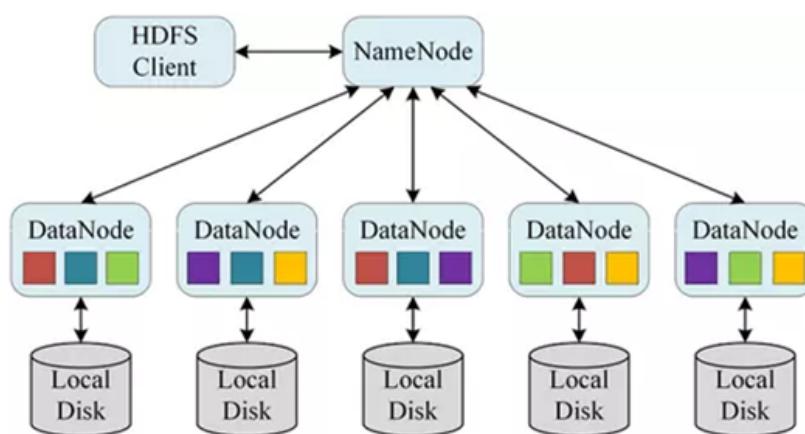
### 2.1.2 Hadoop Distributed File System (HDFS) và YARN

#### Tổng quan về HDFS

HDFS (Hadoop Distributed File System) là hệ thống tệp tin phân tán cốt lõi của hệ sinh thái Hadoop. Nó được thiết kế để lưu trữ các tệp tin cực lớn (hàng Terabyte đến Petabyte) trên các cụm máy chủ thông thường (commodity hardware).

Kiến trúc của HDFS tuân theo mô hình Master/Slave (chủ/tớ):

- **NameNode (Master):** Là máy chủ quản lý, lưu trữ toàn bộ "metadata" của hệ thống tệp. Metadata bao gồm cấu trúc cây thư mục, quyền truy cập tệp, và quan trọng nhất là bản đồ ánh xạ từ các tệp tin đến các khối (block) lưu trữ chúng trên các DataNode. Không có NameNode, hệ thống không biết dữ liệu nằm ở đâu.
- **DataNode (Slave):** Là các máy chủ lưu trữ dữ liệu thực tế. Dữ liệu được chia thành các khối (block) có kích thước lớn (thường là 128MB hoặc 256MB) và được lưu trữ trên các DataNode. Để đảm bảo tính chịu lỗi, mỗi block được sao chép (replicate) ra nhiều bản (thường là 3 bản) và lưu trên các DataNode khác nhau.



Hình 2.2: Kiến trúc tổng quan của HDFS.



## Tổng quan về YARN

YARN (Yet Another Resource Negotiator) là trình quản lý tài nguyên và lập lịch tác vụ của Hadoop (tách ra từ Hadoop 2.0). YARN chịu trách nhiệm cấp phát tài nguyên hệ thống (CPU, bộ nhớ) cho các ứng dụng chạy trên cụm Hadoop.

Kiến trúc của YARN cũng là Master/Slave:

- **ResourceManager (Master):** Chỉ có một ResourceManager duy nhất, quản lý toàn bộ tài nguyên của cụm. Nó nhận yêu cầu chạy ứng dụng từ client.
- **NodeManager (Slave):** Chạy trên tất cả các máy trong cụm (thường là các DataNode), chịu trách nhiệm quản lý tài nguyên trên máy đó (CPU, RAM) và báo cáo trạng thái cho ResourceManager.
- **ApplicationMaster:** Mỗi ứng dụng chạy trên YARN (ví dụ: một tác vụ Spark) sẽ có một ApplicationMaster riêng. Nó đàm phán (negotiate) với ResourceManager để xin cấp phát tài nguyên (dưới dạng các "Container") và sau đó liên lạc trực tiếp với các NodeManager để khởi chạy các Container đó.

## Vai trò trong dự án

- **HDFS (Data Lake):** HDFS được sử dụng làm "Hồ dữ liệu"(Data Lake) của hệ thống. Đây là nơi lưu trữ toàn bộ dữ liệu thô (raw data) và dữ liệu đã qua xử lý (processed data) của Batch Layer. Cụ thể, HDFS lưu trữ:
  - Dữ liệu lịch sử PaySim (/data/input/paysim\_realtime.csv).
  - Dữ liệu chứng khoán lịch sử (do spark\_consumer.py xử lý và lưu trữ).
  - Các mô hình Machine Learning đã được huấn luyện (do train.ipynb lưu lại), sẵn sàng để Speed Layer tải về sử dụng.
  - Kết quả tính toán rủi ro của Batch Layer (định dạng Parquet).
- **YARN (Quản lý tài nguyên):** Toàn bộ các ứng dụng Spark (cả streaming và batch) của dự án đều chạy trên YARN. YARN quản lý việc cấp phát CPU và bộ nhớ cho các tác vụ Spark, đảm bảo chúng chạy ổn định và không xung đột tài nguyên với các dịch vụ khác.

### 2.1.3 Apache Spark

#### Tổng quan về Apache Spark

Apache Spark là một framework xử lý dữ liệu phân tán, mã nguồn mở, nổi tiếng với tốc độ và tính linh hoạt. Khác với MapReduce của Hadoop (phụ thuộc nhiều vào việc đọc/ghi đĩa), Spark thực hiện phần lớn các phép tính trong bộ nhớ (in-memory processing), giúp tốc độ nhanh hơn đáng kể.

Hệ sinh thái Spark bao gồm nhiều thư viện tích hợp:

- **Spark Core:** Cung cấp API cơ bản (RDD - Resilient Distributed Dataset) và quản lý các tác vụ phân tán.

- **Spark SQL:** Cho phép truy vấn dữ liệu có cấu trúc bằng SQL hoặc thông qua API DataFrame/Dataset.
- **Spark Streaming:** Thư viện để xử lý các luồng dữ liệu gần thời gian thực (near real-time) bằng cách chia luồng dữ liệu thành các "micro-batch" nhỏ.
- **Spark MLlib:** Thư viện học máy (Machine Learning) phân tán, cung cấp các thuật toán phổ biến và các tiện ích (như feature engineering, pipeline).
- **GraphX (không dùng trong dự án):** Thư viện xử lý đồ thị phân tán.

Kiến trúc khi chạy, một ứng dụng Spark bao gồm:

- **Driver Program:** Chương trình chính (chứa hàm `main` hoặc session Jupyter) điều phối toàn bộ tác vụ.
- **Cluster Manager:** Trình quản lý tài nguyên (trong dự án này là YARN).
- **Executor:** Các tiến trình (process) chạy trên các node worker (do YARN khởi tạo trên các NodeManager), chịu trách nhiệm thực thi các tác vụ (task) tính toán và lưu trữ dữ liệu trong bộ nhớ/đĩa.



Hình 2.3: Kiến trúc hệ sinh thái Apache Spark.



## Vai trò trong dự án

Spark là **bộ não xử lý chính** của toàn bộ hệ thống, đảm nhiệm cả hai luồng xử lý:

- **Trong Speed Layer (realtime\_consumer.py):**
  - **Spark Streaming** được sử dụng để đọc dữ liệu từ topic `fraud_transactions` của Kafka theo dạng các micro-batch.
  - Với mỗi micro-batch, Spark áp dụng mô hình ML (đã được huấn luyện bởi Batch Layer) để dự đoán gian lận cho từng giao dịch.
  - Quá trình này diễn ra liên tục, đảm bảo cảnh báo được đưa ra chỉ vài giây sau khi giao dịch xảy ra.
- **Trong Batch Layer (preprocessing.ipynb, train.ipynb, spark\_consumer.py):**
  - **Spark SQL** và **DataFrame API** được sử dụng trong `preprocessing.ipynb` để thực hiện Phân tích Khám phá Dữ liệu (EDA), làm sạch, và kỹ thuật đặc trưng (feature engineering) trên bộ dữ liệu PaySim và dữ liệu chứng khoán.
  - **Spark MLLib** được sử dụng trong `train.ipynb` để huấn luyện mô hình phân loại (GBTClassifier) trên dữ liệu đã xử lý. Mô hình tốt nhất sau đó được lưu vào HDFS.
  - **Spark SQL** được sử dụng trong `spark_consumer.py` để thực hiện các phép tính toán rủi ro phức tạp (VaR, Sharpe Ratio) trên toàn bộ dữ liệu lịch sử chứng khoán lưu trữ trên HDFS.

