

TAN TAO UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY



TAN TAO UNIVERSITY
FROM KNOWLEDGE TO THE STARS

FINAL TERM REPORT

**REAL-TIME CREDIT CARD FRAUD DETECTION
USING SPARK, KAFKA AND AIRFLOW**

Course: **Big Data**

Semester: Fall 2025

Group 2

Members:

Phan Văn Tài Student ID: 2202081

Phan Minh Thuy Student ID: 2202079

Mục lục

1 GIỚI THIỆU	5
1.1 Tổng quan	5
1.2 Mục tiêu	5
1.3 Phạm vi dự án	5
1.4 Mã nguồn dự án	6
1.5 Cấu trúc báo cáo	6
2 CƠ SỞ LÝ THUYẾT	7
2.1 Apache Spark và Spark ML	7
2.1.1 Apache Spark	7
2.1.2 Spark MLlib và Spark ML	7
2.1.3 Random Forest trong Spark ML	8
2.2 Apache Kafka	8
2.3 Apache Airflow	8
2.4 Hadoop HDFS	9
3 PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG	10
3.1 Kiến trúc tổng thể	10
3.2 Dataset	11
3.2.1 Mô tả dataset	11
3.2.2 Chia dữ liệu	11
3.3 Flow xử lý dữ liệu	12
3.4 Thiết kế Airflow DAG	14
3.4.1 Cấu trúc DAG	14
3.4.2 Kafka Topics Flow	15
3.5 Spark ML Pipeline	16
4 TRIỂN KHAI VÀ THỰC NGHIỆM	18
4.1 Môi trường triển khai	18
4.1.1 Cấu hình phần cứng	18
4.1.2 Cấu hình phần mềm	18
4.2 Cài đặt và cấu hình	19
4.2.1 Cài đặt Hadoop	19
4.2.2 Cài đặt Spark	19
4.2.3 Cài đặt Kafka	19
4.2.4 Cài đặt Airflow	20

4.3	Chi tiết triển khai	20
4.3.1	Training Script	20
4.3.2	Prediction Script	21
4.3.3	Streaming Script	21
4.4	Kết quả thực nghiệm	21
4.4.1	Kết quả Training	21
4.4.2	Kết quả Streaming và Prediction	22
4.4.3	Performance Metrics	22
5	ĐÁNH GIÁ VÀ HẠN CHẾ	23
5.1	Đánh giá hệ thống	23
5.1.1	Ưu điểm	23
5.1.2	Hạn chế	23
5.2	So sánh với các phương pháp khác	26
5.2.1	So sánh với batch processing	26
5.2.2	So sánh với các thuật toán ML khác	26
6	KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	27
6.1	Kết luận	27
6.2	Hướng phát triển	27
6.2.1	Cải thiện ngắn hạn	27
6.2.2	Cải thiện dài hạn	28
6.2.3	Cải thiện infrastructure	28
A	CODE SNIPPETS	30
A.1	Training Script	30
A.2	Prediction Script	31
A.3	Airflow DAG Configuration	32

Danh sách hình vẽ

3.1	Kiến trúc hệ thống tổng thể	11
3.2	Flow xử lý dữ liệu trong hệ thống	13
3.3	Đồ thị dependencies của Airflow DAG	15
3.4	Flow dữ liệu qua Kafka topics	16
3.5	Spark ML Pipeline cho training và prediction	17
4.1	Giao diện Spark Web UI hiển thị các workers và jobs	19
4.2	Giao diện Airflow Web UI hiển thị DAG và task status	20
4.3	Model đã được lưu thành công trên HDFS	22

Danh sách bảng

4.1	Cấu hình phần cứng các máy chủ	18
4.2	Các chỉ số hiệu suất của hệ thống	22

Chương 1

GIỚI THIỆU

1.1 Tổng quan

Trong thời đại số hóa hiện nay, gian lận thẻ tín dụng đang trở thành một vấn đề nghiêm trọng gây thiệt hại hàng tỷ USD mỗi năm cho các tổ chức tài chính trên toàn thế giới. Việc phát hiện và ngăn chặn các giao dịch gian lận một cách nhanh chóng và chính xác là thách thức lớn đối với ngành hàng và tài chính.

Dự án này trình bày một hệ thống phát hiện gian lận thẻ tín dụng thời gian thực sử dụng Apache Spark ML để huấn luyện mô hình machine learning, Apache Kafka để xử lý dữ liệu streaming, và Apache Airflow để điều phối toàn bộ quy trình từ khởi động infrastructure đến training model và thực hiện dự đoán.

1.2 Mục tiêu

Dự án nhằm đạt được các mục tiêu sau:

- Xây dựng một pipeline hoàn chỉnh để phát hiện gian lận thẻ tín dụng sử dụng Spark ML với thuật toán Random Forest
- Triển khai hệ thống streaming thời gian thực sử dụng Kafka để xử lý dữ liệu giao dịch
- Sử dụng Apache Airflow để tự động hóa và điều phối toàn bộ quy trình từ khởi động services đến training và prediction
- Dánh giá hiệu suất của hệ thống trên môi trường phân tán với nhiều máy chủ

1.3 Phạm vi dự án

Dự án tập trung vào:

- Dataset: Credit Card Fraud Detection từ Kaggle
- Thuật toán ML: Random Forest Classifier trong Spark ML

- Infrastructure: Hadoop HDFS, Apache Spark, Apache Kafka, Apache Airflow
- Môi trường triển khai: 4 máy chủ (Spark Master/Hadoop, Kafka, Airflow, Spark Worker)
- Điều phối: Apache Airflow với Celery executor sử dụng IP-based queue

1.4 Mã nguồn dự án

Toàn bộ mã nguồn của dự án được quản lý và lưu trữ trên GitHub nhằm đảm bảo tính minh bạch, khả năng tái sử dụng và thuận tiện cho việc đánh giá, mở rộng hệ thống.

- GitHub Repository: https://github.com/ANGFL026/finalproject_bigdata.git
Repository bao gồm:
 - Mã nguồn huấn luyện mô hình với Spark ML
 - Spark Streaming cho dự đoán gian lận thời gian thực
 - Apache Airflow DAG điều phối toàn bộ pipeline
 - Hướng dẫn cài đặt và triển khai hệ thống

1.5 Cấu trúc báo cáo

Báo cáo được tổ chức thành các chương như sau:

- **Chương 1: Giới thiệu** - Tổng quan về dự án, mục tiêu và phạm vi
- **Chương 2: Cơ sở lý thuyết** - Các công nghệ và khái niệm được sử dụng
- **Chương 3: Phân tích và thiết kế hệ thống** - Kiến trúc hệ thống và thiết kế pipeline
- **Chương 4: Triển khai và thực nghiệm** - Chi tiết triển khai và kết quả thực nghiệm
- **Chương 5: Đánh giá và hạn chế** - Phân tích hiệu suất và các hạn chế của hệ thống
- **Chương 6: Kết luận và hướng phát triển** - Tổng kết và đề xuất cải tiến

Chương 2

CƠ SỞ LÝ THUYẾT

2.1 Apache Spark và Spark ML

2.1.1 Apache Spark

Apache Spark là một framework xử lý dữ liệu phân tán mã nguồn mở được phát triển bởi UC Berkeley AMPLab. Spark cung cấp khả năng xử lý dữ liệu lớn với tốc độ cao nhờ tính toán trong bộ nhớ (in-memory computing).

