

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**



**BÁO CÁO BÀI TẬP LỚN
HỆ ĐIỀU HÀNH MỞ RỘNG (CO201D)**

**THUẬT TOÁN XỬ LÝ DATA STREAMING VỚI KAFKA, SPARK
VÀ DỰ ĐOÁN XU HƯỚNG GIÁ CHỨNG KHOÁN TRÊN THỊ TRƯỜNG**

GV hướng dẫn: TS. Nguyễn Quang Hùng
SV thực hiện: Huỳnh Tấn Lộc – 2010391
Lưu Quốc Hưng Thịnh – 2010651
Nguyễn Quang Khánh – 2010330

Thành phố Hồ Chí Minh, Tháng 5 năm 2022

Mục lục

1	Tổng quan về đề tài	2
2	Kafka	2
2.1	Event streaming là gì?	2
2.2	Apache Kafka là gì?	3
2.3	Apache Kafka hoạt động như thế nào?	3
2.4	Các API cốt lõi của Kafka	4
3	Spark	5
3.1	Spark là gì	5
3.2	Các thành phần của Spark	5
3.3	Các đặc tính của Spark	6
3.4	Kiến trúc của Spark	7
4	Thuật toán xử lý dữ liệu dòng:	7
5	Mô hình LSTM	9
5.1	Khái niệm và ý tưởng của LSTM	9
5.2	Mô hình LSTM	9
5.3	Ứng dụng của LSTM	10
6	Huấn luyện mô hình dự đoán xu hướng của chứng khoán	11
7	Demo quá trình truyền dữ liệu và dự đoán	12
7.1	Truyền dữ liệu	12
7.2	Dự đoán	13
8	So sánh mô hình LSTM và mô hình RNN cho dự đoán giá chứng khoán	14
9	Hạn chế của mô hình và những định hướng tiếp theo	15
	Tài liệu tham khảo	17

1 Tổng quan về đề tài

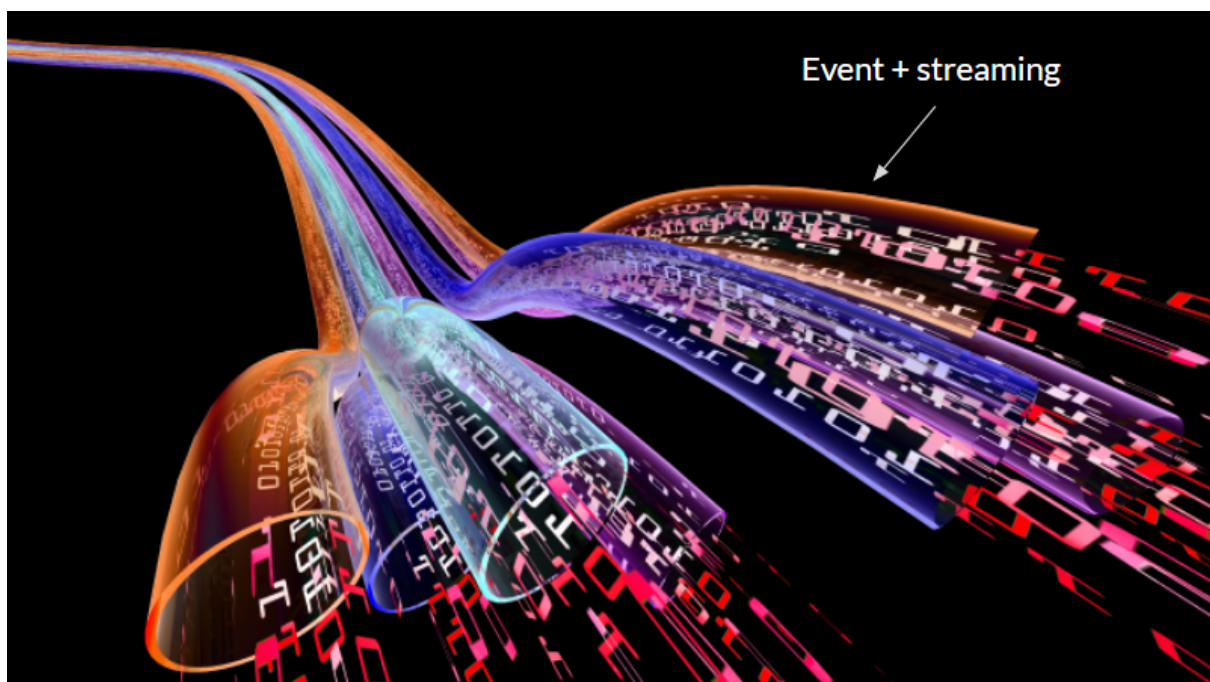
Hiện nay, **dữ liệu lớn** (Big data) là một trong những chủ đề đang rất nóng hiện nay. Hiện tại, các bài toán real-time system cùng với dữ liệu dòng đang dần trở nên phổ biến khi mà dữ liệu đổ về các hệ thống liên tục.

Có rất nhiều công nghệ được sinh ra nhằm giải quyết vấn đề dữ liệu dòng như **Apache Spark**, **Apache Kafka**,... rất hiệu quả, đạt được những kết quả tốt. Trong khuôn khổ bài báo cáo này, nhóm sẽ thực hiện đề tài "**Thuật toán xử lý data streaming với Kafka, Spark và Dự đoán xu hướng giá chứng khoán trên thị trường**", xây dựng thuật toán xử lý dữ liệu dòng trên Kafka và Spark. Đồng thời, song song đó nhóm sẽ sử dụng mô hình Long Short-Term Memory để dự đoán các xu hướng chứng khoán hiện nay.

2 Kafka

2.1 Event streaming là gì?

Event streaming giống như một phiên bản số hoá của hệ thống thần kinh trung ương của cơ thể con người. Ở cơ thể người, hệ thống thần kinh sẽ dẫn truyền các sóng điện từ khi có kích thích dọc theo các dây thần kinh đến tủy sống. Còn ở **event streaming**, các luồng dữ liệu về sự kiện (**event**) được số hoá sẽ được truyền (**stream**) như một dòng chảy về các cơ sở dữ liệu và các nền tảng chuyên về **event streaming**. Đây là một nền tảng công nghệ phù hợp trong một thế giới vận động liên tục và ngày càng tự động hoá như hiện nay.



Hình 1: Event streaming

Có thể nói, quan trọng nhất đối với **event streaming** là khả năng nắm bắt dữ liệu trong thời gian thực (**real-time**) từ những nguồn sự kiện như cơ sở dữ liệu, cảm biến, thiết bị di động, dịch vụ đám mây... dưới dạng các dòng chảy sự kiện (**streams of events**) để lưu trữ, xử lý và điều hướng tức thời. Do đó tính liên tục của luồng thông tin sẽ được đảm bảo và dữ liệu sẽ đến được đúng nơi, đúng thời điểm.

Event streaming có thể được sử dụng cho nhiều trường hợp trong nhiều ngành công nghiệp và tổ chức như:

- Xử lý các giao dịch thanh toán và tài chính trong thời gian ở các ngân hàng.