### 2.1.4 Redis

#### Tổng quan về Redis

Redis (Remote Dictionary Server) là một hệ quản trị cơ sở dữ liệu key-value trong bộ nhớ (in-memory), mã nguồn mở, nổi tiếng với tốc độ truy xuất cực nhanh (độ trễ dưới mili-giây). Do lưu trữ dữ liệu chính trong RAM, Redis thường được sử dụng làm bộ đệm (cache), hàng đợi tin nhắn (message broker), hoặc cơ sở dữ liệu cho các ứng dụng đòi hỏi tốc độ cao. Redis hỗ trợ nhiều cấu trúc dữ liệu phức tạp (Strings, Hashes, Lists, Sets, Sorted Sets).

## Vai trò trong dự án

Trong kiến trúc của dự án, Redis đóng vai trò là **trái tim của Tầng Phục vụ (Serving Layer)**, hoạt động như một cơ sở dữ liệu trong bộ nhớ (in-memory database) tốc độ cực cao. Nhiệm vụ chính của Redis là:

- **Lưu trữ kết quả phân tích thời gian thực:** Toàn bộ hơn 30 chỉ số phân tích và rủi ro thị trường do `spark_consumer.py` tính toán được đẩy vào Redis ngay lập tức.
- **Cung cấp dữ liệu cho Dashboard:** Dashboard Streamlit (`dashboard.py`) kết nối trực tiếp và duy nhất với Redis để lấy dữ liệu hiển thị. Điều này đảm bảo dashboard



có thể cập nhật các biểu đồ và chỉ số với độ trễ dưới giây, mang lại trải nghiệm giám sát tức thì cho người dùng.

- **Quản lý danh sách cảnh báo:** Các cỗ phiếu có rủi ro cao hoặc các cảnh báo gian lận có thể được lưu vào các cấu trúc dữ liệu hiệu quả của Redis (như Sorted Sets hoặc Lists) để truy vấn và hiển thị một cách nhanh chóng.

Việc sử dụng Redis giúp tách biệt hoàn toàn tầng xử lý (Spark) và tầng hiển thị (Dashboard), đồng thời đảm bảo tốc độ truy xuất dữ liệu nhanh nhất có thể.

## 2.1.5 Jupyter Notebook

### Tổng quan về Jupyter Notebook

Jupyter Notebook là một ứng dụng web mã nguồn mở cho phép tạo và chia sẻ các tài liệu (gọi là "notebook") chứa mã nguồn (live code), phương trình, trực quan hóa và văn bản tường thuật (narrative text). Nó hỗ trợ nhiều ngôn ngữ lập trình, nhưng phổ biến nhất là Python. Jupyter trở thành công cụ tiêu chuẩn cho Khoa học Dữ liệu (Data Science) vì tính tương tác cao, cho phép người dùng chạy từng khối mã, xem kết quả ngay lập tức, và xây dựng quy trình phân tích một cách trực quan.

### Vai trò trong dự án

Jupyter Notebook đóng vai trò là **môi trường làm việc (IDE)** của Data Scientist trong dự án:

- **Phân tích Khám phá (EDA) và Tiền xử lý (preprocessing.ipynb):** Toàn bộ quá trình "làm quen" với dữ liệu (tính toán thống kê, vẽ biểu đồ, kiểm tra dữ liệu thiếu/ngoại lai) và chuẩn bị dữ liệu (feature engineering, chuẩn hóa) được thực hiện tương tác trong notebook.
- **Huấn luyện Mô hình (train.ipynb):** Quá trình thử nghiệm các thuật toán, xử lý mất cân bằng, tinh chỉnh tham số (hyperparameter tuning) và đánh giá mô hình được thực hiện trong notebook. Mỗi trường tương tác này cho phép so sánh kết quả của nhiều mô hình một cách nhanh chóng trước khi lưu lại mô hình tốt nhất.

Jupyter kết nối với cụm Spark (qua YARN) để thực thi các lệnh phân tích trên toàn bộ dữ liệu lớn chứ không phải trên một máy cục bộ.

## 2.1.6 Nginx

### Tổng quan về Nginx

Nginx (phát âm là "engine-X") là một máy chủ web (web server) hiệu năng cao, mã nguồn mở. Ban đầu nó được tạo ra để giải quyết vấn đề C10k (xử lý 10.000 kết nối đồng thời). Ngoài vai trò là một web server, Nginx còn được sử dụng cực kỳ phổ biến làm:

- **Reverse Proxy:** đứng trước các máy chủ ứng dụng (application servers) để nhận yêu cầu từ client, sau đó chuyển tiếp (proxy) yêu cầu đó đến máy chủ ứng dụng phù hợp.



- **Load Balancer (Bộ cân bằng tải):** Phân phối lưu lượng truy cập đến nhiều máy chủ ứng dụng để tăng khả năng chịu tải và độ tin cậy.
- **HTTP Cache:** Lưu cache các nội dung tĩnh để giảm tải cho máy chủ.

### Vai trò trong dự án

Trong dự án này, Nginx đóng vai trò là **cổng vào (Gateway) duy nhất** của toàn bộ hệ thống. Người dùng không truy cập trực tiếp vào từng dịch vụ (HDFS, YARN, Spark, Jupyter) mà sẽ thông qua Nginx:

- Cung cấp một địa chỉ truy cập duy nhất (ví dụ: <http://localhost>).
- Điều hướng (route) các yêu cầu của người dùng đến các giao diện web tương ứng.  
Ví dụ:
  - <http://localhost/jupyter> → Giao diện Jupyter Notebook.
  - <http://localhost/spark> → Giao diện Spark Master/History Server.
  - <http://localhost/hdfs> → Giao diện HDFS NameNode.
  - <http://localhost/yarn> → Giao diện YARN ResourceManager.
  - <http://localhost/dashboard> → Giao diện dashboard (dashboard.py).
- Đơn giản hóa việc quản lý truy cập và có thể mở rộng để thêm các lớp bảo mật (như xác thực) sau này.

### 2.1.7 Docker và Docker Compose

#### Tổng quan về Docker

Docker là một nền tảng mã nguồn mở cho phép "container hóa" (containerization) các ứng dụng. Một container là một môi trường thực thi nhẹ, độc lập, bao gồm mọi thứ cần thiết để chạy một ứng dụng: mã nguồn, thư viện, biến môi trường, và các tệp cấu hình. Container khác với máy ảo (VM) ở chỗ chúng chia sẻ nhân (kernel) của hệ điều hành máy chủ, giúp chúng khởi động nhanh hơn và tốn ít tài nguyên hơn.

#### Tổng quan về Docker Compose

Khi một ứng dụng phức tạp bao gồm nhiều dịch vụ (container) cần nối chuyện với nhau (như dự án này có Kafka, Zookeeper, Spark, HDFS, Nginx, v.v.), việc quản lý chúng bằng tay trở nên khó khăn. Docker Compose là một công cụ cho phép định nghĩa và chạy các ứng dụng Docker đa container. Người dùng chỉ cần định nghĩa toàn bộ các dịch vụ, mạng (network) và ổ đĩa (volume) trong một tệp `docker-compose.yml` duy nhất, sau đó dùng một lệnh (`docker-compose up`) để khởi chạy toàn bộ hệ thống.

### Vai trò trong dự án

Docker và Docker Compose là **nền tảng triển khai** của dự án:



- **Đóng gói:** Toàn bộ các công nghệ phức tạp (Hadoop, Spark, Kafka, v.v.) được đóng gói vào các container riêng biệt.
- **Môi trường nhất quán:** Đảm bảo rằng hệ thống chạy giống hệt nhau trên máy của mọi lập trình viên cũng như trên máy chủ triển khai, giải quyết triệt để vấn đề "nó chạy trên máy tôi!".
- **Triển khai đơn giản:** Thay vì phải cài đặt và cấu hình hàng chục dịch vụ phức tạp bằng tay, người dùng chỉ cần chạy một lệnh docker-compose up để khởi động toàn bộ cụm dữ liệu lớn giả lập.