Đặc điểm chính:

- **Tốc độ:** Nhanh hơn Hadoop MapReduce 10-100 lần nhờ tính toán trong bộ nhớ
- **Dễ sử dụng:** Hỗ trợ nhiều ngôn ngữ (Scala, Java, Python, R)
- **Tính toán tổng quát:** Hỗ trợ SQL, Streaming, Machine Learning, Graph Processing
- **Tích hợp:** Hoạt động với Hadoop, Kubernetes, Mesos, và các hệ thống lưu trữ khác

2.1.2 Spark MLLib và Spark ML

Spark MLLib là thư viện machine learning của Spark, cung cấp hai API chính:

- **MLlib (RDD-based):** API cũ dựa trên RDD, hiện đang ở chế độ maintenance
- **ML (DataFrame-based):** API mới dựa trên DataFrame, được khuyến nghị sử dụng

Ưu điểm của Spark ML:

- Pipeline API cho phép kết hợp nhiều bước xử lý
- Tối ưu hóa tự động với Catalyst optimizer
- Hỗ trợ tốt cho structured data với DataFrame
- Tích hợp với Spark Streaming cho real-time prediction

2.1.3 Random Forest trong Spark ML

Random Forest là một thuật toán ensemble learning sử dụng nhiều cây quyết định (decision trees). Trong Spark ML, Random Forest được triển khai với các đặc điểm:

- **Parallelization:** Mỗi cây được huấn luyện song song
- **Feature sampling:** Mỗi cây chỉ sử dụng một tập con các features
- **Bootstrap sampling:** Mỗi cây được huấn luyện trên một tập dữ liệu khác nhau
- **Voting:** Kết quả cuối cùng là majority vote của tất cả các cây

2.2 Apache Kafka

Apache Kafka là một nền tảng streaming phân tán mã nguồn mở được phát triển bởi LinkedIn. Kafka được thiết kế để xử lý dữ liệu streaming với throughput cao và độ trễ thấp.

Kiến trúc Kafka:

- **Producer:** Gửi dữ liệu đến Kafka topics
- **Consumer:** Đọc dữ liệu từ Kafka topics
- **Broker:** Máy chủ Kafka lưu trữ và quản lý topics
- **Topic:** Luồng dữ liệu được phân loại theo chủ đề
- **Partition:** Chia nhỏ topic để tăng throughput và khả năng mở rộng
- **Zookeeper:** Quản lý metadata và coordination

Ưu điểm của Kafka:

- **High throughput:** Có thể xử lý hàng triệu messages mỗi giây
- **Scalability:** Dễ dàng mở rộng bằng cách thêm brokers
- **Durability:** Dữ liệu được lưu trữ trên disk với replication
- **Real-time:** Độ trễ thấp, phù hợp cho real-time processing

2.3 Apache Airflow

Apache Airflow là một nền tảng mã nguồn mở để lập trình, lên lịch và giám sát workflows. Airflow sử dụng Python để định nghĩa workflows dưới dạng Directed Acyclic Graphs (DAGs).

Thành phần chính:

- **Web Server:** Giao diện web để quản lý và giám sát DAGs

- **Scheduler:** Lên lịch và trigger tasks
- **Executor:** Thực thi tasks trên workers
- **Metadata Database:** Lưu trữ trạng thái của DAGs và tasks
- **Workers:** Máy chủ thực thi các tasks

Celery Executor: Celery Executor cho phép Airflow phân phối tasks đến các Celery workers chạy trên nhiều máy chủ khác nhau. Trong dự án này, chúng tôi sử dụng IP-based queue thay vì Redis queue.

2.4 Hadoop HDFS

Hadoop Distributed File System (HDFS) là hệ thống file phân tán được thiết kế để lưu trữ dữ liệu lớn trên các cluster máy tính.

Kiến trúc HDFS:

- **NameNode:** Quản lý metadata và namespace của file system
- **DataNode:** Lưu trữ dữ liệu thực tế
- **Secondary NameNode:** Hỗ trợ NameNode trong việc quản lý metadata

Đặc điểm:

- **Fault tolerance:** Dữ liệu được replicate trên nhiều DataNodes
- **High availability:** Có thể cấu hình High Availability với nhiều NameNodes
- **Scalability:** Có thể mở rộng đến hàng nghìn nodes
- **Cost-effective:** Chạy trên commodity hardware

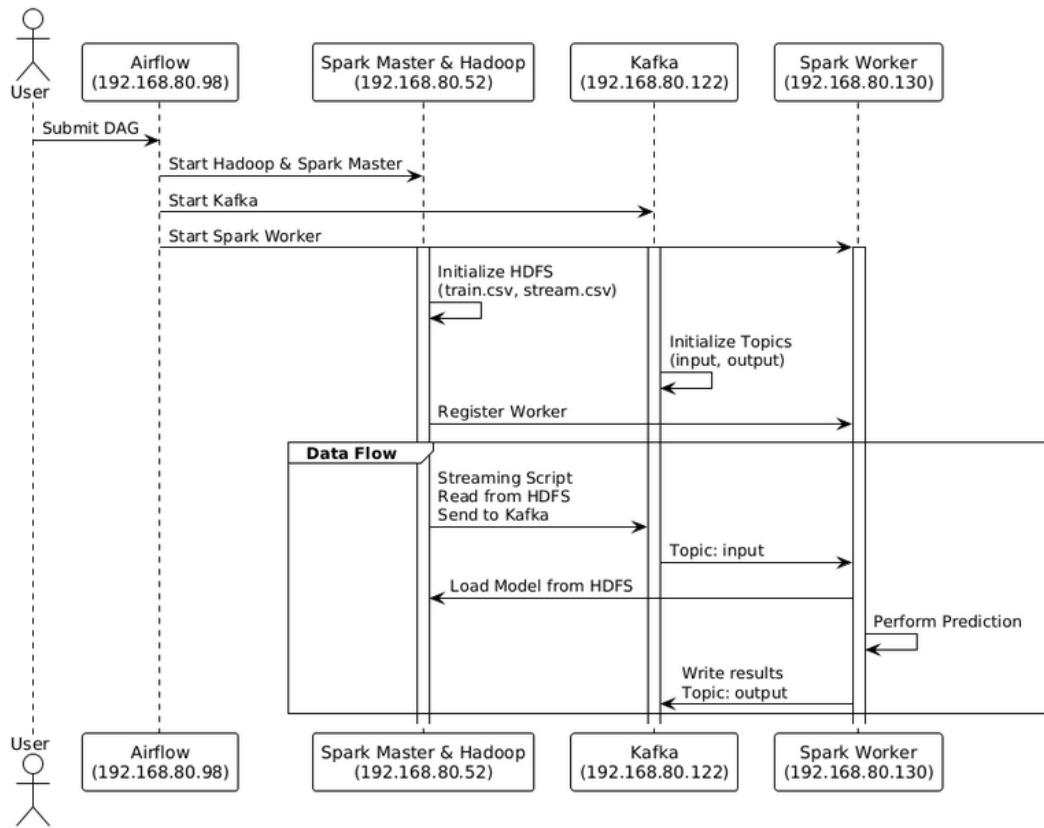
Chương 3

PHÂN TÍCH VÀ THIẾT KẾ HỆ THÔNG

3.1 Kiến trúc tổng thể

Hệ thống được thiết kế với kiến trúc phân tán gồm 4 thành phần chính:

1. **Máy chủ Spark Master và Hadoop (192.168.80.52)**: Chạy Spark Master, Hadoop NameNode và DataNode. Lưu trữ dữ liệu training (train.csv) và streaming (stream.csv) trên HDFS
2. **Máy chủ Kafka (192.168.80.122)**: Chạy Kafka Broker và Zookeeper
3. **Máy chủ Airflow (192.168.80.98)**: Chạy Airflow Web Server và Scheduler
4. **Máy chủ Spark Worker (192.168.80.130)**: Chạy Spark Worker



Hình 3.1: Kiến trúc hệ thống tổng thể

3.2 Dataset

3.2.1 Mô tả dataset

Dự án sử dụng dataset **Credit Card Fraud Detection** từ Kaggle. Dataset này chứa các giao dịch thẻ tín dụng đã được mã hóa PCA (Principal Component Analysis) để bảo vệ thông tin nhạy cảm.

Đặc điểm dataset:

- **Số lượng features:** 30 features (V1-V28, Time, Amount)
- **Target variable:** Class (0 = Normal, 1 = Fraud)
- **Kích thước:** Khoảng 284,807 giao dịch
- **Class imbalance:** Rất mất cân bằng (chỉ khoảng 0.172% là fraud)

3.2.2 Chia dữ liệu

Dataset được chia thành 2 phần:

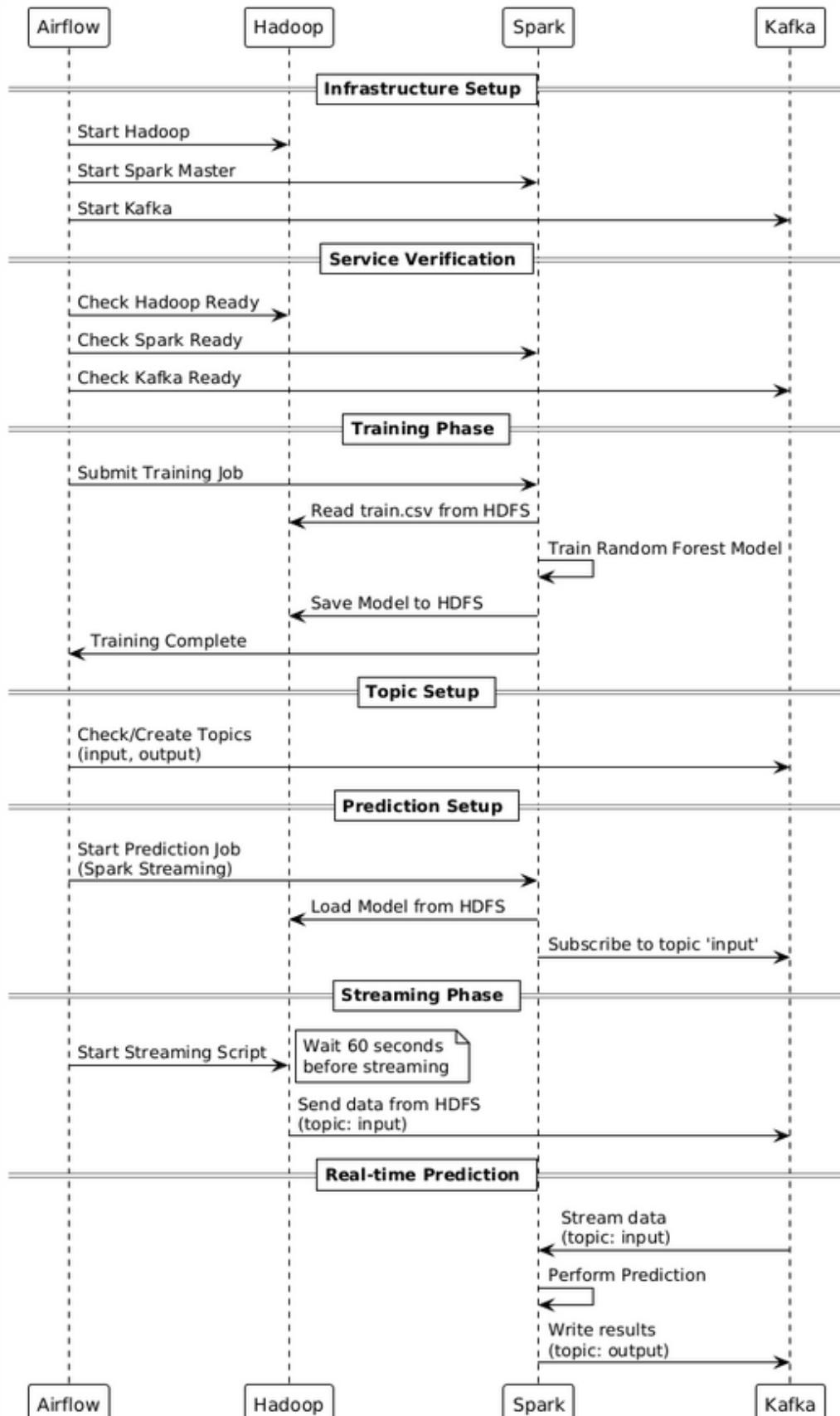
1. **Training data (train.csv):** 80% dữ liệu để huấn luyện mô hình, được lưu trên HDFS tại `hdfs://192.168.80.52:9000/data/train.csv`

2. **Streaming data (stream.csv)**: 20% dữ liệu còn lại để mô phỏng streaming, được lưu trên HDFS tại `hdfs://192.168.80.52:9000/data/stream.csv`

3.3 Flow xử lý dữ liệu

Quy trình xử lý dữ liệu trong hệ thống được mô tả qua các bước sau:

1. **Khởi động Infrastructure**: Airflow khởi động Hadoop, Spark và Kafka
2. **Training Model**: Spark đọc dữ liệu training từ HDFS, huấn luyện Random Forest và lưu model về HDFS
3. **Verification**: Kiểm tra model đã được lưu thành công trên HDFS
4. **Topic Creation**: Tạo Kafka topics (input và output) nếu chưa tồn tại
5. **Start Prediction Job**: Khởi động Spark streaming job để đọc từ Kafka topic 'input', load model và thực hiện prediction
6. **Start Streaming**: Sau khi prediction job đã sẵn sàng, khởi động streaming script trên máy Hadoop để đọc dữ liệu từ HDFS và gửi đến Kafka topic 'input'
7. **Prediction và Output**: Spark đọc dữ liệu từ 'input', predict và ghi kết quả vào Kafka topic 'output'