- Theo dõi và quản lý các phương tiện vận tải (xe tải, tàu thuyền...), các đơn hàng trong ngành quản lý chuỗi cung ứng (logistic).
- Liên tục thu thập và phân tích dữ liệu cảm biến từ các thiết bị IoT tại nhà máy hoặc hộ gia đình.
- Nắm bắt và phản ứng kịp thời đến các tương tác, giao dịch của khách hàng của ngành du lịch, thương mại điện tử.

2.2 Apache Kafka là gì?

Apache Kafka là một nền tảng mã nguồn mở phân tán chuyên dành cho **event streaming** được sử dụng bởi hàng ngàn công ty cho các hệ thống dữ liệu hiệu năng cao, phân tích dòng chảy, hợp nhất dữ liệu và các ứng dụng quan trọng khác.

Apache Kafka kết hợp ba khả năng then chốt cho **event streaming**:

- Viết (**publish/write**) và đọc (**subscribe to/read**) các dòng chảy sự kiện (**streams of events**), bao gồm việc liên tục nhập/xuất dữ liệu đến các hệ thống khác.
- Lưu trữ (**store**) các dòng chảy sự kiện một cách lâu dài, bền vững.
- Xử lý (**process**) các dòng chảy sự kiện ngay khi xuất hiện hoặc sau này.



Hình 2: *Apache Kafka*

2.3 Apache Kafka hoạt động như thế nào?

Kafka là một hệ thống phân tán bao gồm các máy chủ (**servers**) và các máy khách (**clients**) giao tiếp với nhau thông qua giao thức mạng TCP hiệu năng cao.

- **Servers: Kafka** được vận hành như một cụm gồm một hoặc nhiều máy chủ giúp trải dài các trung tâm dữ liệu xuyên khắp các khu vực. Một số máy chủ có chức năng hình thành nên lớp lưu trữ, gọi là **broker**. Một số khác vận hành một công cụ gọi là **Kafka Connect** để liên tục nhập/xuất dữ liệu dưới dạng **event streams**.
- **Clients:** cho phép người dùng viết các ứng dụng, dịch vụ vi mô phân tán mà đọc, viết và xử lý các **streams of events** một cách song song, đồng thời. Cộng đồng **Kafka** cung cấp rất nhiều **client** khả dụng cho nhiều ngôn ngữ: Java và Scala (bao gồm cả thư viện **Kafka Streams** cấp cao), Python, C/C++, Go và nhiều ngôn ngữ lập trình khác.

Để hiểu rõ hơn cụ thể **Kafka** hoạt động như thế nào, cần biết một số khái niệm và thuật ngữ chính sau.

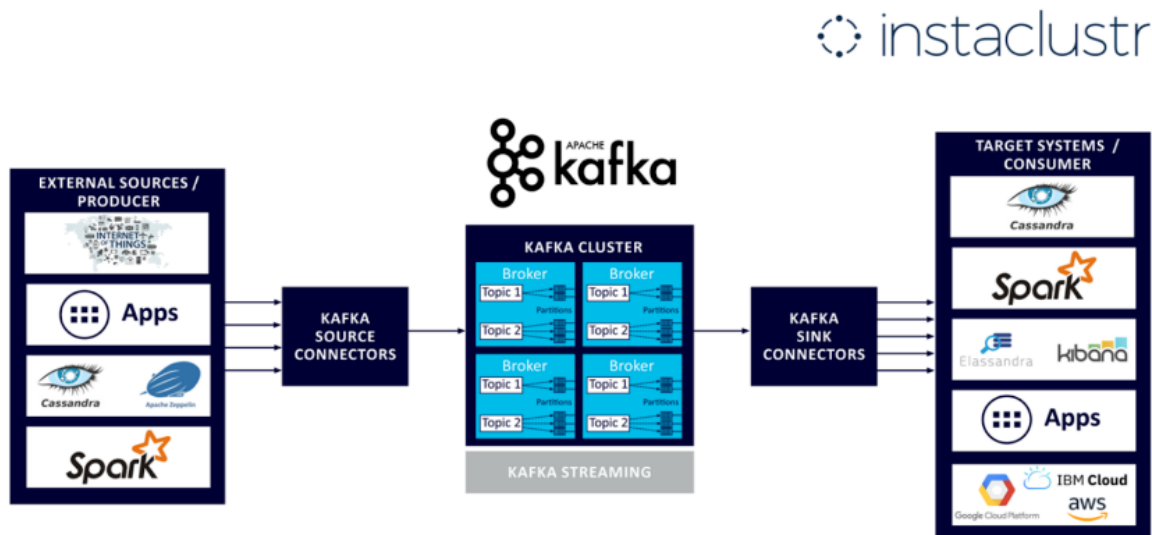
Một sự kiện (**event**) ghi lại thông tin, sự thật về một thứ gì đó đã xảy ra trong đời thực hoặc trong doanh nghiệp. Thông thường, một sự kiện gồm các trường: khoá, giá trị và mốc thời gian.

Producers là các ứng dụng phía **client** có chức năng viết các sự kiện đến **Kafka**. Ngược lại, **consumers** là các ứng dụng sử dụng (đọc và xử lý) các sự kiện này.

Các sự kiện được sắp xếp và lưu trữ trong **topic**. Nói một cách đơn giản, **topic** tương tự như một thư mục trong hệ thống tập tin, và các sự kiện là các tập tin lưu trong thư mục đó.

Các **topic** được chia ra thành các **partition**. Khi một sự kiện mới được ghi vào một **topic**, thực chất nó đã được thêm vào một trong các **partition**. Các sự kiện cùng trường khoá (chẳng hạn một khách hàng hoặc định danh của một phương tiện) được viết vào cùng một **partition**.

Kafka đảm bảo tính bền vững của dữ liệu bằng cách nhân bản **replicate** các **topic** trên các **broker** khác nhau.



Hình 3: Tổng quát về cách hoạt động của Kafka

2.4 Các API cốt lõi của Kafka

Bên cạnh việc sử dụng **command line** để quản lý các tác vụ, **Kafka** cung cấp năm **API** cốt lõi cho Java và Scala:

- **Admin API:** quản lý, giám sát các **topic**, **broker** và các vấn đề liên quan đến **Kafka** khác.
- **Producer API:** viết các **streams of events** đến các **Kafka topic**.
- **Consumer API:** đọc hoặc xử lý các **streams of events** từ các **Kafka topic**.
- **Kafka Streams API:** hiện thực các ứng dụng, dịch vụ vi mô xử lý dòng chảy với những chức năng cấp cao (chuyển đổi dữ liệu, các thao tác như hợp,...).
- **Kafka Connect API:** xây dựng và chạy các kết nối nhập/xuất dữ liệu để đọc/viết các **streams of events** đến các hệ thống và ứng dụng bên ngoài tích hợp với **Kafka**.

3 Spark

3.1 Spark là gì

Apache spark là một framework mã nguồn mở tính toán cụm được phát triển vào năm 2009 bởi AMPLap. Đến năm 2013, nó được trao lại cho Apache Software Foundation và tiếp tục phát triển cho đến ngày nay.