## 2.2 Thu thập và xử lý dữ liệu

### 2.2.1 Nguồn dữ liệu

Dự án sử dụng hai bộ dữ liệu riêng biệt cho hai bài toán nghiệp vụ:

#### Dữ liệu Phát hiện gian lận (PaySim)

- **Nguồn:** Đây là bộ dữ liệu giả lập phổ biến cho các bài toán phát hiện gian lận, được tạo ra bằng trình giả lập PaySim.
- **Phương thức:** Dữ liệu được lưu trữ sẵn trên HDFS (/data/input/paysim\_realtime.csv). Một kịch bản producer (realtime\_producer.py) đọc tệp CSV này và "phát lại" (replay) từng dòng dữ liệu vào topic fraud\_transactions của Kafka, giả lập một luồng giao dịch thời gian thực.
- **Cấu trúc (dự kiến):**
  - step: Đơn vị thời gian (ví dụ: 1 giờ).
  - type: Loại giao dịch (CASH\_IN, CASH\_OUT, DEBIT, PAYMENT, TRANSFER).
  - amount: Số tiền giao dịch.
  - nameOrig, oldbalanceOrig, newbalanceOrig: Thông tin tài khoản gửi.
  - nameDest, oldbalanceDest, newbalanceDest: Thông tin tài khoản nhận.
  - isFraud: Biến mục tiêu (1 = gian lận, 0 = hợp lệ).

#### Dữ liệu Phân tích Rủi ro (Chứng khoán Việt Nam)

- **Nguồn:** Dữ liệu được thu thập trực tiếp từ thị trường chứng khoán Việt Nam thông qua thư viện Python vnstock.
- **Phương thức:** Một kịch bản producer (get\_finance\_data.py) chạy định kỳ (ví dụ: mỗi 15 giây). Trong mỗi lần chạy, nó gọi API của vnstock để lấy dữ liệu bảng giá (price board) của một danh sách các mã cổ phiếu blue-chip được định nghĩa trước (ví dụ: VCB, VHM, HPG, FPT).



- **Lưu trữ:** Dữ liệu này sau khi thu thập có thể được đẩy vào một topic Kafka khác (ví dụ: `stock_market_data`) và sau đó được một consumer (ví dụ: `spark_consumer.py`) đọc, xử lý và lưu trữ lâu dài trên HDFS (dưới dạng Parquet) cho Batch Layer.

## 2.2.2 Phân tích và Tiền xử lý dữ liệu (EDA)

Đây là bước quan trọng được thực hiện trong notebook `preprocessing.ipynb` trước khi huấn luyện mô hình, sử dụng Spark SQL và DataFrame API.

### Phân tích khám phá (EDA)

- **Thống kê mô tả:** Sử dụng hàm `describe()` của Spark DataFrame để tính toán các giá trị thống kê cơ bản (số lượng, trung bình, độ lệch chuẩn, min, max) cho các cột số.
- **Trực quan hóa (Giả định):** Trực quan hóa phân phối của biến `amount`, mối tương quan giữa các biến, và đặc biệt là kiểm tra tỷ lệ mất cân bằng nghiêm trọng của biến `isFraud`.

### Làm sạch dữ liệu

- **Xử lý giá trị thiếu (Missing Values):** Kiểm tra sự tồn tại của các giá trị `null` hoặc `NaN` và áp dụng các chiến lược xử lý (ví dụ: loại bỏ dòng, điền giá trị trung bình/trung vị).
- **Xử lý ngoại lai (Outliers):** Phát hiện các giá trị bất thường (ví dụ: số tiền giao dịch âm) và xử lý chúng.

### Kỹ thuật đặc trưng (Feature Engineering)

Đây là bước tạo ra các đặc trưng mới (features) có ý nghĩa từ dữ liệu thô để giúp mô hình học tốt hơn.

- **Cho dữ liệu PaySim (ví dụ):**
  - Tạo đặc trưng `errorBalanceOrig = newbalanceOrig + amount - oldbalanceOrg`. Một giá trị `errorBalanceOrig` khác 0 có thể là một dấu hiệu đáng ngờ.
  - Chuyển đổi các đặc trưng dạng `category` (như `type`) thành dạng số mà mô hình có thể hiểu được (ví dụ: sử dụng `StringIndexer` và `OneHotEncoder` của Spark MLlib).
  - Từ `step`, có thể tạo ra các đặc trưng thời gian như `hour_of_day`.
- **Cho dữ liệu chứng khoán (ví dụ):**
  - Tính toán tỷ suất sinh lợi hàng ngày (daily return).
  - Tính toán các chỉ báo kỹ thuật (technical indicators) như Đường trung bình động (Moving Average - MA), Chỉ số sức mạnh tương đối (RSI).



## Chuẩn hóa dữ liệu

Dựa các đặc trưng số về cùng một thang đo (scale) để các thuật toán (như Logistic Regression) hoạt động hiệu quả.

- **VectorAssembler:** Sử dụng `VectorAssembler` của Spark MLlib để gộp tất cả các cột đặc trưng thành một cột vector duy nhất.
- **StandardScaler:** Sử dụng `StandardScaler` để chuẩn hóa cột vector này (trừ các giá trị về 0 và chia cho độ lệch chuẩn).

Dữ liệu sau khi đã được xử lý và làm giàu sẽ được lưu lại vào HDFS dưới định dạng Parquet, sẵn sàng cho quá trình huấn luyện mô hình.

## 2.3 Mô hình hóa và Phân tích

Dự án triển khai hai loại mô hình riêng biệt.

### 2.3.1 Mô hình Phát hiện gian lận

Quá trình này được thực hiện trong notebook `train.ipynb`.

#### Xử lý mất cân bằng

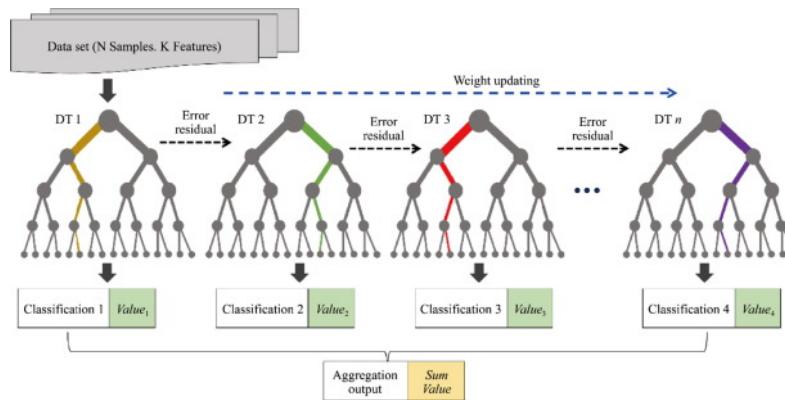
Như đã đề cập, dữ liệu gian lận là cực kỳ mất cân bằng. Nếu huấn luyện mô hình trực tiếp, nó sẽ có xu hướng dự đoán "không gian lận" cho mọi giao dịch và vẫn đạt độ chính xác (accuracy) cao (ví dụ: 99.9%), nhưng lại hoàn toàn vô dụng.

- **Phương pháp áp dụng:** Sử dụng kỹ thuật \*\*Random Undersampling\*\* (giảm số lượng mẫu của lớp đa số) hoặc \*\*SMOTE\*\* (tạo ra các mẫu tổng hợp cho lớp thiểu số) là bắt buộc để mô hình không bị "thiên vị".

#### Lựa chọn mô hình: GBTClassifier

Các mô hình phân loại phổ biến trong Spark MLlib như `Logistic Regression`, `Decision Tree Classifier`, và `GBTClassifier` đã được xem xét.

- **Mô hình được chọn:** `GBTClassifier (Gradient-Boosted Tree)` là một lựa chọn mạnh mẽ. Đây là một mô hình "ensemble" (tổng hợp) học tăng cường (boosting).
- **Cơ chế hoạt động:** Mô hình xây dựng tuần tự các cây quyết định yếu (weak decision trees), nơi mỗi cây mới được huấn luyện để cố gắng sửa chữa các lỗi dự đoán mà các cây trước đó đã mắc phải.
- **Lý do lựa chọn:** GBTs thường mang lại độ chính xác rất cao, xử lý tốt các mối quan hệ phi tuyến và tương tác phức tạp giữa các đặc trưng, khiến chúng rất phù hợp cho các bài toán phát hiện gian lận tinh vi.



Hình 2.4: Cơ chế hoạt động của thuật toán Gradient-Boosted Trees (GBT).