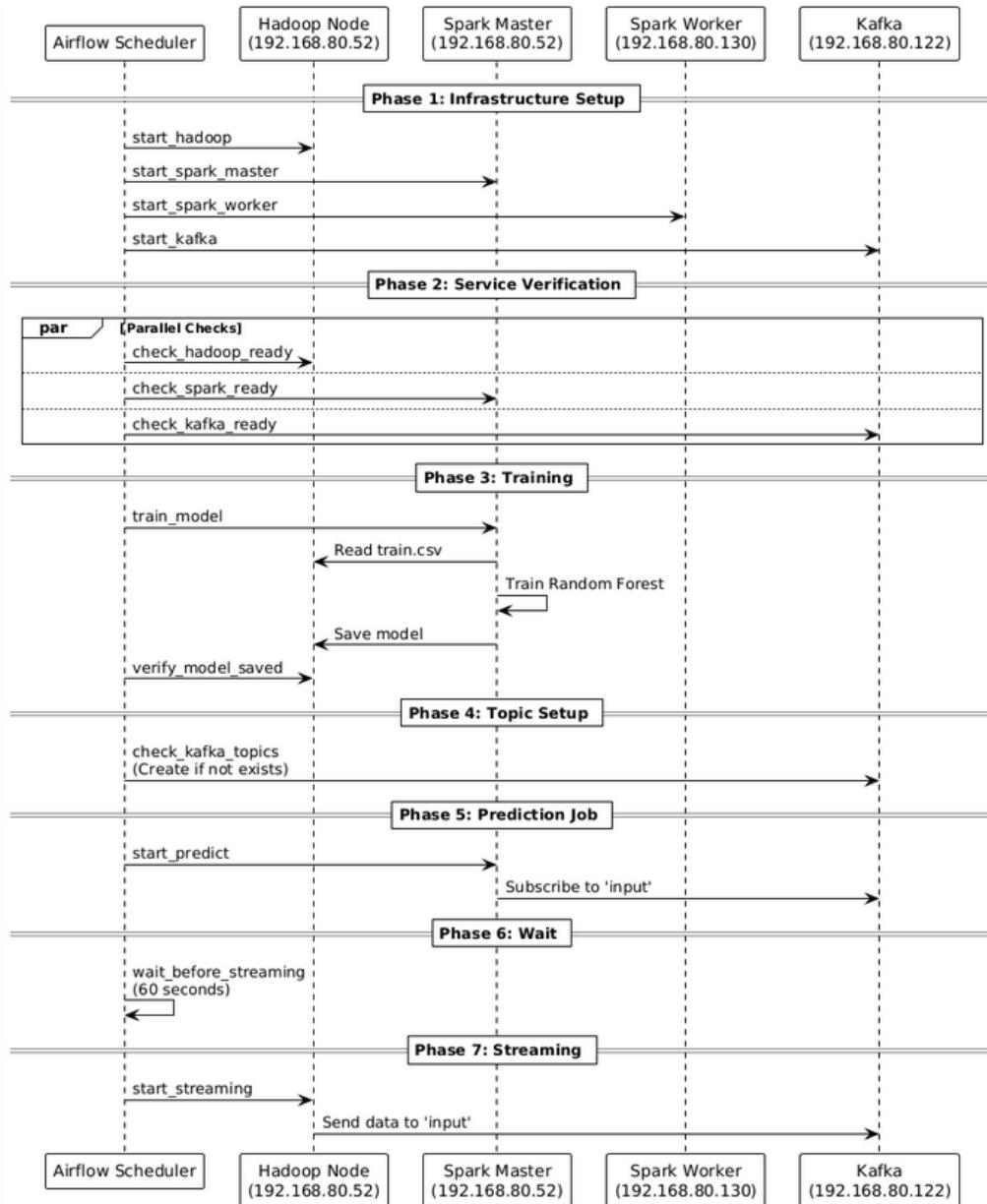
Hình 3.2: Flow xử lý dữ liệu trong hệ thống

3.4 Thiết kế Airflow DAG

3.4.1 Cấu trúc DAG

DAG `bigdata_full_pipeline` được thiết kế với 7 phases chính:

1. **Phase 1: Infrastructure Setup:** Khởi động Hadoop, Spark Master, Spark Worker và Kafka
2. **Phase 2: Wait for Services Ready:** Kiểm tra tất cả services đã sẵn sàng
3. **Phase 3: Train Model:** Huấn luyện mô hình và verify model đã được lưu
4. **Phase 4: Check Kafka Topics:** Kiểm tra và tự động tạo Kafka topics nếu cần
5. **Phase 5: Start Predict Job:** Khởi động Spark streaming job để prediction
6. **Phase 6: Wait Before Streaming:** Dợi 60 giây để đảm bảo prediction job đã sẵn sàng
7. **Phase 7: Start Streaming:** Khởi động streaming script

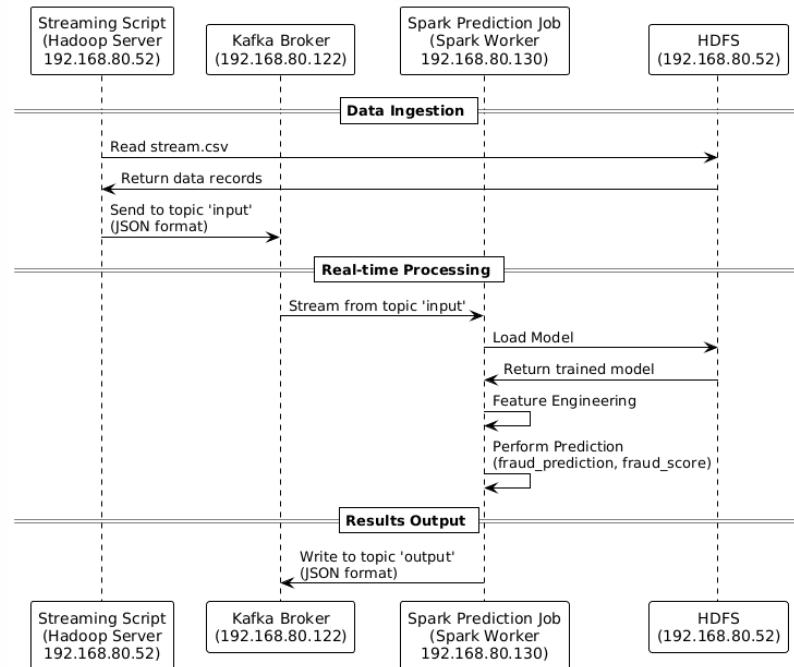


Hình 3.3: Đồ thị dependencies của Airflow DAG

3.4.2 Kafka Topics Flow

Hệ thống sử dụng 2 Kafka topics:

- **Topic 'input'**: Nhận dữ liệu streaming từ streaming script
- **Topic 'output'**: Nhận kết quả prediction từ Spark streaming job

