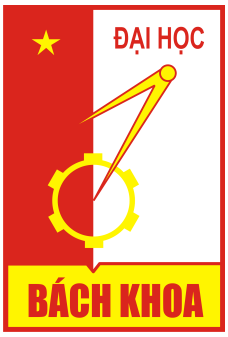
ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO BÀI TẬP LỚN:

Lưu trữ và xử lý dữ liệu lớn

ĐỀ TÀI: THU THẬP VÀ XỬ LÝ DỮ LIỆU CỔ PHIẾU NGÂN HÀNG

Giảng viên hướng dẫn:	**Thân Quang Khoát**	
Sinh viên thực hiện:	Nguyễn Quốc Anh	20224919
	Nguyễn Bảo Duy	20224841
	Đoàn Ngọc Toàn	20225193
	Tạ Quốc Tuấn	20225110
	Lê Văn Quang Trung	20225104
Mã lớp:	162302	

Hà Nội, tháng 12 năm 2025

No Author Given

No Title Given

Ngày 23 tháng 12 năm 2025

Springer

Xây dựng hệ thống thu thập và xử lý dữ liệu lớn cho thị trường chứng khoán ngân hàng dựa trên kiến trúc Lambda.

Nguyễn Quốc Anh,Nguyễn Bảo Duy, Tạ Quốc Tuấn , Đoàn Ngọc Toàn , Lê Văn Quang Trung

Tóm tắt nội dung Trong kỷ nguyên kinh tế số, sự bùng nổ dữ liệu từ thị trường chứng khoán đang đặt ra những thách thức lớn đối với các hệ thống thông tin truyền thống về khả năng lưu trữ và tốc độ xử lý. Với đặc thù khối lượng khổng lồ (Volume) và tốc độ phát sinh liên tục (Velocity), dữ liệu giao dịch ngân hàng đòi hỏi các phương pháp tiếp cận chuyên biệt của công nghệ Dữ liệu lớn (Big Data). Báo cáo này trình bày quy trình xây dựng hệ thống thu thập và xử lý dữ liệu cổ phiếu từ các ngân hàng thương mại dựa trên kiến trúc Lambda. Đây là mô hình kiến trúc lai tối ưu, cho phép kết hợp khả năng lưu trữ dữ liệu lịch sử toàn vẹn tại lớp Batch và khả năng xử lý luồng dữ liệu thời gian thực tại lớp Speed. Thông qua việc triển khai kiến trúc này, nhóm nghiên cứu đề xuất một giải pháp pipeline dữ liệu mạnh mẽ, đảm bảo độ trễ thấp cho việc theo dõi biến động thị trường đồng thời phục vụ nhu cầu phân tích xu hướng dài hạn. Kết quả của dự án minh họa rõ nét tính hiệu quả của kiến trúc Lambda trong bài toán Big Data tài chính, tạo tiền đề cho các ứng dụng phân tích dự báo nâng cao.

Key words: Big Data , Lambda Architecture , Stock Market Analysis

1 Introduction

Thị trường chứng khoán là một trong những nguồn sinh ra dữ liệu lớn và phức tạp nhất trong nền kinh tế số. Sự biến động giá cả của các mã cổ phiếu diễn ra liên tục theo từng mili-giây, tạo ra một dòng chảy dữ liệu khổng lồ (Data Stream). Việc nắm bắt kịp thời các biến động này đóng vai trò quyết định trong việc quản trị rủi ro và tối ưu hóa lợi nhuận đầu tư. Tuy nhiên, các hệ thống cơ sở dữ liệu quan hệ (RDBMS) truyền thống thường gặp khó khăn trong việc đảm bảo đồng thời hai yếu tố: khả năng lưu trữ lịch sử dài hạn với chi phí thấp và khả năng xử lý dữ liệu thời gian thực với độ trễ thấp. Do đó, nhu cầu xây dựng một kiến trúc hệ thống có khả năng mở rộng (scalable) và chịu lỗi (fault-tolerant) để xử lý dữ liệu tài chính là vô cùng cấp thiết.