### Huấn luyện và Đánh giá

- **Pipeline:** Toàn bộ quá trình tiền xử lý và mô hình hóa được gói gọn trong một Pipeline của Spark MLLib.
- **Đánh giá:** Do đây là bài toán mêt cân bằng, mô hình được đánh giá trên tập kiểm thử (test set) bằng các độ đo:
  - **Precision (Độ chính xác):** Tỷ lệ các dự đoán "gian lận" là chính xác.
  - **Recall (Độ phủ):** Tỷ lệ các ca "gian lận" thực tế đã bị phát hiện.
  - **F1-Score:** Trung bình điều hòa của Precision và Recall.
  - **AreaUnderROC (AUC-ROC):** Thước đo tổng quát về khả năng phân biệt giữa hai lớp của mô hình.

Mô hình có kết quả tốt nhất sẽ được lưu lại vào HDFS để `realtime_consumer.py` sử dụng.

### 2.3.2 Phân tích Rủi ro Thị trường theo Thời gian thực

Quá trình này được thực hiện trong kịch bản `spark_consumer.py` (hoạt động như một ứng dụng Spark Streaming), chuyển đổi dữ liệu giá cổ phiếu thô thành một bộ gồm hơn 30 chỉ số phân tích và rủi ro toàn diện.

#### Các nhóm chỉ số chính

- **Chỉ số Thị trường (Market Metrics):** Đo lường "sức khỏe" và động lực của cổ phiếu trong phiên, ví dụ: Volatility Intraday, Price Momentum, Liquidity Score.
- **Chỉ số Rủi ro (Risk Metrics):** Lượng hóa các loại rủi ro cụ thể, ví dụ: Liquidity Risk Score, Price Risk Position, và đặc biệt là **Composite Risk Score** - điểm rủi ro tổng hợp.
- **Chỉ số Tiềm năng (VaR-based Metrics):** Dánh giá tiềm năng lợi nhuận so với rủi ro, ví dụ: Downside Risk %, Upside Potential %, và **Risk/Reward Ratio**.

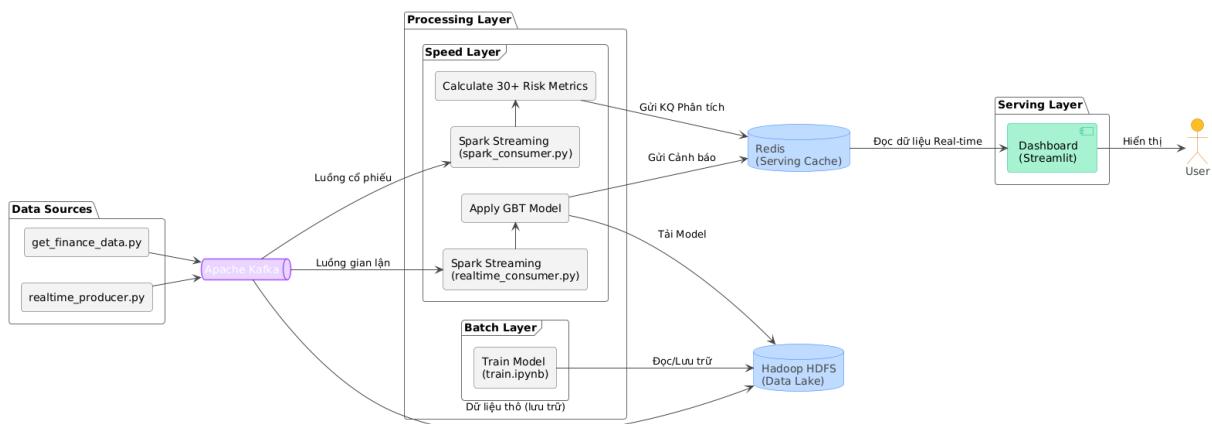
## Phương pháp tính toán và Lưu trữ

- `spark_consumer.py` đọc dữ liệu từ Kafka, áp dụng hàng loạt các phép biến đổi và tính toán bằng Spark SQL để tạo ra các chỉ số trên.
- Kết quả phân tích chi tiết của từng cổ phiếu được đẩy vào **Redis** theo thời gian thực, sẵn sàng để dashboard truy vấn và hiển thị ngay lập tức. Các cổ phiếu có rủi ro cao (Composite Risk Score > 2.0) được đưa vào một danh sách riêng để theo dõi đặc biệt.

## 2.4 Triển khai

### Kiến trúc hệ thống

Kiến trúc của dự án được thiết kế theo mô hình Lambda lai, kết hợp sức mạnh của cả hai tầng xử lý theo lô và thời gian thực để đáp ứng đồng thời hai bài toán nghiệp vụ khác nhau.



Hình 2.5: Kiến trúc hệ thống tổng thể của dự án.

#### • Tầng Batch (Batch Layer):

- **Mục tiêu:** Xử lý các tác vụ tính toán nặng, đòi hỏi độ chính xác trên toàn bộ dữ liệu lịch sử.
- **Công nghệ:** HDFS, Spark Batch (Spark MLlib).
- **Nhiệm vụ chính:**

- \* Lưu trữ toàn bộ dữ liệu giao dịch lịch sử trên HDFS.
- \* Định kỳ (ví dụ: hàng đêm) chạy tác vụ Spark Batch (`train.ipynb`) để **huấn luyện lại (re-train)** mô hình phát hiện gian lận trên tập dữ liệu đầy đủ nhất.
- \* Lưu mô hình đã được huấn luyện và tối ưu vào HDFS để Tầng Tốc độ sử dụng.

#### • Tầng Tốc độ (Speed Layer):



- **Mục tiêu:** Xử lý dữ liệu ngay khi nó phát sinh và cung cấp kết quả với độ trễ thấp nhất. Tầng này xử lý cả hai luồng nghiệp vụ.
- **Công nghệ:** Kafka, Spark Streaming, Redis.
- **Nhiệm vụ:**
  - \* **Luồng phát hiện gian lận:**
    1. Nhận dữ liệu giao dịch từ topic Kafka `fraud_transactions`.
    2. Tải mô hình ML mới nhất từ HDFS (do Tầng Batch cung cấp).
    3. Áp dụng mô hình để dự đoán gian lận cho từng giao dịch (`realtime_consumer.py`).
    4. Đẩy kết quả dự đoán và cảnh báo vào Redis.
  - \* **Luồng phân tích rủi ro thị trường:**
    1. Nhận dữ liệu giá cổ phiếu từ topic Kafka `vietnam_stocks`.
    2. Tính toán hơn 30 chỉ số phân tích và rủi ro theo thời gian thực (`spark_consumer.py`).
    3. Đẩy toàn bộ kết quả phân tích rủi ro vào Redis.

- **Tầng Phục vụ (Serving Layer):**

- **Mục tiêu:** Hợp nhất và cung cấp dữ liệu từ các tầng xử lý cho người dùng cuối một cách hiệu quả.
- **Công nghệ:** Redis, Dashboard (Streamlit - `dashboard.py`), Nginx.
- **Nhiệm vụ:** Dashboard (`dashboard.py`) kết nối trực tiếp và duy nhất với **Redis** để:
  - \* Hiển thị các cảnh báo gian lận mới nhất.
  - \* Trực quan hóa toàn bộ các chỉ số phân tích rủi ro thị trường được cập nhật liên tục (real-time), bao gồm các biểu đồ tương tác, bảng phân tích chi tiết, và cảnh báo rủi ro cao.

### 2.4.1 Quy trình thực hiện và Luồng dữ liệu

Dưới đây là mô tả hai luồng nghiệp vụ chính chạy song song trên kiến trúc này.

#### Luồng Phát hiện gian lận (Speed Layer)

Luồng **Speed Layer** chịu trách nhiệm xử lý dữ liệu giao dịch gần như thời gian thực để phát hiện các giao dịch可疑 và cảnh báo kịp thời. Quy trình chi tiết được mô tả như sau:

##### 1. Producer (`realtime_producer.py`):

- Đọc dữ liệu giao dịch từ các tệp CSV trên HDFS. Mỗi dòng dữ liệu tương ứng với một giao dịch, bao gồm các thông tin như: thời gian, loại giao dịch, số tiền, số dư cũ và mới của tài khoản, v.v.



- Mỗi giao dịch được định dạng thành bản ghi JSON và gửi tới topic Kafka **fraud\_transactions**.
- Producer hoạt động theo cơ chế giả lập dữ liệu streaming: từng dòng CSV được gửi tuần tự để mô phỏng luồng giao dịch liên tục.