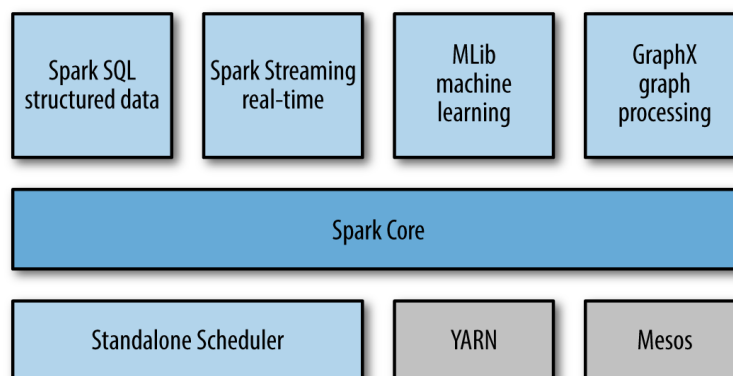


Apache Spark cho phép ta xây dựng những mô hình dự đoán nhanh chóng với khả năng thực hiện tính toán cùng lúc trên một nhóm các máy tính hay trên toàn bộ các tập dữ liệu mà không cần thiết phải trích xuất các mẫu tính toán thử nghiệm. Tốc độ xử lý dữ liệu của Apache Spark có được là do khả năng thực hiện các tính toán trên nhiều máy khác nhau cùng một lúc tại bộ nhớ trong (in-memories) hay hoàn toàn trên RAM.

Ngoài ra, đối với Apache Spark ta cũng có thể thực hiện tính toán dữ liệu thời gian thực. Điều đó có nghĩa là ta có thể vừa nhận dữ liệu từ các nguồn khác nhau, vừa thực hiện tính toán trên các dữ liệu ấy một cách đồng thời.

Spark không có hệ thống file của riêng mình, nó sử dụng hệ thống file khác như: HDFS, Cassandra, S3,... Spark hỗ trợ nhiều kiểu định dạng file khác nhau (text, csv, json...) đồng thời nó hoàn toàn không phụ thuộc vào bất cứ một hệ thống file nào.

3.2 Các thành phần của Spark

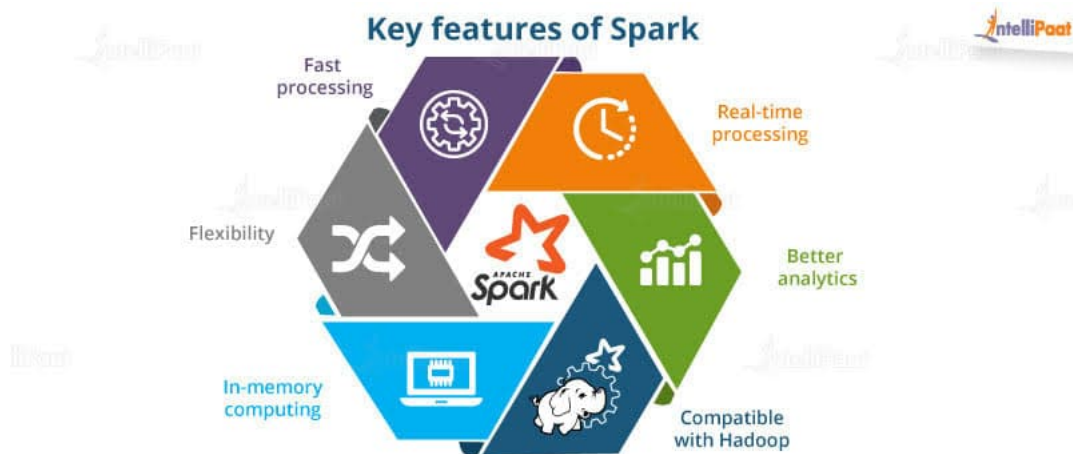


Hình 4: Các thành phần của Apache Spark

Apache Spark gồm có 5 thành phần chính: Spark Core, Spark Streaming, Spark SQL, MLlib và GraphX.

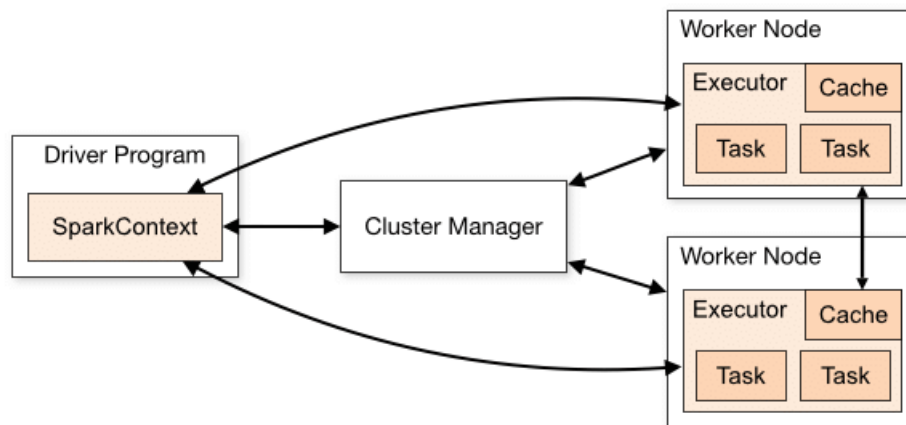
- **Spark Core** là nền tảng cho các thành phần còn lại và các thành phần này muốn khởi chạy được thì đều phải thông qua Spark Core do Spark Core đảm nhận vai trò thực hiện công việc tính toán và xử lý trong bộ nhớ, đồng thời nó cũng tham chiếu các dữ liệu được lưu trữ tại các hệ thống lưu trữ bên ngoài.
- **Spark SQL** cung cấp một kiểu data abstraction mới (SchemaRDD) nhằm hỗ trợ cho cả kiểu dữ liệu có cấu trúc (structured data) và dữ liệu nửa cấu trúc (semi-structured data – thường là dữ liệu dữ liệu có cấu trúc nhưng không đồng nhất và cấu trúc của dữ liệu phụ thuộc vào chính nội dung của dữ liệu ấy). Spark SQL hỗ trợ DSL (Domain-specific language) để thực hiện các thao tác trên DataFrames bằng ngôn ngữ Scala, Java hoặc Python và nó cũng hỗ trợ cả ngôn ngữ SQL với giao diện command-line và ODBC/JDBC server.
- **Spark Streaming** được sử dụng để thực hiện việc phân tích stream bằng việc coi stream là các mini-batches và thực hiện kỹ thuật RDD transformation đối với các dữ liệu mini-batches này. Qua đó cho phép các đoạn code được viết cho xử lý batch có thể được tận dụng lại vào trong việc xử lý stream, làm cho việc phát triển lambda architecture được dễ dàng hơn. Tuy nhiên điều này lại tạo ra độ trễ trong xử lý dữ liệu (độ trễ chính bằng mini-batch duration) và do đó nhiều chuyên gia cho rằng Spark Streaming không thực sự là công cụ xử lý streaming giống như Storm hoặc Flink.
- **MLlib (Machine Learning Library)** là một nền tảng học máy phân tán bên trên Spark do kiến trúc phân tán dựa trên bộ nhớ. Theo các so sánh benchmark Spark MLlib nhanh hơn 9 lần so với phiên bản chạy trên Hadoop (Apache Mahout).
- **Graphx** là nền tảng xử lý đồ thị dựa trên Spark.