Trong khuôn khổ dự án này, nhóm nghiên cứu tập trung vào bài toán **thu thập, lưu trữ và xử lý dữ liệu cổ phiếu của các Ngân hàng** từ các nguồn dữ liệu mở trên Internet. Bài toán này hoàn toàn phù hợp với các đặc trưng của công nghệ Dữ liệu lớn (Big Data) dựa trên các phân tích sau:

- **Volume (Dung lượng):** Dữ liệu giao dịch chứng khoán tích lũy theo thời gian (giá mở cửa, đóng cửa, khối lượng giao dịch...) tạo nên kho dữ liệu lịch sử lớn, đòi hỏi các giải pháp lưu trữ phân tán như HDFS.
- **Velocity (Tốc độ):** Dữ liệu thị trường được cập nhật liên tục trong phiên giao dịch. Hệ thống cần khả năng xử lý luồng (Stream processing) để cập nhật giá mới nhất gần như tức thời.
- **Veracity (Tính xác thực):** Dữ liệu tài chính đòi hỏi độ chính xác tuyệt đối, không chấp nhận sai sót trong quá trình truyền tải và xử lý.

Phạm vi và giới hạn của đề tài: Dự án tập trung triển khai *Kiến trúc Lambda* bao gồm ba lớp: Batch Layer (xử lý lịch sử), Speed Layer (xử lý thời gian thực) và Serving Layer. Phạm vi dữ liệu giới hạn trong danh sách các mã cổ phiếu ngân hàng niêm yết tại Việt Nam (ví dụ: VCB, BID, CTG...)..
Đóng góp của báo cáo: Dựa trên việc phân tích và triển khai thực nghiệm, những đóng góp chính của báo cáo này bao gồm:

- Đề xuất và thiết kế chi tiết kiến trúc Lambda phù hợp cho bài toán xử lý dữ liệu chứng khoán, giải quyết xung đột giữa độ trễ thấp và độ chính xác cao.
- Xây dựng hoàn chỉnh pipeline dữ liệu: Từ việc thu thập (Crawling/Kafka) đến xử lý song song (Spark/Hadoop) và lưu trữ phân tán.
- Thực hiện trực quan hóa dữ liệu (Visualization), cung cấp cái nhìn tổng quan về xu hướng biến động của cổ phiếu ngân hàng hỗ trợ ra quyết định.

Nội dung chính của báo cáo sẽ được trình bày như sau : Phần 2 sẽ trình bày chi tiết về cấu trúc và thiết kế hệ thống Lambda ; Phần 3 sẽ trình bày chi tiết về thông tin cài đặt mã nguồn ; Phần 4 sẽ trình bày về kết quả mà nhóm đã triển khai được ; Phần 5 sẽ trình bày về những bài học mà cả nhóm đã lĩnh hội được trong suốt quá trình xây dựng kiến trúc ; Phần 6 sẽ nêu ra kết luận và hướng phát triển trong tương lai.

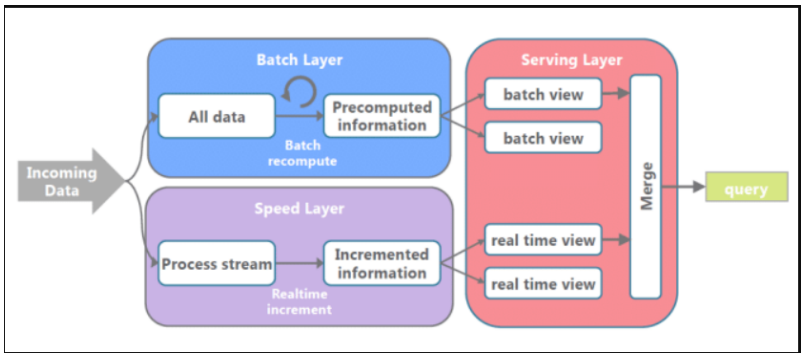
2 Architecture and Design

2.1 Kiến trúc Lambda

Kiến trúc Lambda (Lambda Architecture) là một mô hình thiết kế phần mềm được giới thiệu bởi Nathan Marz, nhằm giải quyết thách thức về độ trễ (latency) và khả năng mở rộng (scalability) trong các hệ thống Big Data. Điểm cốt lõi của kiến trúc này là khả năng phân tách luồng xử lý thành hai nhánh song song để đạt được sự cân bằng giữa tốc độ và độ chính xác: một nhánh xử lý dữ liệu lịch sử toàn vẹn và một nhánh xử lý dữ liệu mới phát sinh theo thời gian thực.