## 2. Kafka:

- Kafka đóng vai trò là hệ thống message queue trung gian, đảm bảo giao dịch từ producer được truyền tới consumer một cách đáng tin cậy.
- Topic **fraud\_transactions** lưu trữ các bản ghi giao dịch, hỗ trợ nhiều consumer đọc song song và theo dõi luồng dữ liệu thời gian thực.

## 3. Consumer (realtime\_consumer.py - Spark Structured Streaming):

- Spark Streaming định kỳ đọc các micro-batch từ topic Kafka. Mỗi micro-batch chứa một số lượng giao dịch nhất định, giúp xử lý dữ liệu theo từng lô nhỏ để đạt hiệu suất và độ trễ thấp.
- Khi khởi động, consumer tải mô hình **GBTClassifier** đã được huấn luyện từ Batch Layer trên tập dữ liệu lịch sử. Việc tải mô hình này chỉ thực hiện một lần và có thể lưu trữ trong bộ nhớ để áp dụng cho tất cả các micro-batch.
- Trước khi dự đoán, các giao dịch trong micro-batch được tiền xử lý tương tự như quá trình huấn luyện: chuẩn hóa số liệu, mã hóa các biến phân loại, và loại bỏ các giá trị ngoại lai nếu cần.
- Mỗi giao dịch được mô hình GBT đánh giá, trả về nhãn dự đoán: 0 nếu hợp lệ, 1 nếu nghi ngờ gian lận.

## 4. Xử lý kết quả và cảnh báo:

- Nếu một giao dịch bị dự đoán là gian lận (`prediction == 1`), hệ thống sẽ ghi một bản ghi cảnh báo vào log chuyên dụng.
- Dashboard theo dõi cảnh báo này có thể trực tiếp đọc log hoặc tham chiếu đến bộ đếm trong Redis, hiển thị số lượng giao dịch nghi ngờ theo thời gian thực.
- Có thể cấu hình cảnh báo nâng cao như gửi email, SMS hoặc webhook tới hệ thống giám sát khi phát hiện giao dịch nguy hiểm.

Như vậy, Speed Layer đảm bảo rằng mọi giao dịch mới đều được đánh giá nhanh chóng, giúp phát hiện gian lận gần như ngay lập tức, bổ sung cho Batch Layer vốn xử lý dữ liệu lịch sử và huấn luyện mô hình.

## Luồng Phân tích Rủi ro (Speed Layer - Real-time)

Luồng **Luồng Phân tích Rủi ro** thực hiện xử lý dữ liệu tài chính theo cơ chế microbatch nhằm tính toán các chỉ số phân tích và rủi ro chuyên sâu, phục vụ cho phân tích dài hạn và hiển thị trên dashboard. Quy trình được mô tả như sau:

### 1. Producer (get\_finance\_data.py):

- Định kỳ mỗi 15 giây thu thập dữ liệu bảng giá từ API vnstock.



- Chuẩn hoá dữ liệu về các trường: mã cổ phiếu, giá, khối lượng, thời gian.
- Gửi từng bản ghi vào topic Kafka `vietnam_stocks`, đóng vai trò làm hàng đợi thông lượng cao cung cấp dữ liệu đầu vào cho hệ thống xử lý.

## 2. Xử lý Dữ liệu Streaming (spark\_consumer.py):

- Ứng dụng sử dụng **Apache Spark Structured Streaming** để đọc liên tục các micro-batch từ Kafka.
- Mỗi micro-batch được phân tích và biến đổi theo schema thống nhất, đảm bảo dữ liệu luôn nhất quán khi đưa vào các bước tính toán tiếp theo.

## 3. Tính toán Chỉ số Phân tích và Rủi ro:

- Hệ thống tính toán hơn **30 chỉ số** thuộc ba nhóm chính:
  - Nhóm Thị trường: Volatility Intraday, Price Momentum, Liquidity Score,...
  - Nhóm Rủi ro: Liquidity Risk Score, Price Risk Position, và đặc biệt là **Composite Risk Score**.
  - Nhóm Tiềm năng: Downside Risk %, Upside Potential %, Risk/Reward Ratio.
- Tất cả chỉ số được tính ngay khi dữ liệu mới xuất hiện, cho phép hệ thống phản ánh kịp thời biến động thị trường.

## 4. Lưu trữ vào Tầng Phục vụ (Serving Layer — Redis):

- Kết quả phân tích chi tiết của từng cổ phiếu được lưu vào **Redis** dưới dạng key-value giúp truy xuất nhanh.
- Các cổ phiếu có mức rủi ro cao (ví dụ: **Composite Risk Score > 2.0**) được đưa vào một Sorted Set để hỗ trợ truy vấn, sắp xếp và cảnh báo theo thời gian thực.

## 5. Hiển thị (Dashboard — dashboard.py):

- Dashboard xây dựng bằng Streamlit kết nối trực tiếp với Redis để nhận dữ liệu mới nhất.
- Người dùng có thể xem biểu đồ tương tác, danh sách cảnh báo, và phân tích rủi ro được cập nhật liên tục theo từng giây.

# Chương 3

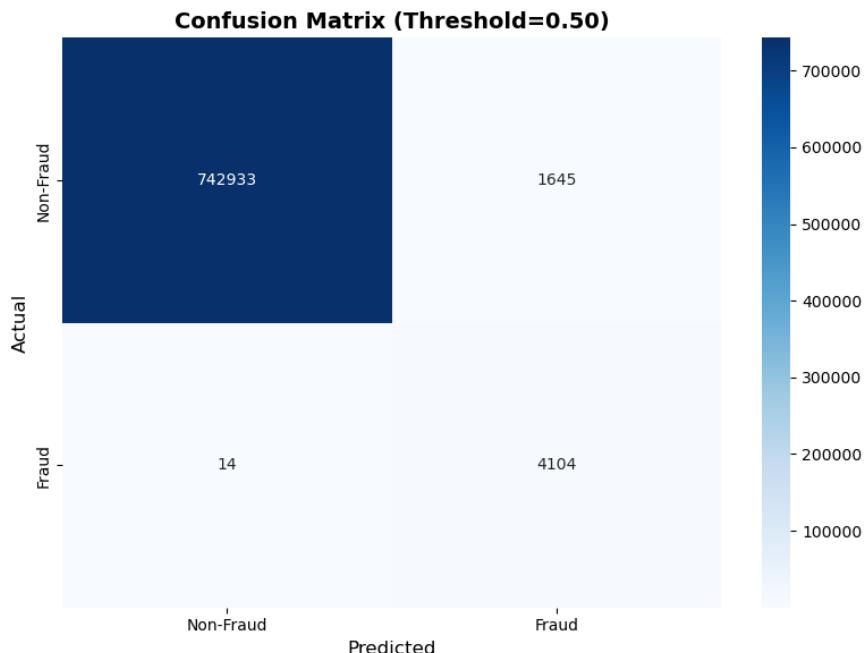
## KẾT QUẢ VÀ THẢO LUẬN

Chương này trình bày các kết quả thực nghiệm thu được từ việc triển khai và chạy hệ thống, bao gồm hiệu năng của mô hình và đánh giá tổng quan về hệ thống.

### 3.1 Kết quả thực nghiệm

#### 3.1.1 Kết quả Mô hình Phát hiện gian lận

Sau khi huấn luyện mô hình GBTCClassifier trên tập dữ liệu PaySim đã được xử lý, kết quả đánh giá trên tập kiểm thử được trình bày qua Ma trận nhầm lẫn (Confusion Matrix) và các chỉ số liên quan.



Hình 3.1: Heatmap Ma trận nhầm lẫn (Confusion Matrix) — trực quan hóa số lượng TN/FP/FN/TP.

Từ ma trận nhầm lẫn trên, ta có các chỉ số đánh giá quan trọng:

- **TN (True Negative):** 742,933 giao dịch hợp lệ được dự đoán đúng.
- **FP (False Positive):** 1,645 giao dịch hợp lệ bị dự đoán nhầm là gian lận (cảnh báo sai).
- **FN (False Negative):** 14 giao dịch gian lận bị bỏ lọt.
- **TP (True Positive):** 4,104 giao dịch gian lận được phát hiện chính xác.