3.3 Các đặc tính của Spark



Hình 5: Đặc tính của Apache Spark

- **Xử lý luồng thời gian thực**
- **Hỗ trợ nhiều ngôn ngữ:** Python, R, Scale, SQL.
- **Tính toán ở bộ nhớ trong:** Không giống như Hadoop MapReduce, Apache Spark có khả năng xử lý các tác vụ trong bộ nhớ và không bắt buộc phải ghi lại các kết quả trung gian vào đĩa.
- **Tích hợp với Hadoop:** Apache Spark tích hợp rất tốt với hệ thống tệp Hadoop HDFS.
- **Tốc độ:** Apache Spark có thể chạy nhanh hơn 10 lần so với Hadoop ở trên đĩa cứng và 100 lần khi chạy trên bộ nhớ RAM.



Hình 6: Kiến trúc của Spark

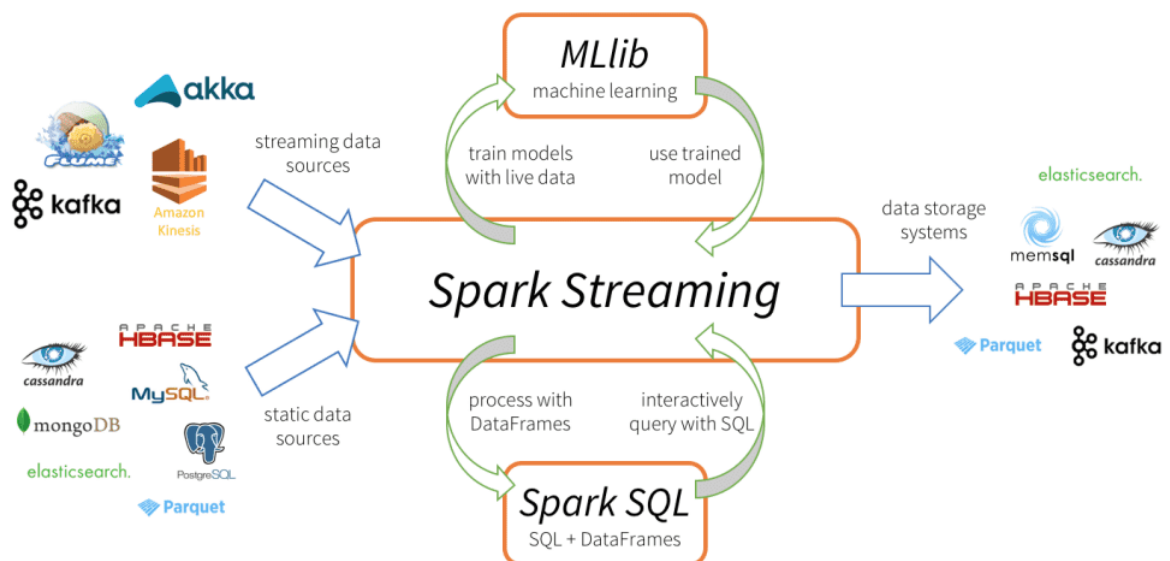
3.4 Kiến trúc của Spark

Bây giờ, chúng ta sẽ tìm hiểu sâu hơn về kiến trúc của Apache Spark. Một cách cơ bản, Apache Spark có hai thành phần chính:

- **Spark Driver:** Là trình điều khiển thực thi của một ứng dụng, dùng để chuyển đổi mã của người dùng thành nhiều tác vụ (tasks) có thể được phân phối trên các nút xử lý (worker nodes).
- **Spark Executors:** Là trình thực thi chạy trên các **Worker Nodes** và thực thi các nhiệm vụ được giao bởi trình điều khiển.

Ngoài ra, Apache Spark còn tồn tại thành phần khác là **Cluster Manager** có trách nhiệm duy trì cụm máy thực thi một ứng dụng của Spark. Ta có thể dùng chính **Cluster Manager** của chính Apache Spark hoặc có thể dùng trình quản lý tài nguyên khác như **YARN**

4 Thuật toán xử lý dữ liệu dòng:



Hình 7: Kiến trúc xử lý dữ liệu dòng Spark streaming

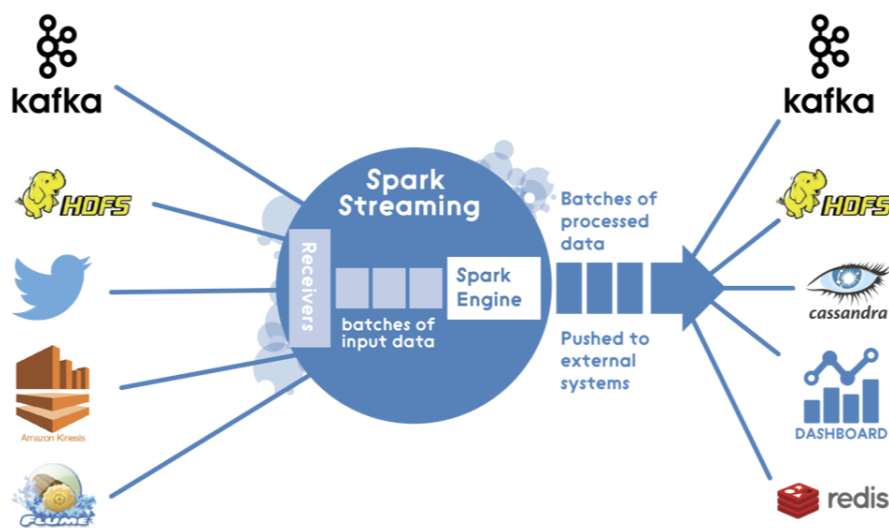
Real-time đang trở thành một xu hướng khi các hệ thống đều đẩy dữ liệu về liên tục, dữ liệu rất lớn theo thời gian. Chính vì vậy việc sử dụng các mô hình thuật toán để xử lý dữ liệu real-time, dữ liệu

dòng rất cần thiết trong bộ xử lý dữ liệu của các hệ thống.