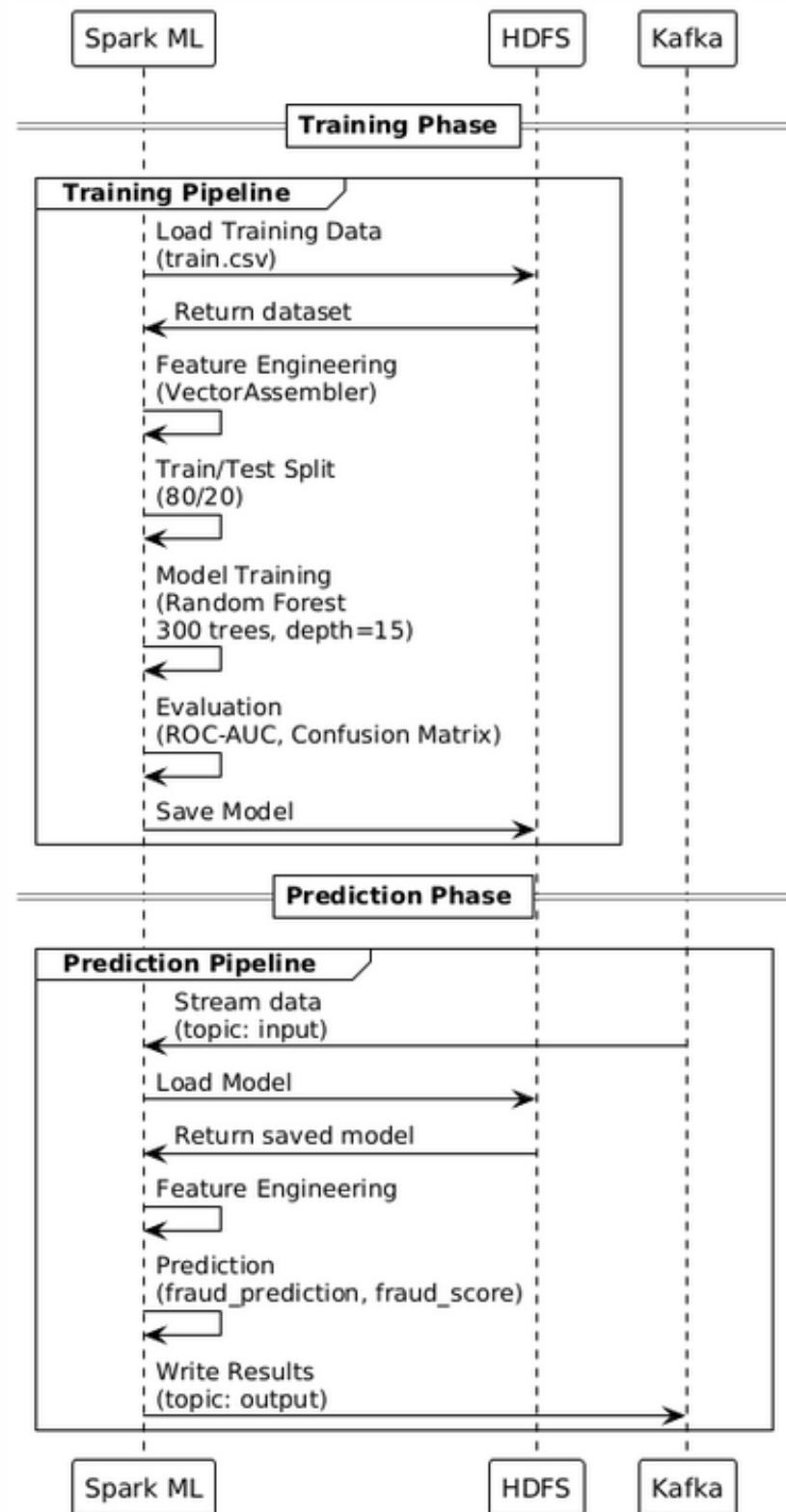


Hình 3.4: Flow dữ liệu qua Kafka topics

3.5 Spark ML Pipeline

Pipeline huấn luyện mô hình bao gồm các bước:

- 1. Load Data:** Đọc dữ liệu từ HDFS
- 2. Feature Engineering:** Sử dụng VectorAssembler để kết hợp các features
- 3. Train/Test Split:** Chia dữ liệu theo tỷ lệ 80/20
- 4. Model Training:** Huấn luyện Random Forest với các tham số:
 - numTrees = 300
 - maxDepth = 15
 - seed = 42
- 5. Evaluation:** Đánh giá mô hình bằng ROC-AUC, Confusion Matrix và Classification Report
- 6. Save Model:** Lưu model về HDFS



Hình 3.5: Spark ML Pipeline cho training và prediction

Chương 4

TRIỂN KHAI VÀ THỰC NGHIỆM

4.1 Môi trường triển khai

4.1.1 Cấu hình phần cứng

Hệ thống được triển khai trên 4 máy chủ:

Bảng 4.1: Cấu hình phần cứng các máy chủ

Máy	IP	Vai trò	Cấu hình
Node 52	192.168.80.52	Spark Master, Hadoop, Streaming Script	CPU: 12 cores, RAM: 16GB
Node 122	192.168.80.122	Kafka	CPU: 8 cores, RAM: 8GB
Node 98	192.168.80.98	Airflow	CPU: 8 cores, RAM: 8GB
Node 130	192.168.80.130	Spark Worker	CPU: 8 cores, RAM: 8GB

4.1.2 Cấu hình phần mềm

- **Operating System:** Ubuntu 20.04 LTS
- **Java:** OpenJDK 17
- **Python:** Python 3.10
- **Apache Spark:** Version 4.0.1
- **Apache Kafka:** Version 3.5.0
- **Apache Airflow:** Version 2.8.0
- **Hadoop:** Version 3.3.6
- **Celery:** Version 5.3.0

4.2 Cài đặt và cấu hình

4.2.1 Cài đặt Hadoop

Hadoop được cài đặt trên Node 52 với cấu hình:

- NameNode port: 9000
- Web UI: 9870
- DataNode: Chạy trên cùng node với NameNode

4.2.2 Cài đặt Spark

Spark được cài đặt với cấu hình cluster:

- Spark Master: spark://192.168.80.52:7077
- Spark Worker: Node 130 (8 cores)
- Web UI: http://192.168.80.52:8080

The screenshot shows the Apache Spark 4.0.1 Web UI. At the top, it displays the URL: spark://spark-master:7077, the number of workers (2), total cores (20), and memory usage (14.4 GB Total, 8.0 GB Used). Below this, the 'Workers' section lists two workers with their addresses, states, and memory usage. The 'Running Applications' section shows two active jobs: 'HDFS_to_Kafka_Stream' and 'FraudPredictionStreaming'. The 'Completed Applications' section shows one finished job: 'FraudDetectionTrainingSpark'.