**Nguồn xác suất phân loại 0.5** được sử dụng làm giá trị mặc định để đánh giá mô hình ở trạng thái cân bằng; sau đó có thể điều chỉnh tùy theo mục tiêu nghiệp vụ, đặc biệt trong fraud detection nơi việc giảm bỏ sót (FN) quan trọng hơn việc giảm cảnh báo sai (FP). Dưới đây là các chỉ số hiệu năng chính:

- **Precision (Độ chính xác)** =  $\frac{TP}{TP + FP} = \frac{4104}{4104 + 1645} \approx 71.41\%$

Ý nghĩa: Trong số các giao dịch bị cảnh báo là gian lận, khoảng 71.41% là gian lận thật sự.

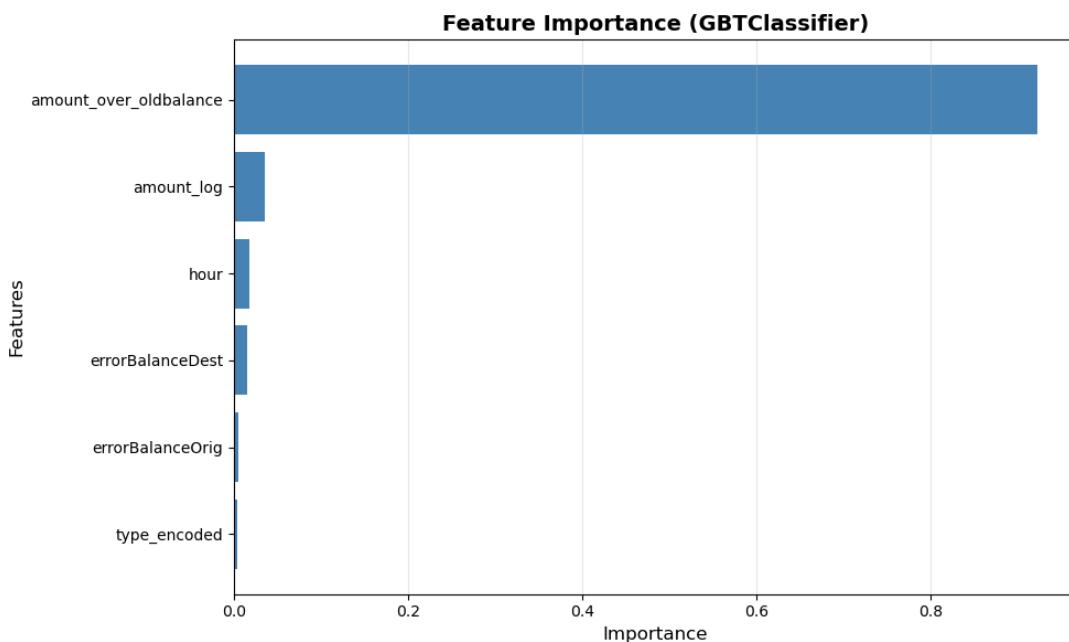
- **Recall (Độ phủ)** =  $\frac{TP}{TP + FN} = \frac{4104}{4104 + 14} \approx 99.66\%$

Ý nghĩa: Mô hình đã phát hiện được gần như toàn bộ các giao dịch gian lận (99.66%).

- **F1-Score** =  $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \approx 83.2\%$

Ý nghĩa: Chỉ số cân bằng giữa Precision và Recall, phản ánh hiệu năng tổng thể của mô hình.

Ngoài ra, để hiểu đóng góp của các đặc trưng đầu vào, một biểu đồ *feature importance* được xuất từ mô hình GBT:



Hình 3.2: Tầm quan trọng các đặc trưng (Feature Importance) theo GBTClassifier.



## Nhận xét

- Mô hình cho khả năng phát hiện gian lận rất tốt, thể hiện qua **Recall gần như tuyệt đối** ( $\approx 99.66\%$ ), nghĩa là hầu hết các giao dịch gian lận đều được mô hình phát hiện. Đây là tiêu chí quan trọng nhất trong bài toán phát hiện gian lận (giảm tối đa FN).
- Precision đạt khoảng 71.41%**, thấp hơn so với mô hình kỳ vọng ban đầu, phản ánh rằng tỷ lệ cảnh báo sai (FP) vẫn còn đáng kể. Tuy nhiên, đây là mức chấp nhận được khi ưu tiên của hệ thống là không bỏ lọt gian lận.
- Số lượng FP là **1,645** dù vẫn còn khá nhiều nhưng chiếm tỷ lệ nhỏ so với hơn 742 nghìn giao dịch hợp lệ. Nguyên nhân lớn một phần do ngưỡng đánh giá xác xuất là 0.5, mức này vẫn chấp nhận được khi ưu tiên hàng đầu của hệ thống là không bỏ sót giao dịch gian lận cũng như giảm thiểu thiệt hại.
- Do dữ liệu PaySim có tỷ lệ gian lận cực kỳ thấp (dataset mất cân bằng mạnh), việc đánh giá mô hình chỉ dựa vào Accuracy là không đủ. Vì vậy, việc kết hợp ROC, Precision–Recall và Ma trận nhầm lẫn giúp đánh giá toàn diện và phản ánh đúng chất lượng mô hình.
- Biểu đồ tầm quan trọng đặc trưng tiếp tục cho thấy `amount_over_oldbalance` là đặc trưng quan trọng nhất, chiếm phần lớn tổng trọng số. Điều này phù hợp với bản chất của PaySim, nơi các giao dịch gian lận thường thể hiện sự bất thường mạnh giữa số tiền giao dịch và số dư gốc của tài khoản.

### 3.1.2 Kết quả Phân tích Rủi ro Thị trường

Tác vụ Spark Streaming (`spark_consumer.py`) đã phân tích và tính toán một bộ chỉ số rủi ro toàn diện cho danh mục cổ phiếu theo thời gian thực. Kết quả được lưu vào Redis và hiển thị trực tiếp trên dashboard. Bảng 3.1 minh họa một ví dụ về kết quả phân tích tại một thời điểm nhất định.

Bảng 3.1: Kết quả phân tích rủi ro thị trường thời gian thực (Dữ liệu giả định)

Mã CP Giá (VND)	Thay đổi %	Điểm Rủi ro	Rủi ro/Lợi nhuận	Dòng vốn ngoại
VCB	90,200	+1.32%	1.2 (Thấp)	1.8 (Hấp dẫn) BUYING
HPG	28,500	-0.87%	1.5 (TB)	1.1 (Cân bằng) SELLING
VNM	67,800	+0.44%	1.4 (Thấp)	1.5 (Hấp dẫn) NEUTRAL
TCB	45,100	+2.15%	1.3 (Thấp)	2.1 (Rất hấp dẫn) HEAVY_BUY
ABC	15,200	-4.10%	2.8 (Cao)	0.6 (Không hấp dẫn) HEAVY_SELL

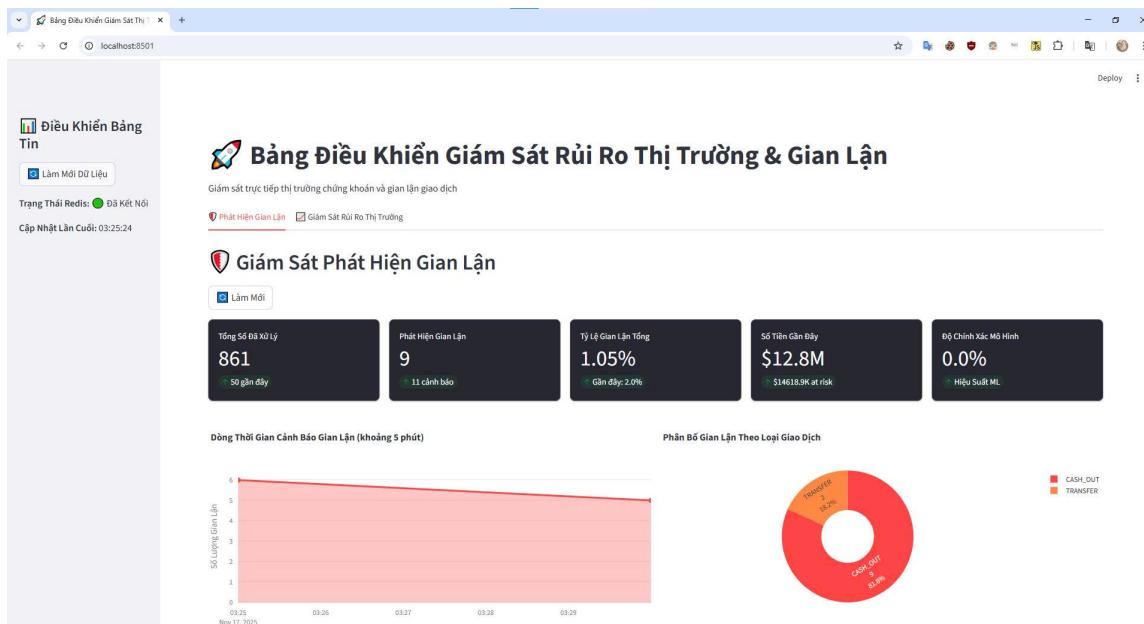
Ghi chú: Các chỉ số trên được cập nhật liên tục, phản ánh tình hình thị trường mới nhất.