Tổng quan một hệ thống bao gồm 4 giai đoạn:

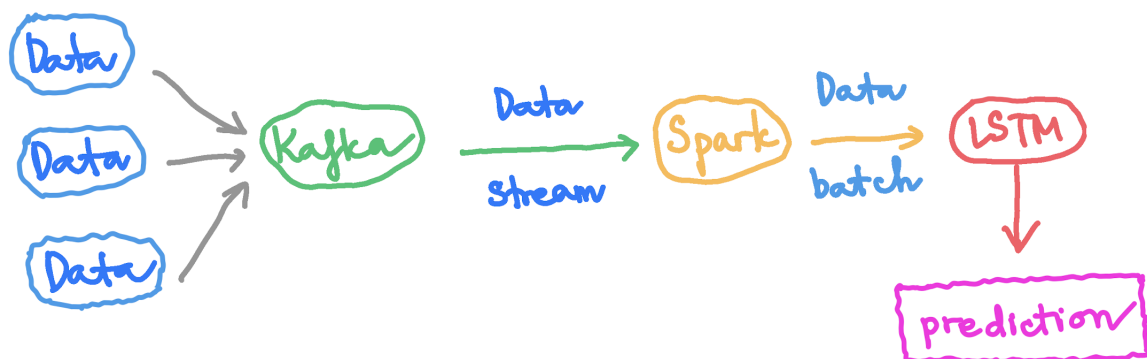
- Dữ liệu đẩy vào Spark Streaming có thể đa dạng nguồn từ realtime streaming như Akka, Kafka, Flume, AWS... hoặc static như HBase, MySQL, PostgreSQL, Elastic Search, Mongo DB, Cassandra...
- Từ Spark Streaming Dữ liệu có thể được đưa vào MLlib để áp dụng các mô hình học máy.
- Hoặc dữ liệu cũng có thể đưa vào Spark SQL phục vụ cho truy vấn dữ liệu
- Cuối cùng, sau các thao tác với dữ liệu, nó sẽ được lưu vào database hoặc file system.

Vậy, dữ liệu đẩy vào Spark Streaming sẽ được xử lý thế nào? Dữ liệu đi qua Spark Streaming sẽ được chia thành các batch nhỏ rồi được Spark Engine xử lý để output ra series các batch dữ liệu mới.



Hình 8: Kiến trúc data streaming pipeline

Dựa vào kiến trúc trên, nhóm đề xuất kiến trúc thuật toán để xử lý dữ liệu dòng đơn giản, xử lý trên dữ liệu **chứng khoán**: Thuật toán xử lý dữ liệu có thể được miêu tả qua các bước:



Hình 9: Thuật toán xử lý data streaming từ IoT

- Data sẽ được lấy từ web chứng khoán dưới dạng file CSV, sau đó được tạo giả thành dữ liệu dòng đổ về kafka.
- Data stream sẽ được xử lý tại Apache Kafka.
- Producer của Apache Kafka sẽ gửi thông điệp (Message) đến consumer, data stream sẽ tiếp tục được đẩy đến Apache Spark Streaming.

- Dữ liệu tại Apache Spark Streaming sẽ được xử lý. Sau đó, dữ liệu sẽ được tiếp tục đi tới các bước sử dụng mô hình.
- Tại đây, nhóm sẽ sử dụng mô hình Long Short-Term Memory (LSTM) để đưa ra những dự đoán về chứng khoán.

5 Mô hình LSTM

5.1 Khái niệm và ý tưởng của LSTM

Long Short-Term Memory (LSTM) là một mạng nơ-ron nhân tạo dùng trong lĩnh vực trí tuệ nhân tạo (AI) và học sâu (Deep learning). Không như các mạng nơ-ron truyền thẳng tiêu chuẩn (feedforward neural network), LSTM mang bản chất là một mạng nơ-ron hồi quy (Recurrent Neural Network - RNN). Đồng nghĩa với việc, LSTM có các kết nối phản hồi.

Các RNN nói chung và LSTM nói riêng được dùng để xử lý thông tin dạng chuỗi (sequence/time-series). Một RNN như vậy có thể xử lý không chỉ một điểm dữ liệu (như là bức ảnh) mà toàn bộ chuỗi dữ liệu (chẳng hạn video - một chuỗi các bức ảnh).

Theo lý thuyết, một RNN cổ điển có thể theo dõi sự phụ thuộc dài hạn trong chuỗi đầu vào. Tuy nhiên vấn đề với nó lại nằm ở bản chất tính toán: khi huấn luyện một RNN cổ điển dùng thuật toán lan truyền ngược (back-propagation), Gradient dài hạn được tính toán bằng lan truyền ngược có thể "biến mất" (tức dần tiến về 0) hoặc "phát nổ" (tiến tới vô cùng). Nguyên do là bởi các tính toán liên quan trong quá trình mà dùng các số được ước lượng thiếu chính xác.

Như vậy về lý thuyết là RNN có thể mang thông tin từ các lớp trước đến các lớp sau, nhưng thực tế là thông tin chỉ mang được qua một số lượng trạng thái nhất định, sau đó thì sẽ bị "vanishing gradient", hay nói cách khác là model RNN cổ điển chỉ học được từ các trạng thái gần nó (short-term memory). Đó là lý do Long short-term Memory (LSTM) ra đời.

5.2 Mô hình LSTM

Một đơn vị LSTM thông dụng bao gồm các thành phần: cell, input gate, output gate và forget gate. Cell được dùng để ghi nhớ các giá trị xuyên suốt các khoảng thời gian và ba cổng (gate) sẽ điều phối dòng chảy của thông tin vào và ra các cell.

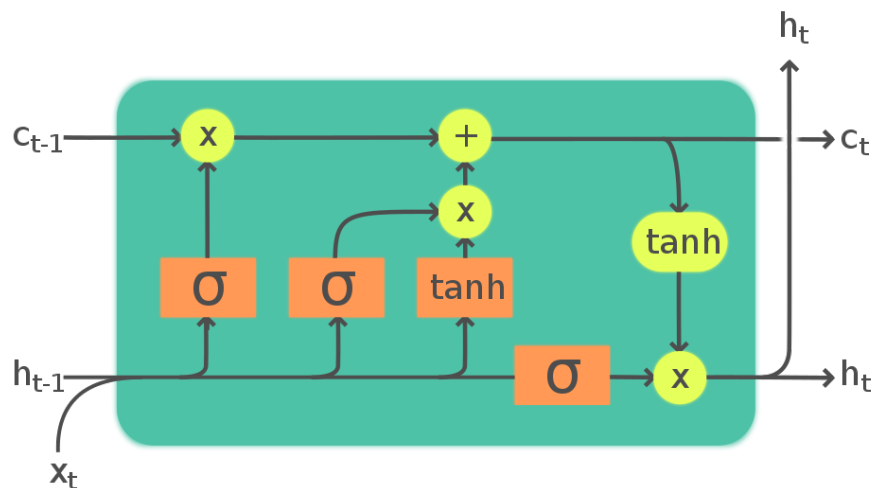
Dạng chuẩn của các phương trình khi truyền vào của đơn vị LSTM được cho như sau:

- $f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$
- $i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$
- $o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$
- $\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$
- $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$
- $h_t = o_t \circ \sigma_h(c_t)$

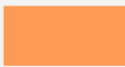



Trong đó:

- t : bước thời gian
- Toán tử \circ : tích Hadamard (tích các phần tử với nhau)
- $x_t \in \mathbb{R}^d$: vector đầu vào của đơn vị LSTM
- $f_t \in (0, 1)^h$: vector kích hoạt của forget gate
- $i_t \in (0, 1)^h$: vector kích hoạt của input gate

- $o_t \in (0, 1)^h$: vector kích hoạt của output gate
- $h_t \in (-1, 1)^h$: vector của hidden state, là vector đầu ra của 1 đơn vị LSTM
- $\tilde{c}_t \in (-1, 1)^h$: vector kích hoạt của đầu vào một cell
- $c_t \in \mathbb{R}^h$: vector trạng thái một cell
- $W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}, b \in \mathbb{R}^h$: ma trận trọng số và các tham số vector bias cần được học lúc huấn luyện
- d, h : số lượng input features và số lượng hidden units
- σ_g : hàm sigmoid
- σ_c : hàm tanh
- σ_h : hàm tanh hoặc hàm $\sigma_h(x) = x$



Legend:

Layer	Componentwise	Copy	Concatenate
			

Hình 10: Mô hình LSTM

5.3 Ứng dụng của LSTM

Về cơ bản, khi áp dụng thuật toán back propagation cho LSTM tương tự như RNN, thì LSTM vẫn bị vanishing gradient nhưng bị ít hơn nhiều so với RNN. Do đó LSTM được dùng phổ biến hơn RNN cho các bài toán thông tin dạng chuỗi. Một mạng LSTM rất thích hợp để dùng trong các tác vụ phân loại, xử lý và dự đoán dựa trên dữ liệu time-series (như dữ liệu chứng khoán mà nhóm sẽ trình bày trong phần sau).