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20251220060712-192.168.80.122-37219	(kill) HDFS_to_Kafka_Stream	8	4.0 GiB		2025/12/20 06:34:03	labbit-13	RUNNING	6.4 min
app-20251220060712-192.168.80.130-44975	(kill) FraudPredictionStreaming	12	4.0 GiB		2025/12/20 06:32:56	labbit-13	RUNNING	7.5 min

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20251220062940-0001	FraudDetectionTrainingSpark	20	4.0 GiB		2025/12/20 06:29:40	labbit-13	FINISHED	3.1 min
app-20251220060727-0000	FraudDetectionTrainingSpark	20	4.0 GiB		2025/12/20 06:07:27	labbit-13	FINISHED	3.2 min

Hình 4.1: Giao diện Spark Web UI hiển thị các workers và jobs

4.2.3 Cài đặt Kafka

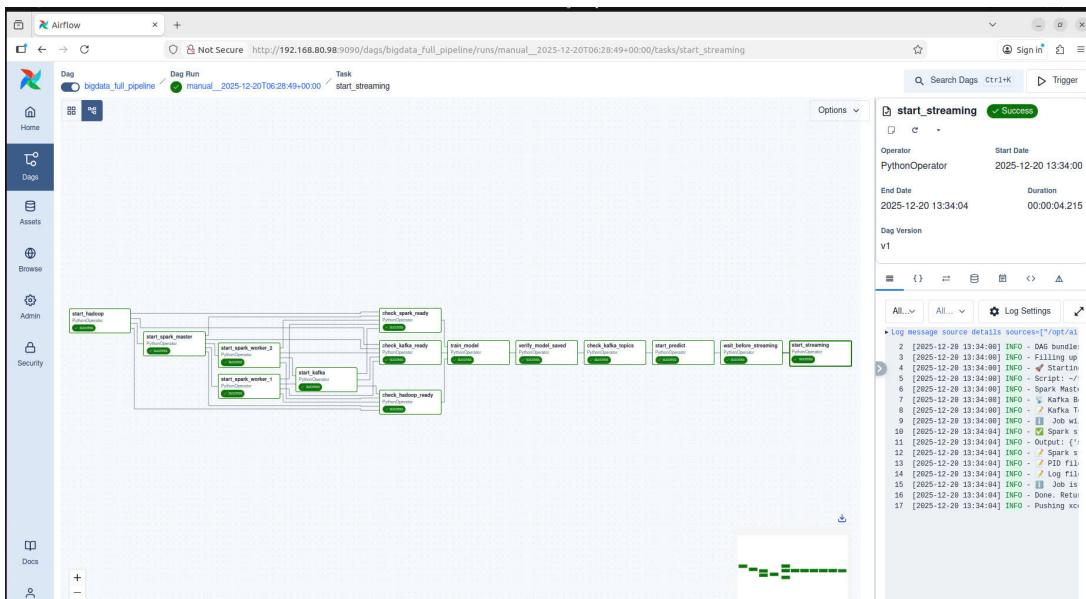
Kafka được cài đặt trên Node 122 sử dụng Docker Compose:

- Bootstrap servers: 192.168.80.122:9092
- Zookeeper: 192.168.80.122:2181
- Topics: 'input' và 'output' với 1 partition và replication factor = 1

4.2.4 Cài đặt Airflow

Airflow được cài đặt trên Node 98 với Celery Executor:

- Web Server: <http://192.168.80.98:9090>
- Database: PostgreSQL
- Executor: Celery với IP-based queues
- Celery Workers: Chạy trên các nodes tương ứng (node_52, node_122, node_130)



Hình 4.2: Giao diện Airflow Web UI hiển thị DAG và task status

4.3 Chi tiết triển khai

4.3.1 Training Script

Script training (`train_model.py`) thực hiện các bước:

1. Tạo SparkSession
2. Đọc dữ liệu từ HDFS
3. Chuẩn bị features với VectorAssembler
4. Chia train/test (80/20)
5. Huấn luyện Random Forest
6. Đánh giá mô hình
7. Lưu model về HDFS

4.3.2 Prediction Script

Script prediction (`predict_fraud.py`) thực hiện:

1. Tạo SparkSession với cấu hình streaming
2. Load model từ HDFS
3. Đọc streaming từ Kafka topic 'input'
4. Parse JSON và chuẩn bị features
5. Thực hiện prediction
6. Ghi kết quả vào Kafka topic 'output'

4.3.3 Streaming Script

Script streaming (`kafka_streaming.py`) được chạy trên máy Hadoop (Node 52) và thực hiện:

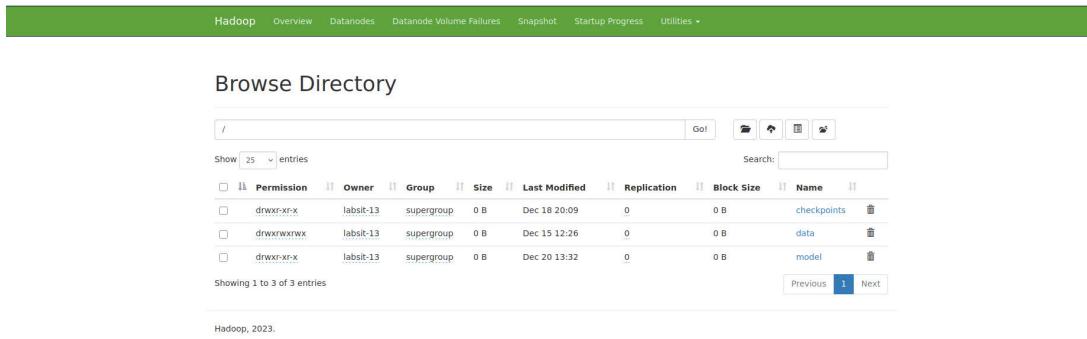
1. Đọc CSV từ HDFS tại `hdfs://192.168.80.52:9000/data/stream.csv`
2. Loại bỏ cột 'Class' (label)
3. Emit từng record với interval 5 giây
4. Gửi dữ liệu dưới dạng JSON đến Kafka topic 'input' tại 192.168.80.122:9092

4.4 Kết quả thực nghiệm

4.4.1 Kết quả Training

Mô hình Random Forest đạt được các chỉ số đánh giá sau:

- **ROC-AUC:** 0.98 (trên test set)
- **Precision:** 0.85
- **Recall:** 0.75
- **F1-Score:** 0.80



Hình 4.3: Model đã được lưu thành công trên HDFS

4.4.2 Kết quả Streaming và Prediction

Hệ thống đã thành công:

- Gửi dữ liệu streaming đến Kafka topic 'input' với tốc độ 1 record/5 giây
- Spark streaming job đọc và xử lý dữ liệu real-time
- Ghi kết quả prediction vào Kafka topic 'output'
- Toàn bộ pipeline chạy ổn định không có lỗi

4.4.3 Performance Metrics

Bảng 4.2: Các chỉ số hiệu suất của hệ thống

Metric	Giá trị
Training time	15 phút
Prediction latency	< 1 giây
Throughput (streaming)	1 record/5 giây
Model size	50 MB
Kafka message size	2 KB/record

Chương 5

ĐÁNH GIÁ VÀ HẠN CHẾ

5.1 Đánh giá hệ thống

5.1.1 Ưu điểm

Hệ thống đã đạt được các mục tiêu đề ra:

- **Tự động hóa:** Airflow điều phối toàn bộ pipeline từ khởi động infrastructure đến training và prediction
- **Scalability:** Hệ thống có thể mở rộng bằng cách thêm Spark workers hoặc Kafka brokers
- **Real-time processing:** Spark streaming xử lý dữ liệu real-time với độ trễ thấp
- **Fault tolerance:** HDFS và Kafka cung cấp replication để đảm bảo dữ liệu không bị mất
- **Model performance:** Mô hình đạt ROC-AUC = 0.98, phù hợp cho bài toán phát hiện gian lận

5.1.2 Hạn chế

Hạn chế khi sử dụng IP-based Queue

Trong dự án này, chúng tôi sử dụng Celery Executor với IP-based queue thay vì Redis queue. Điều này dẫn đến một số hạn chế:

1. Về thời gian và hiệu suất:

- **Network latency:** Mỗi lần gửi task đến worker phải qua network, phụ thuộc vào tốc độ mạng giữa các máy. Nếu mạng chậm hoặc không ổn định, thời gian xử lý sẽ tăng đáng kể.
- **Không có message persistence:** Nếu Airflow scheduler hoặc worker bị crash, các task đang chờ xử lý có thể bị mất, không có cơ chế retry tự động tốt.

- **Polling overhead:** Celery workers phải liên tục poll để kiểm tra task mới, gây tốn tài nguyên CPU và network bandwidth.
- **Thời gian retry:** Khi task fail, việc retry phụ thuộc vào cấu hình Airflow, không có cơ chế priority queue để ưu tiên các task quan trọng.

2. Về khả năng và tính năng:

- **Khó scale:** Khi muốn thêm worker mới, phải cấu hình lại IP và queue trong Airflow config, không linh hoạt như Redis queue.
- **Không có priority queue:** Tất cả tasks được xử lý theo thứ tự FIFO, không thể ưu tiên các task quan trọng hơn.
- **Khó monitor:** Việc theo dõi số lượng task đang chờ, đang xử lý khó khăn hơn so với Redis queue có các công cụ monitoring tích hợp.
- **Không có rate limiting:** Không thể giới hạn số lượng task được xử lý đồng thời trên mỗi worker một cách dễ dàng.

3. Bất tiện khi dùng IP máy:

- **Hardcode IP addresses:** Phải hardcode IP của các máy trong cấu hình Airflow và code, khi thay đổi IP hoặc thêm máy mới phải sửa nhiều nơi.
- **Khó maintain:** Khi infrastructure thay đổi (thêm/xóa máy, thay đổi IP), phải cập nhật cấu hình ở nhiều file khác nhau.
- **Không có service discovery:** Phải biết trước IP của tất cả workers, không có cơ chế tự động phát hiện workers mới.
- **Network dependency:** Phụ thuộc hoàn toàn vào network giữa các máy, nếu có vấn đề về network thì toàn bộ hệ thống bị ảnh hưởng.
- **Security concerns:** Phải đảm bảo các máy có thể giao tiếp với nhau qua network, có thể gây lo ngại về bảo mật.

Điểm mạnh của Redis Queue nếu được sử dụng

Nếu hệ thống được triển khai với Redis queue thay vì IP-based queue, sẽ có những lợi ích sau:

1. Hiệu suất và độ tin cậy:

- **Message persistence:** Redis có thể cấu hình persistence, đảm bảo tasks không bị mất khi hệ thống restart.
- **Lower latency:** Redis là in-memory database, tốc độ xử lý nhanh hơn so với network communication trực tiếp.
- **Atomic operations:** Redis hỗ trợ atomic operations, đảm bảo tính nhất quán khi nhiều workers cùng xử lý tasks.

- **Better retry mechanism:** Có thể cấu hình retry với exponential backoff dễ dàng hơn.

2. Tính năng và khả năng mở rộng:

- **Priority queues:** Redis hỗ trợ sorted sets, có thể implement priority queue để ưu tiên các task quan trọng.
- **Rate limiting:** Dễ dàng implement rate limiting để kiểm soát throughput.
- **Better monitoring:** Redis có các công cụ monitoring như Redis Insight, dễ dàng theo dõi queue size, throughput, etc.
- **Easy scaling:** Thêm worker mới chỉ cần kết nối đến Redis, không cần cấu hình lại Airflow.
- **Service discovery:** Có thể sử dụng Redis để implement service discovery pattern.

3. Bảo trì và quản lý:

- **Centralized configuration:** Tất cả workers kết nối đến một Redis instance, dễ quản lý hơn.
- **No IP dependency:** Workers chỉ cần biết địa chỉ Redis, không cần biết IP của các workers khác.
- **Flexible deployment:** Có thể deploy Redis trên cloud hoặc on-premise, dễ dàng migrate.
- **Better debugging:** Redis CLI và monitoring tools giúp debug dễ dàng hơn.

Các hạn chế khác

- **Class imbalance:** Dataset có tỷ lệ fraud rất thấp (0.172%), có thể ảnh hưởng đến hiệu suất mô hình
- **Feature engineering hạn chế:** Do dataset đã được PCA encode, không thể thực hiện feature engineering chi tiết
- **Resource constraints:** Hệ thống chạy trên hardware hạn chế, có thể ảnh hưởng đến performance khi scale up
- **Single point of failure:** Một số components như NameNode có thể trở thành single point of failure nếu không cấu hình HA

5.2 So sánh với các phương pháp khác

5.2.1 So sánh với batch processing

Streaming processing có ưu điểm:

- **Real-time:** Phát hiện gian lận ngay lập tức thay vì phải đợi batch processing
- **Lower latency:** Kết quả prediction có sẵn trong vài giây thay vì vài phút hoặc vài giờ
- **Better user experience:** Có thể cảnh báo người dùng ngay lập tức về giao dịch đáng ngờ

5.2.2 So sánh với các thuật toán ML khác

Random Forest được chọn vì:

- **Interpretability:** Có thể xem feature importance để hiểu mô hình
- **Robustness:** Ít bị overfitting và xử lý tốt với noisy data
- **Parallelization:** Spark ML Random Forest có thể train song song nhiều trees
- **Performance:** Đạt kết quả tốt trên dataset này với ROC-AUC = 0.98

Chương 6

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Dự án đã thành công xây dựng một hệ thống phát hiện gian lận thẻ tín dụng thời gian thực sử dụng Spark ML, Kafka và Airflow. Hệ thống đạt được các mục tiêu đề ra:

- Huấn luyện thành công mô hình Random Forest với Spark ML đạt ROC-AUC = 0.98
- Triển khai pipeline streaming thời gian thực với Kafka
- Tự động hóa toàn bộ quy trình bằng Apache Airflow
- Hệ thống hoạt động ổn định trên môi trường phân tán với 3 máy chủ

Tuy nhiên, hệ thống cũng có một số hạn chế, đặc biệt là việc sử dụng IP-based queue thay vì Redis queue, gây khó khăn trong việc scale và maintain hệ thống.