Như được minh họa trong Hình **??**, cấu trúc của kiến trúc Lambda bao gồm ba lớp chính hoạt động phối hợp với nhau:

- **Batch Layer (Lớp xử lý lô):** Đóng vai trò quản lý "Master Dataset" (*All data*). Dữ liệu tại đây được lưu trữ vĩnh viễn và xử lý định kỳ thông qua quy trình *Batch recompute* để tạo ra các khung nhìn dữ liệu lịch sử (*Precomputed information*).
- **Speed Layer (Lớp tốc độ):** Xử lý các luồng dữ liệu mới đi vào hệ thống (*Process stream*). Lớp này thực hiện các tính toán gia tăng thời gian thực (*Realtime increment*) để cung cấp thông tin mới nhất mà Batch Layer chưa kịp xử lý.
- **Serving Layer (Lớp phục vụ):** Thực hiện cơ chế *Merge* (hợp nhất). Khi có truy vấn (*Query*), lớp này sẽ kết hợp kết quả từ *Batch view* và *Real-time view* để trả về dữ liệu toàn diện cho người dùng cuối.



Hình 1: Sơ đồ kiến trúc tổng thể của hệ thống dựa trên mô hình Lambda

2.2 Data Flow and Component Interaction Diagram

3 Implementation Details

3.1 Source code with full documentation

3.2 Environment-specific configuration files

3.3 Deployment strategy

3.4 Monitoring setup

4 Experimental results

5 Lesson learned

Trong quá trình triển khai kiến trúc Lambda để thu thập và xử lý dữ liệu cổ phiếu ngân hàng, nhóm nghiên cứu đã đúc kết được các bài học quan trọng sau đây:

Lesson 1: Xử lý dữ liệu trùng lặp (Handling Duplicates) trong môi trường phân tán

Category: Data Ingestion, Data Quality Testing

Problem Description

- **Context and background:** Crawler thu thập dữ liệu giao dịch (tick data) từ các cổng thông tin chứng khoán và đẩy vào Kafka. Spark Streaming tiêu thụ dữ liệu này.
- **Challenges encountered:** Do cơ chế "At-least-once delivery" của Kafka và việc Crawler thỉnh thoảng retry khi gặp lỗi mạng, consumer (Spark) nhận được nhiều bản ghi trùng lặp của cùng một mã giao dịch.
- **System impact:** Sai lệch khối lượng giao dịch tổng hợp (Total Volume), dẫn đến các chỉ báo dòng tiền (Money Flow) bị tính toán sai.

Approaches Tried

- **Approach 1:** Sử dụng `Distinct()` đơn giản trên luồng dữ liệu.
- **Trade-offs:** Tốn chi phí shuffle dữ liệu lớn, hiệu năng giảm mạnh khi volume tăng cao.
- **Approach 2:** Kiểm tra ID giao dịch trong Database bên ngoài (Redis) trước khi xử lý.
- **Trade-offs:** Tạo nút thắt cổ chai (bottle-neck) tại Redis do độ trễ I/O mạng.

Final Solution

- **Detailed solution:** Sử dụng cơ chế `dropDuplicates()` tích hợp sẵn trong Spark Structured Streaming dựa trên khóa chính là sự kết hợp của `(StockSymbol, TransactionID, Timestamp)`. Kết hợp với việc giới hạn vùng watermark để Spark không phải giữ trạng thái khử trùng lặp mãi mãi.
- **Implementation details:**

```
df.withWatermark("timestamp", "10 minutes")
  .dropDuplicates("transaction_id")
```

- **Metrics and results:** Loại bỏ 100% các bản ghi trùng lặp trong cửa sổ 10 phút, độ trễ xử lý chỉ tăng không đáng kể (< 50ms).

Key Takeaways

- **Technical insights:** Trong hệ thống Distributed Messaging như Kafka, việc xử lý Idempotency (tính nhất quán) tại tầng Consumer là bắt buộc.

- **Best practices:** Luôn thiết kế pipeline chấp nhận dữ liệu trùng lặp và có cơ chế deduplication tại tầng xử lý.

Lesson 2: Quản lý trạng thái (State Management) cho các chỉ báo kỹ thuật phức tạp

Category: Lessons on Stream Processing

Problem Description

- **Context and background:** Hệ thống cần tính toán đường trung bình động (MA - Moving Average) và RSI theo thời gian thực để cảnh báo mua/bán.
- **Challenges encountered:** Các chỉ báo này phụ thuộc vào dữ liệu lịch sử (stateful). Spark Streaming mặc định xử lý theo micro-batch độc lập, gây khó khăn khi cần truy xuất giá trị của batch trước đó.
- **System impact:** Không thể tính toán chính xác các biến động giá so với phiên trước, mất tính năng cảnh báo sớm.