Khả năng ít nhạy cảm với độ dài khoảng trống giúp LSTM có lợi thế hơn so với RNN, các mô hình hidden Markov và các phương pháp học tuần tự trong nhiều ứng dụng. Có thể kể đến một số ứng dụng phổ biến của LSTM như: dự đoán chứng khoán, điều khiển robot, nhận dạng hành động, nhận dạng giọng nói và chăm sóc sức khỏe.

6 Huấn luyện mô hình dự đoán xu hướng của chứng khoán

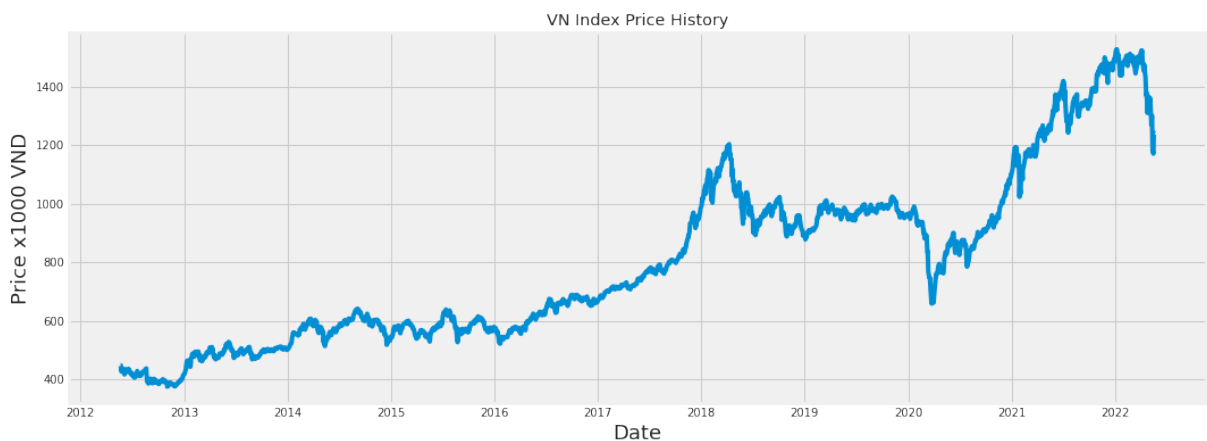
Dữ liệu được dùng để huấn luyện và dự đoán là chỉ số chứng khoán VN Index (VNI), được lấy từ trang: <https://www.investing.com/indices/vn>. Nhóm sẽ thực hiện huấn luyện mô hình thông qua Google Colab, đường dẫn: <https://colab.research.google.com/drive/1DR5RKsh5hWftYNb8k-fC8dSmls0MqDYR?usp=sharing>.

Dữ liệu sẽ là chỉ số giá VN Index được lấy trong 10 năm, từ 20/5/2012 đến 20/5/2022. Sau khi tải dữ liệu từ file csv và tiền xử lý trong Dataframe, dữ liệu sẽ trông như sau:

	Price
Date	
2012-05-21	448.02
2012-05-22	447.94
2012-05-23	436.75
2012-05-24	426.92
2012-05-25	437.38
...	...
2022-05-16	1171.95
2022-05-17	1228.37
2022-05-18	1240.76
2022-05-19	1241.64
2022-05-20	1240.71
2497 rows × 1 columns	

Hình 11: Chỉ số giá VNI trong 10 năm

Bên dưới là đồ thị thể hiện trường "Price" từ dữ liệu trên:



Hình 12: Đồ thị giá của VNI

Tập dữ liệu dùng để huấn luyện sẽ là 90% của tập dữ liệu gốc và được scale về khoảng (0, 1) trước khi bắt đầu. Tập huấn luyện sẽ gồm y_{train} gồm các giá trị cần dự đoán và tương ứng là x_{train} với mỗi điểm dữ liệu sẽ là 60 giá trị trước giá trị cần dự đoán.

Cấu trúc model được xây dựng như hình dưới với hai lớp LSTM:

```
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

Hình 13: Cấu trúc model

7 Demo quá trình truyền dữ liệu và dự đoán

7.1 Truyền dữ liệu

Thực hiện khởi tạo cho **zookeeper**, **server kafka** và tạo **Topic** thông qua chạy bash script có tên **runner.sh**.

```
(hungthinhluu@Friday)-[~/OS_MR/FINAL]
$ ./runner.sh
```

Hình 14: Chạy file runner.sh

Tiếp theo thực hiện chạy file **topic2spark.py** để khởi động cho việc spark nhận dữ liệu theo từng **batch** từ **Topic**.

```
(hungthinhluu@Friday)-[~/OS_MR/FINAL]
$ sudo /opt/spark-3.2.1-bin-hadoop3.2/bin/spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.1 topic2spark.py
```

Hình 15: Chạy file topic2spark.py

Khi chạy ta sẽ thấy hiển thị ra **Schema** của **dataframe** và **Batch 0** sẽ rỗng bởi vì ta chưa truyền dữ liệu vào **Topic**.

```
root
├─ key: binary (nullable = true)
├─ value: binary (nullable = true)
├─ topic: string (nullable = true)
├─ partition: integer (nullable = true)
├─ offset: long (nullable = true)
├─ timestamp: timestamp (nullable = true)
├─ timestampType: integer (nullable = true)

root
├─ Date: string (nullable = true)
├─ Price: double (nullable = true)

Batch: 0

+----+----+
|Date|Price|
+----+----+
+----+----+
```

Hình 16: Schema của dataframe và Batch 0

Tiếp đến, ta sẽ chạy file **kafka2topic.py** để thực hiện đưa dữ liệu vào **Topic** đã tạo trước đó.

```
(hungthinhluu@Friday)-[~/OS_MR/FINAL]
$ python3 kafka2topic.py
```

Hình 17: Chạy file kafka2topic.py

Khi chạy file ta sẽ thấy được các message lần lượt được gửi đi đến **Topic**.