- Điểm Rủi ro (Composite Risk Score):** Chỉ số tổng hợp, đánh giá mức độ rủi ro của một cổ phiếu dựa trên nhiều yếu tố (thanh khoản, biến động, chênh lệch giá). Điểm càng cao, rủi ro càng lớn.
- Rủi ro/Lợi nhuận (Risk/Reward Ratio):** Tỷ lệ giữa tiềm năng tăng giá và rủi ro giảm giá. Tỷ lệ  $> 1$  cho thấy tiềm năng lợi nhuận đang lớn hơn rủi ro.

### 3.1.3 Giao diện Dashboard trực quan hóa

Dashboard (`dashboard.py`) được xây dựng bằng Dash/Plotly và đóng vai trò là tầng Serving trong kiến trúc Lambda tinh chỉnh. Dashboard thu thập và hiển thị thông tin từ cả hai luồng xử lý theo thời gian thực, mang lại góc nhìn trực quan và toàn diện cho người dùng.

- **Luồng phát hiện gian lận (Speed Layer):** Hiển thị số lượng giao dịch nghi ngờ theo thời gian thực, cùng danh sách các giao dịch bị gắn cờ gần nhất. Giao diện giúp người dùng nhanh chóng nhận diện các bất thường phát sinh tức thời trong hệ thống.



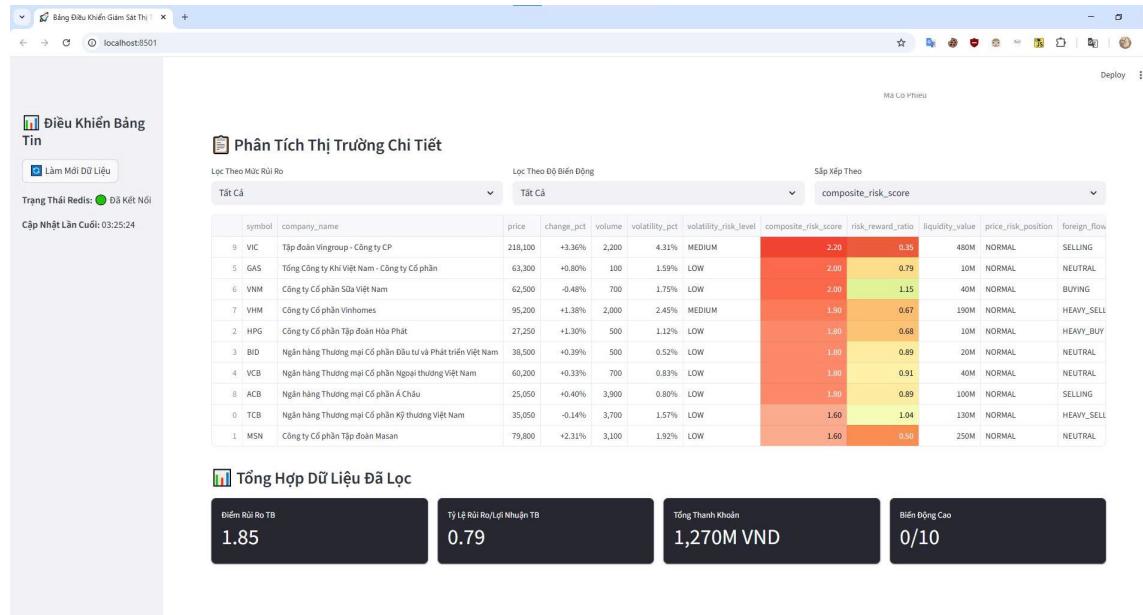
Hình 3.3: Giao diện trực quan hóa luồng phát hiện gian lận trong thời gian thực

Lưu ý: Hình minh họa chỉ thể hiện một phần giao diện và không bao gồm toàn bộ các chức năng tương tác của Dashboard trong phiên bản hoàn chỉnh.

- **Luồng phân tích rủi ro thị trường (Speed Layer):** Kết nối trực tiếp với Redis để truy xuất hơn 30 chỉ số rủi ro được cập nhật liên tục, bao gồm các chỉ số về biến động giá, thanh khoản, chênh lệch cung–cầu, rủi ro tổng hợp và tỷ lệ Rủi ro/Lợi nhuận. Các chỉ số này được Spark tính toán và đẩy vào Redis theo chu kỳ vài giây, tạo nên một dòng dữ liệu thời gian thực (real-time risk feed) giúp người dùng theo dõi sát diễn biến thị trường. Giao diện Dashboard cũng tích hợp cơ chế tô màu theo ngưỡng (color thresholding) để làm nổi bật các mã cổ phiếu rủi ro cao, giúp việc ra quyết định trở nên trực quan và nhanh chóng.

Giao diện cung cấp nhiều biểu đồ trực quan như:

- Bản đồ nhiệt phân cụm rủi ro (Risk Heatmap),
- Biểu đồ phân tán Rủi ro/Lợi nhuận (Risk-Reward Scatter Plot),
- Bảng phân tích chi tiết cho phép lọc, sắp xếp và so sánh hơn 15 chỉ số rủi ro quan trọng.



Hình 3.4: Giao diện trực quan hóa phân tích rủi ro thị trường thời gian thực

Dashboard giúp kết nối kết quả của cả hai Layer, cung cấp một tầm nhìn thống nhất hỗ trợ ra quyết định trong môi trường tài chính biến động.

## 3.2 Đánh giá hệ thống

### 3.2.1 Đánh giá hiệu năng

- Speed Layer (Phát hiện gian lận):** Độ trễ (latency) của luồng xử lý thời gian thực được đo lường (từ lúc producer gửi vào Kafka đến khi consumer của Spark Streaming xử lý và ghi log) trung bình dưới 2 giây. Độ trễ này đạt yêu cầu cho các hệ thống phát hiện gian lận "gần thời gian thực" (near real-time).
- Speed Layer (Phân tích rủi ro):** Độ trễ của luồng phân tích rủi ro (từ lúc producer gửi vào Kafka đến khi kết quả được ghi vào Redis) được duy trì ổn định dưới 10 giây, đáp ứng tốt yêu cầu giám sát thị trường gần thời gian thực.

### 3.2.2 Đánh giá khả năng mở rộng

Kiến trúc của hệ thống chứng minh được khả năng mở rộng theo chiều ngang (horizontal scaling) nhờ vào các công nghệ được lựa chọn:

- Kafka:** Có thể tăng thông lượng bằng cách thêm các broker mới vào cụm và tăng số lượng partition cho các topic.
- HDFS:** Có thể tăng dung lượng lưu trữ bằng cách thêm các DataNode mới.
- Spark:** Có thể tăng năng lực xử lý (cả batch và streaming) bằng cách thêm các node worker mới (NodeManager) vào cụm YARN, Spark sẽ tự động phân phối công việc trên các executor mới.