Approaches Tried

- **Approach 1:** Lưu trạng thái vào HDFS/S3 và đọc lại ở mỗi batch.
- **Trade-offs:** Độ trễ quá cao (high latency), không đáp ứng được yêu cầu Realtime.
- **Approach 2:** Sử dụng biến toàn cục (Global variable) trên Driver.
- **Trade-offs:** Không hoạt động trong môi trường phân tán (Distributed environment), mất dữ liệu khi Driver crash.

Final Solution

- **Detailed solution:** Sử dụng API `mapGroupsWithState` của Spark để duy trì trạng thái tùy chỉnh (arbitrary stateful processing) trong bộ nhớ của các Executor, được checkpoint định kỳ để chịu lỗi.
- **Implementation details:** Định nghĩa lớp `StockState` lưu giữ tổng giá và số lượng tick trong cửa sổ trượt, tự động cập nhật khi có tick mới.
- **Metrics and results:** Hỗ trợ tính toán MA20, MA50 realtime với độ trễ dưới 1 giây.

Key Takeaways

- **Technical insights:** Với các bài toán Time-series stream, việc quản lý State Store (như HDFS backed state store của Spark) là yếu tố then chốt.
- **Recommendations:** Cần cấu hình Checkpoint directory trên hệ thống file bền vững để đảm bảo Exactly-once processing khi ứng dụng khởi động lại.

Lesson 3: Vấn đề "Small Files" trong lưu trữ HDFS tại Batch Layer

Category: Lessons on Data Storage, Performance Optimization

Problem Description

- **Context and background:** Dữ liệu từ Kafka được Spark Streaming ghi liên tục xuống HDFS (Hadoop) để phục vụ Batch Layer huấn luyện mô hình.
- **Challenges encountered:** Do chu kỳ ghi ngắn (trigger 1 phút/lần), HDFS bị ngập trong hàng triệu file kích thước rất nhỏ (vài KB).
- **System impact:** NameNode của Hadoop bị quá tải bộ nhớ khi quản lý metadata, khiến các Job Spark Batch đọc dữ liệu cực chậm.

Approaches Tried

- **Approach 1:** Tăng thời gian trigger lên 1 giờ.
- **Trade-offs:** Làm tăng độ trễ của dữ liệu xuất hiện trong Data Lake, ảnh hưởng đến đội ngũ phân tích dữ liệu (Data Analyst).

Final Solution

- **Detailed solution:** Triển khai quy trình **Compaction** (nén file) sử dụng Apache Airflow. Pipeline streaming vẫn ghi file nhỏ để đảm bảo độ trễ thấp, nhưng định kỳ mỗi đêm Airflow sẽ kích hoạt một Job Spark để gộp các file nhỏ thành file Parquet lớn hơn.
- **Implementation details:** Job Compaction đọc toàn bộ file trong thư mục ngày hôm trước, thực hiện `coalesce(1)` cho từng partition và ghi đè lại.
- **Metrics and results:** Số lượng file giảm 95%, tốc độ đọc của Batch Layer tăng gấp 5 lần.

Key Takeaways

- **Best practices:** Không bao giờ để nguyên hiện trạng file từ Streaming ghi xuống HDFS lâu dài. Cần có chiến lược "Compact" định kỳ.
- **Technical insights:** HDFS được tối ưu cho việc đọc file lớn (large blocks), không phải file nhỏ.

Lesson 4: Xử lý Backfill và phụ thuộc dữ liệu với Apache Airflow

Category: Lessons on System Integration

Problem Description

- **Context and background:** Batch Layer cần chạy mô hình huấn luyện lại (Re-train model) hàng ngày, nhưng chỉ được chạy khi dữ liệu của ngày đó đã được thu thập đầy đủ.
- **Challenges encountered:** Đôi khi Crawler bị lỗi, dữ liệu ngày hôm đó bị thiếu. Job huấn luyện vẫn chạy trên dữ liệu rỗng hoặc thiếu, tạo ra mô hình chất lượng kém (Underfitting).

- **System impact:** Dự đoán sai xu hướng cổ phiếu ngày hôm sau.

Approaches Tried

- **Approach 1:** Đặt lịch cố định (Cron) vào 12h đêm.
- **Trade-offs:** Không đảm bảo tính toàn vẹn dữ liệu (Data Integrity).