```
Message: {'Date': 'Jun 09, 2021', 'Price': 1332.9, 'Open': 1323.52, 'High': 1339.14, 'Low': 1312.08, 'Vol.': '822.29K', 'Change %': '0.99%'}
Message: {'Date': 'Jun 10, 2021', 'Price': 1323.58, 'Open': 1332.9, 'High': 1336.71, 'Low': 1317.49, 'Vol.': '723.49K', 'Change %': '-0.70%'}
Message: {'Date': 'Jun 11, 2021', 'Price': 1351.74, 'Open': 1326.79, 'High': 1352.47, 'Low': 1322.08, 'Vol.': '731.26K', 'Change %': '2.13%'}
Message: {'Date': 'Jun 14, 2021', 'Price': 1361.72, 'Open': 1352.73, 'High': 1368.53, 'Low': 1352.73, 'Vol.': '775.73K', 'Change %': '0.74%'}
Message: {'Date': 'Jun 15, 2021', 'Price': 1367.36, 'Open': 1361.58, 'High': 1372.72, 'Low': 1356.59, 'Vol.': '730.94K', 'Change %': '0.41%'}
Message: {'Date': 'Jun 16, 2021', 'Price': 1356.52, 'Open': 1363.85, 'High': 1370.14, 'Low': 1350.73, 'Vol.': '769.63K', 'Change %': '-0.79%'}
Message: {'Date': 'Jun 17, 2021', 'Price': 1359.92, 'Open': 1341.06, 'High': 1364.55, 'Low': 1337.49, 'Vol.': '768.73K', 'Change %': '0.25%'}
Message: {'Date': 'Jun 18, 2021', 'Price': 1377.77, 'Open': 1366.19, 'High': 1377.77, 'Low': 1363.52, 'Vol.': '809.47K', 'Change %': '1.31%'}
Message: {'Date': 'Jun 21, 2021', 'Price': 1372.63, 'Open': 1373.59, 'High': 1381.74, 'Low': 1367.85, 'Vol.': '768.18K', 'Change %': '-0.37%'}
Message: {'Date': 'Jun 22, 2021', 'Price': 1379.97, 'Open': 1380.84, 'High': 1385.98, 'Low': 1375.0, 'Vol.': '746.95K', 'Change %': '0.53%'}
Message: {'Date': 'Jun 23, 2021', 'Price': 1376.87, 'Open': 1383.15, 'High': 1388.33, 'Low': 1372.06, 'Vol.': '710.77K', 'Change %': '-0.22%'}
Message: {'Date': 'Jun 24, 2021', 'Price': 1379.72, 'Open': 1376.87, 'High': 1384.71, 'Low': 1372.99, 'Vol.': '591.75K', 'Change %': '0.21%'}
Message: {'Date': 'Jun 25, 2021', 'Price': 1390.12, 'Open': 1382.11, 'High': 1390.12, 'Low': 1372.8, 'Vol.': '598.11K', 'Change %': '0.75%'}
Message: {'Date': 'Jun 28, 2021', 'Price': 1405.81, 'Open': 1397.96, 'High': 1406.02, 'Low': 1390.56, 'Vol.': '699.51K', 'Change %': '1.13%'}
```

Hình 18: Các message đang được gửi vào Topic

Khi có các message được gửi vào Topic, khi đó spark sẽ lấy được dữ liệu đó theo dòng và đưa ra theo từng mẻ (Batch).

```
Batch: 1
+-----+-----+
|      Date| Price|
+-----+-----+
|May 24, 2021|1297.98|
|May 25, 2021|1308.58|
|May 26, 2021| 1316.7|
|May 27, 2021|1303.57|
|May 28, 2021|1320.46|
|May 31, 2021|1328.05|
|Jun 01, 2021|1337.78|
+-----+-----+

Batch: 2
+-----+-----+
|      Date| Price|
+-----+-----+
|Jun 02, 2021|1340.78|
|Jun 03, 2021|1364.28|
|Jun 04, 2021|1374.05|
|Jun 07, 2021|1358.78|
|Jun 08, 2021|1319.88|
|Jun 09, 2021| 1332.9|
|Jun 10, 2021|1323.58|
|Jun 11, 2021|1351.74|
|Jun 14, 2021|1361.72|
|Jun 15, 2021|1367.36|
+-----+-----+
```

Hình 19: Các Batch dữ liệu từ spark

7.2 Dự đoán

Khi đã có được các dữ liệu và mô hình đã được huấn luyện từ trước đó, ta sẽ thực hiện dự đoán giá cổ phiếu từ dữ liệu đã lấy được từ spark.

Ta thực hiện lấy mô hình đã được huấn luyện trước đó.

```
!wget "https://github.com/HungThinhLuu/OS_KSTN/raw/master/stock_LSTM.zip"
```

Hình 20: Lấy model đã huấn luyện

Sau đó, ta load model hình đã lấy về vào colab.

```
model = load_model('stock_LSTM')
```

Hình 21: Load model

```
x_test = []
y_test = data.values[60:, :]
for i in range(60, len(scaled_data)):
    x_test.append(scaled_data[i-60:i, 0])

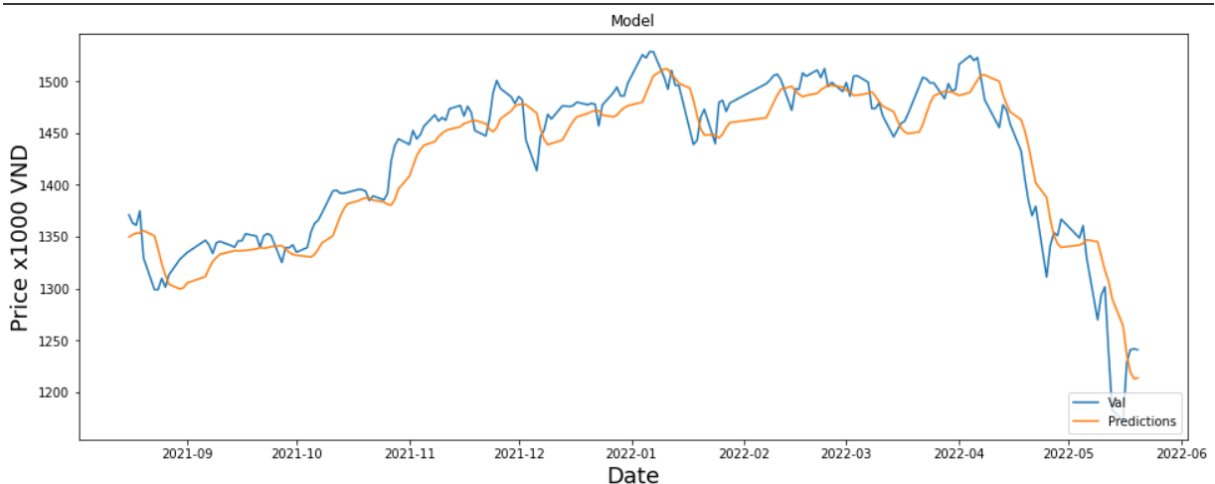
x_test = np.array(x_test)

x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))

predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

Hình 22: Dự đoán giá cổ phiếu

Thực hiện dự đoán giá cổ phiếu từ dữ liệu đã lấy được.
Kết quả được thể hiện như hình dưới.