6.2 Hướng phát triển

6.2.1 Cải thiện ngắn hạn

- **Migrate sang Redis queue:** Thay thế IP-based queue bằng Redis queue để cải thiện hiệu suất và khả năng mở rộng
- **Implement model versioning:** Quản lý nhiều phiên bản model để có thể rollback khi cần
- **Add monitoring và alerting:** Tích hợp Prometheus và Grafana để monitor hệ thống
- **Improve error handling:** Thêm retry mechanism và dead letter queue cho failed tasks

6.2.2 Cải thiện dài hạn

- **Model retraining tự động:** Tự động retrain model định kỳ với dữ liệu mới
- **A/B testing:** So sánh hiệu suất của các mô hình khác nhau
- **Feature store:** Xây dựng feature store để quản lý và versioning features
- **Real-time visualization:** Xây dựng dashboard để visualize kết quả prediction real-time
- **Multi-model ensemble:** Kết hợp nhiều mô hình để cải thiện độ chính xác
- **Explainability:** Thêm tính năng giải thích tại sao một giao dịch bị đánh dấu là gian lận

6.2.3 Cải thiện infrastructure

- **Kubernetes deployment:** Triển khai hệ thống trên Kubernetes để dễ dàng scale và manage
- **High Availability:** Cấu hình HA cho các components quan trọng như NameNode, Kafka
- **Cloud migration:** Migrate lên cloud (AWS, GCP, Azure) để tận dụng các managed services
- **Containerization:** Containerize tất cả applications để dễ dàng deploy và maintain

Tài liệu tham khảo

- [1] Apache Spark. *Apache Spark - Unified Analytics Engine for Big Data.* <https://spark.apache.org/>
- [2] Apache Kafka. *Apache Kafka - Open-source distributed event streaming platform.* <https://kafka.apache.org/>
- [3] Apache Airflow. *Apache Airflow - A platform to programmatically author, schedule and monitor workflows.* <https://airflow.apache.org/>
- [4] Apache Hadoop. *Apache Hadoop - Open-source software for reliable, scalable, distributed computing.* <https://hadoop.apache.org/>
- [5] Kaggle. *Credit Card Fraud Detection Dataset.* <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>
- [6] Meng, X., et al. (2016). *MLlib: Machine Learning in Apache Spark.* Journal of Machine Learning Research, 17(1), 1235-1241.
- [7] Breiman, L. (2001). *Random Forests.* Machine Learning, 45(1), 5-32.
- [8] Zaharia, M., et al. (2013). *Discretized Streams: Fault-tolerant Streaming Computation at Scale.* Proceedings of SOSP.
- [9] Celery. *Distributed Task Queue.* <https://docs.celeryproject.org/>
- [10] Redis. *Redis - In-memory data structure store.* <https://redis.io/>

Phụ lục A

CODE SNIPPETS

A.1 Training Script

Listing A.1: Code training model với Spark ML

```

1 from pyspark.sql import SparkSession
2 from pyspark.ml.feature import VectorAssembler
3 from pyspark.ml.classification import RandomForestClassifier
4 from pyspark.ml.evaluation import BinaryClassificationEvaluator
5
6 # Create Spark session
7 spark = SparkSession.builder \
8     .appName("FraudDetectionTrainingSpark") \
9     .getOrCreate()
10
11 # Load data from HDFS
12 data_path = "hdfs://192.168.80.52:9000/data/train.csv"
13 df_spark = spark.read.csv(data_path, header=True, inferSchema=True)
14
15 # Prepare features
16 feature_cols = [c for c in df_spark.columns if c != "Class"]
17 assembler = VectorAssembler(
18     inputCols=feature_cols,
19     outputCol="features"
20 )
21 df_assembled = assembler.transform(df_spark)
22 df_final = df_assembled.select("features", "Class") \
23     .withColumnRenamed("Class", "label")
24
25 # Train/Test split
26 train_df, test_df = df_final.randomSplit([0.8, 0.2], seed=42)
27
28 # Train Random Forest
29 rf = RandomForestClassifier(
30     numTrees=300,
31     maxDepth=15,
32     labelCol="label",
33     featuresCol="features",
34     seed=42
35 )
36 model = rf.fit(train_df)
37
38 # Evaluate
39 evaluator = BinaryClassificationEvaluator(
40     labelCol="label",
41     rawPredictionCol="rawPrediction",
42     metricName="areaUnderROC"
43 )
44 roc_auc = evaluator.evaluate(model.transform(test_df))
45
46 # Save model
47 model_output_path = "hdfs://192.168.80.52:9000/model"
48 model.write().overwrite().save(model_output_path)

```

A.2 Prediction Script

Listing A.2: Code prediction với Spark Streaming

```

1 from pyspark.sql import SparkSession
2 from pyspark.sql import functions as F
3 from pyspark.ml.classification import RandomForestClassificationModel
4
5 # Create Spark session
6 spark = SparkSession.builder \
7     .appName("FraudPredictionStreaming") \
8     .config("spark.cores.max", "12") \
9     .getOrCreate()
10
11 # Load model
12 MODEL_PATH = "hdfs://192.168.80.52:9000/model"
13 model = RandomForestClassificationModel.load(MODEL_PATH)
14
15 # Read from Kafka
16 raw_stream = spark.readStream.format("kafka") \
17     .option("kafka.bootstrap.servers", "192.168.80.122:9092") \
18     .option("subscribe", "input") \
19     .option("startingOffsets", "latest") \
20     .load()
21
22 # Parse JSON and prepare features
23 # ... (feature preparation code)
24
25 # Predict
26 predictions = model.transform(assembled_df)
27
28 # Write to Kafka
29 query = predictions.select(
30     F.to_json(F.struct("transaction_id", "fraud_prediction",
31                         "fraud_score")).alias("value")
32 ).writeStream.format("kafka") \
33     .option("kafka.bootstrap.servers", "192.168.80.122:9092") \
34     .option("topic", "output") \
35     .option("checkpointLocation",
36             "hdfs://192.168.80.52:9000/checkpoints/fraud_prediction") \
37     .start()
38
39 query.awaitTermination()

```

A.3 Airflow DAG Configuration

Listing A.3: Câu hình Airflow DAG

```
1 from airflow import DAG
2 from airflow.providers.standard.operators.python import PythonOperator
3 from datetime import datetime
4
5 FULL_PIPELINE_CONFIG = {
6     'hadoop_host': '192.168.80.52',
7     'spark_master_url': 'spark://192.168.80.52:7077',
8     'kafka_bootstrap': '192.168.80.122:9092',
9     'train_input': 'hdfs://192.168.80.52:9000/data/train.csv',
10    'model_path': 'hdfs://192.168.80.52:9000/model',
11}
12
13 with DAG(
14     dag_id='bigdata_full_pipeline',
15     start_date=datetime(2024, 1, 1),
16     schedule=None,
17 ) as dag:
18
19     task_start_hadoop = PythonOperator(
20         task_id='start_hadoop',
21         python_callable=start_hadoop,
22     )
23
24     # ... other tasks
25
26     # Dependencies
27     task_start_hadoop >> task_start_spark_master >> \
28     [task_start_spark_worker_1, task_start_spark_worker_2] >> \
29     task_start_kafka
```