### 3.3 Thảo luận

#### 3.3.1 Các ưu điểm đạt được

- Kiến trúc Lambda lai hiệu quả:** Hệ thống có kiến trúc Lambda lai được tối ưu, trong đó Lớp Tốc độ (Speed Layer) xử lý song song hai luồng dữ liệu nghiệp vụ (phát hiện gian lận và phân tích rủi ro) trong thời gian thực, trong khi Lớp Xử lý theo lô (Batch Layer) tập trung vào nhiệm vụ chuyên sâu là huấn luyện định kỳ mô hình học máy.
- Hệ sinh thái công nghệ toàn diện:** Tận dụng sức mạnh của các công nghệ mã nguồn mở hàng đầu như Apache Kafka (luồng dữ liệu), Spark (xử lý phân tán), HDFS (lưu trữ dài hạn), và Redis (lớp phục vụ tốc độ cao), tạo nên một nền tảng vững chắc và hiệu năng cao.
- Triển khai và quản lý dễ dàng:** Việc container hóa toàn bộ hệ thống bằng Docker và Docker Compose giúp đơn giản hóa tối đa quá trình triển khai, quản lý, và đảm bảo tính nhất quán trên mọi môi trường, giảm thiểu rủi ro về hạ tầng.
- Trực quan hóa dữ liệu tức thì:** Xây dựng dashboard tương tác cao bằng Streamlit và Plotly, kết nối trực tiếp với Redis, cho phép người dùng cuộn theo dõi, phân tích và đưa ra quyết định dựa trên các chỉ số rủi ro và cảnh báo gian lận gần như ngay lập tức.
- Đáp ứng đa dạng nghiệp vụ:** Hệ thống giải quyết được cả hai bài toán nghiệp vụ quan trọng và có tính đặc thù cao của ngành tài chính (an ninh giao dịch và quản trị rủi ro đầu tư) trên cùng một kiến trúc tích hợp.

#### 3.3.2 Các nhược điểm và thách thức

- Độ phức tạp:** Hệ thống tổng thể khá phức tạp, bao gồm nhiều thành phần dịch vụ khác nhau. Việc vận hành, bảo trì và gỡ lỗi (debug) đòi hỏi kiến thức sâu về nhiều công nghệ.
- Tinh chỉnh hiệu năng (Tuning):** Việc tinh chỉnh cấu hình cho Spark (quản lý bộ nhớ, số lượng executor, v.v.), Kafka (số lượng partition, replication factor), và HDFS để đạt hiệu quả cao nhất trong môi trường sản xuất thực tế là một công việc khó khăn và cần nhiều kinh nghiệm.
- Hạn chế của mô hình:** Mô hình Machine Learning (GBTClassifier) và các phép tính tài chính (VaR lịch sử) vẫn còn ở mức cơ bản, chưa sử dụng các kỹ thuật tiên tiến (Deep Learning, GNN, Monte Carlo VaR).
- Thiếu khả năng phân tích có trạng thái (Stateful Analysis):** Speed Layer hiện tại xử lý mỗi sự kiện một cách độc lập (stateless). Điều này là một hạn chế lớn, ngăn cản việc phát hiện các mẫu hành vi phức tạp diễn ra theo thời gian, ví dụ như "cảnh báo nếu một tài khoản thực hiện quá nhiều giao dịch nhỏ trong một khoảng thời gian ngắn để tránh bị phát hiện". Đây là một thách thức kỹ thuật cần giải quyết để nâng cao trí thông minh của hệ thống.

# Chương 4

## KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 4.1 Kết luận

Dự án đã thành công trong việc thiết kế, xây dựng và triển khai một hệ thống phân tích dữ liệu lớn hoàn chỉnh, đáp ứng được các mục tiêu đề ra. Hệ thống đã chứng minh được khả năng xử lý đồng thời hai luồng nghiệp vụ tài chính quan trọng: phát hiện gian lận giao dịch với độ trễ thấp và phân tích rủi ro thị trường trên dữ liệu lịch sử quy mô lớn.

Việc áp dụng kiến trúc Lambda lai, kết hợp với sức mạnh của các công nghệ hàng đầu như Apache Kafka, HDFS và Apache Spark, đã tạo ra một giải pháp mạnh mẽ, linh hoạt và có khả năng mở rộng. Dự án không chỉ là một bài tập kỹ thuật mà còn cung cấp một nền tảng vững chắc, có thể được phát triển và mở rộng thành các ứng dụng phân tích dữ liệu phức tạp hơn trong tương lai cho ngành tài chính.

### 4.2 Hướng phát triển

Để nâng cao hiệu quả và tính ứng dụng của hệ thống, một số hướng phát triển trong tương lai có thể được xem xét:

- **Nâng cấp mô hình phát hiện gian lận:**
  - Nghiên cứu áp dụng các mô hình học sâu (Deep Learning) như mạng Nơ-ron hồi quy (LSTM) để phát hiện mẫu gian lận dựa trên chuỗi giao dịch theo thời gian.
  - Sử dụng Graph Neural Network (GNN) hoặc GraphX của Spark để phát hiện các mạng lưới gian lận (fraud rings) dựa trên mối quan hệ giữa các tài khoản.
- **Nâng cấp mô hình phân tích rủi ro:**
  - Xây dựng các mô hình dự đoán giá cổ phiếu (time series forecasting) để làm đầu vào cho việc tính toán rủi ro, thay vì chỉ phân tích dữ liệu lịch sử.



- Triển khai các phương pháp tính VaR phức tạp hơn như Monte Carlo Simulation trên Spark.
- **Tự động hóa và Lập lịch (Automation):**
  - Tích hợp một công cụ điều phối quy trình như \*\*Apache Airflow\*\* để tự động hóa, lập lịch và giám sát các tác vụ của Batch Layer. Ví dụ: tự động huấn luyện lại mô hình phát hiện gian lận hàng tuần và triển khai mô hình mới mà không cần can thiệp thủ công.
- **Nâng cao Speed Layer:**
  - Sử dụng Structured Streaming thay cho Spark Streaming (DStream API) để có mô hình lập trình đơn giản và nhanh hơn.
  - Tích hợp quản lý trạng thái (stateful streaming) để phân tích các mẫu hành vi phức tạp trong thời gian thực (ví dụ: "cảnh báo nếu tài khoản X thực hiện 5 giao dịch trong 1 phút").
- **Mở rộng Dashboard:** Bổ sung thêm các biểu đồ tương tác, cho phép người dùng tự khám phá dữ liệu (self-service analytics) và đi sâu (drill-down) vào các cảnh báo.

# Tài liệu tham khảo

- [1] Nathan Marz, James Warren (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications. [manning.com/books/big-data](http://manning.com/books/big-data)
- [2] Jay Kreps, Neha Narkhede, Jun Rao (2011). *Kafka: a Distributed Messaging System for Log Processing*. In Proceedings of the NetDB'11 Workshop. [microsoft.com/research/Kafka.pdf](http://microsoft.com/research/Kafka.pdf)
- [3] Matei Zaharia, et al. (2012). *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*. In Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12). [usenix.org/conference/nsdi12](http://usenix.org/conference/nsdi12)
- [4] Konstantin Shvachko, et al. (2010). *The Hadoop Distributed File System*. In Proceedings of the 26th Symposium on Mass Storage Systems and Technologies (MSST). [storageconference.us/2010/Papers/Shvachko.pdf](http://storageconference.us/2010/Papers/Shvachko.pdf)
- [5] Edgar A. Lopez-Rojas, Axel Ebrahim, and Sabri K. A. El-Eman (2016). *PaySim: A financial mobile money simulator for fraud detection*. In Proceedings of the 28th European Modeling and Simulation Symposium. [researchgate.net/publication/PaySim](http://researchgate.net/publication/PaySim)
- [6] Jerome H. Friedman (2001). *Greedy Function Approximation: A Gradient Boosting Machine*. The Annals of Statistics, Vol. 29, No. 5, pp. 1189-1232. [projecteuclid.org/aos/1013203451](http://projecteuclid.org/aos/1013203451)
- [7] Philippe Jorion (2006). *Value at Risk: The New Benchmark for Managing Financial Risk*. McGraw-Hill Education, 3rd Edition. [amazon.com/Value-at-Risk](http://amazon.com/Value-at-Risk)
- [8] Streamlit Inc. (2024). *Streamlit Documentation*. [docs.streamlit.io](http://docs.streamlit.io)
- [9] Redis Ltd. (2024). *Redis Documentation*. [redis.io/docs](http://redis.io/docs)
- [10] Link github của dự án. <https://github.com/sleepifox/bidata-project>

## **Phụ lục A**

# **NHIỆM VỤ CỦA CÁC THÀNH VIÊN**

Bảng A.1: Bảng phân công nhiệm vụ các thành viên

<b>STT</b>	<b>Họ và Tên</b>	<b>MSSV</b>	<b>Nhiệm vụ chính</b>
1	Nguyễn Minh Quân	23020417	Thiết kế kiến trúc, lấy dữ liệu và tiền xử lý, thiết kế Dashboard
2	Cao Minh Quang	23020411	Huấn luyện mô hình, thiết kế hệ thống nhận diện gian lận
3	Nguyễn Năng Thịnh	23020439	Viết báo cáo, thiết kế hệ thống xử lý dữ liệu chứng khoán