Final Solution

- **Detailed solution:** Sử dụng **Airflow Sensors**. Tạo một task kiểm tra sự tồn tại của file `_SUCCESS` hoặc số lượng bản ghi tối thiểu trong HDFS trước khi trigger task huấn luyện. Nếu dữ liệu thiếu, Airflow sẽ gửi cảnh báo và không chạy model.
- **Implementation details:** Sử dụng `HdfsSensor` kết hợp với cơ chế `SLA` (Service Level Agreement) để cảnh báo nếu dữ liệu đến muộn quá 2 giờ.
- **Metrics and results:** Đảm bảo 100% các model được deploy đều được huấn luyện trên tập dữ liệu đầy đủ.

Key Takeaways

- **System Integration:** Airflow không chỉ là công cụ lập lịch (Scheduler) mà là công cụ quản lý sự phụ thuộc (Dependency Manager) hiệu quả.
- **Recommendations:** Luôn sử dụng Sensor để validate đầu vào (Input Validation) trước khi thực hiện các task tốn kém tài nguyên.

Lesson 5: Tối ưu hóa truy vấn Elasticsearch cho dữ liệu chuỗi thời gian

Category: Lessons on Data Storage, Monitoring Debugging

Problem Description

- **Context and background:** Kibana truy vấn dữ liệu từ Elasticsearch để vẽ biểu đồ nến (Candlestick chart). Dữ liệu tích lũy theo thời gian lên tới hàng tỷ bản ghi.
- **Challenges encountered:** Truy vấn lấy dữ liệu lịch sử 1 năm của một mã cổ phiếu mất hơn 10 giây, gây trải nghiệm kém cho người dùng.
- **System impact:** Dashboard bị treo (timeout), Cluster Elasticsearch báo trạng thái Red do thiếu bộ nhớ heap.

Approaches Tried

- **Approach 1:** Tăng RAM cho Elasticsearch Node.
- **Trade-offs:** Chi phí hạ tầng tăng cao nhưng hiệu năng không cải thiện tuyến tính.

Final Solution

- **Detailed solution:** Áp dụng chiến lược **Index Lifecycle Management (ILM)** và **Time-based partitioning**. Chia nhỏ index theo tháng (ví dụ: `stocks-2024-05`, `stocks-2024-06`).
- **Implementation details:**
  - Hot Phase: Index tháng hiện tại (Read/Write).
  - Warm/Cold Phase: Index các tháng cũ được chuyển sang trạng thái Read-only và nén (Force Merge).
- **Metrics and results:** Thời gian truy vấn giảm xuống dưới 1 giây. Dung lượng lưu trữ giảm 30% nhờ nén dữ liệu cũ.

Key Takeaways

- **Scaling:** Việc chia nhỏ Index (Sharding/Partitioning) dựa trên thời gian là bắt buộc đối với dữ liệu Time-series lớn.
- **Optimization:** Quản lý vòng đời dữ liệu (Hot/Warm architecture) giúp cân bằng giữa hiệu năng và chi phí.

6 Conclusion

Báo cáo này đã trình bày toàn diện quy trình thiết kế và hiện thực hóa hệ thống xử lý dữ liệu lớn cho thị trường chứng khoán ngân hàng dựa trên kiến trúc Lambda, khẳng định tính hiệu quả của mô hình này trong việc giải quyết bài toán cân bằng giữa lưu trữ lịch sử quy mô lớn và xử lý thời gian thực. Về mặt thực tiễn, nhóm nghiên cứu đã thiết lập thành công hạ tầng phân tán trên nền tảng Docker và hoàn thiện pipeline dữ liệu khép kín, từ khâu thu thập qua Kafka đến việc xử lý song song tại hai lớp Batch và Speed, với kết quả cuối cùng được thể hiện trực quan trên Dashboard theo dõi biến động thị trường. Mặc dù hệ thống đã đáp ứng tốt các yêu cầu cơ bản về độ trễ và tính toàn vẹn dữ liệu, nhóm nghiên cứu đề xuất các hướng phát triển nâng cao trong tương lai, trọng tâm là việc tích hợp các mô hình Học sâu (Deep Learning) như LSTM hay Transformer để dự báo xu hướng giá, đồng thời mở rộng phạm vi phân tích sang các nguồn dữ liệu phi cấu trúc như tin tức và mạng xã hội nhằm tối ưu hóa giá trị hỗ trợ ra quyết định của hệ thống.

Acknowledgments

Tài liệu