Hình 23: Đồ thị thể hiện giá thật và giá dự đoán

Đồng thời, ta có kết quả đánh giá mô hình trên như sau:

Model	RMSE
LSTM	26.45

Đánh giá: Ta thấy mô hình trên dự đoán tương đối được xu hướng của giá chứng khoán nhưng có vẻ vẫn có độ trễ và chưa thể hiện được những cái xung gai trên đồ thị.

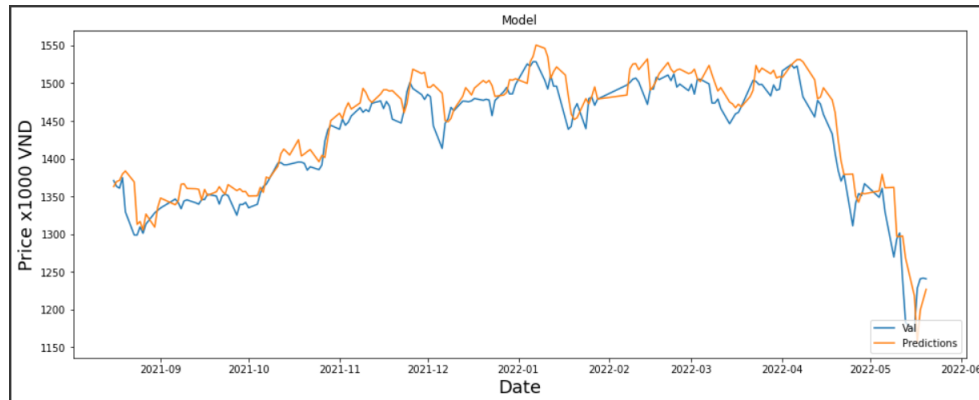
8 So sánh mô hình LSTM và mô hình RNN cho dự đoán giá chứng khoán

Thực hiện xây dựng thêm mô hình dự đoán chứng khoán bằng RNN để so sánh với mô hình LSTM đã được thực hiện trước đó. Mô hình RNN sẽ bao gồm hai lớp RNN và hai lớp trải phẳng.

```
model = Sequential()
model.add(SimpleRNN(128, return_sequences=True, input_shape= (x_train.shape[1], 1)))
model.add(SimpleRNN(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

Hình 24: Cấu trúc model RNN cho dự đoán chứng khoán

Sau khi đã thực hiện huấn luyện mô hình, sau đây là đồ thị thể hiện giá trị thực và giá trị dự đoán của mô hình trên.



Hình 25: Đồ thị thể hiện giá trị thực và giá trị dự đoán của mô hình RNN

Ta thấy giữa hai mô hình có giá trị **RMSE** như sau:

Model	RMSE
LSTM	26.45
RNN	25.72641

Nhìn chung ta thấy có vẻ như mô hình của **RNN** lại đem lại kết quả tốt hơn, các xu hướng lên xuống của mô hình RNN dự đoán bám tương đối đúng với thực tế, có nhiều xung gai trong biểu đồ hơn, về vấn đề này có lẽ là do mô hình mà ta huấn luyện đang chỉ hướng đến việc dự đoán giá chứng khoán của một ngày sau đó nên không thể thấy được sức mạnh thật sự của **LSTM**.

9 Hạn chế của mô hình và những định hướng tiếp theo

Đây chỉ là mô hình đơn giản, trong tương lai, nếu có thời gian thì nhóm sẽ hiện thực các mô hình phức tạp hơn.

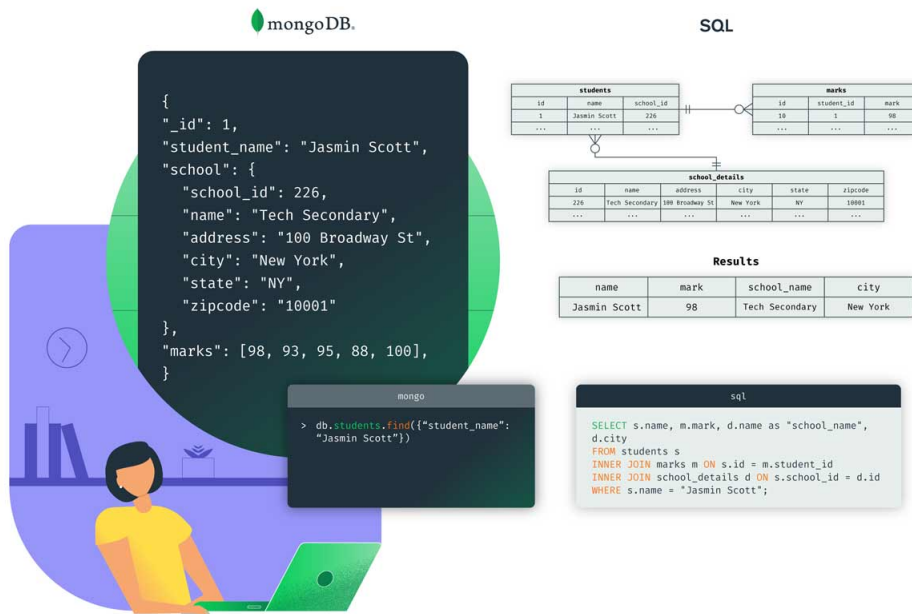


Hình 26: Giá cổ phiếu và chứng khoán

Về hạn chế, mô hình chỉ dựa trên những dữ liệu số đã có sẵn, tuy nhiên, sự thay đổi giá chứng khoán phụ thuộc vào rất nhiều yếu tố tác động bên ngoài. Do đó, mô hình không thể dự đoán được

kết quả với độ chính xác quá cao khi có những sự kiện đặc biệt diễn ra. Trong tương lai, có thể nhóm sẽ nghiên cứu thêm về các mô hình có thể cho người dùng cập nhật các yếu tố tác động đến giá, hay thiết kế thêm các hàm nội suy để có thể tự suy luận ra những yếu tố sẽ tác động đến giá cả trên thị trường.

Ngoài ra, nhóm cũng sẽ nghiên cứu thêm về các cách lưu trữ dữ liệu lâu dài như trong data warehouse, data lake, cũng như 1 số nơi lưu trữ dữ liệu nổi tiếng như MongoDB, Cassandra,... để nâng cao hiệu suất sử dụng dữ liệu vào thực tế.



Hình 27: MongoDB

Tài liệu

- [1] <https://spark.apache.org/>
- [2] <https://www.knowledgehut.com/tutorials/apache-spark-tutorial>
- [3] <https://viblo.asia/p/tim-hieu-ve-apache-spark-ByEZkQQW5Q0>
- [4] <https://viblo.asia/p/kafka-la-gi-gDVK2Q7A5Lj>
- [5] https://www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm?fbclid=IwAR06M5-3JgAw0UM9eR6sipNbzg6_Jtel7-FwVneHGhn-S-8iA1S32Ea4dJs
- [5] <https://www.datacamp.com/tutorial/lstm-python-stock-market>
- [6] <https://www.analyticsvidhya.com/blog/2021/05/stock-price-prediction-and-forecasting-using-sta#:~:text=LSTMs%20are%20widely%20used%20for,the%20information%20that%20is%20not.>
- [7] https://en.wikipedia.org/wiki/Long_short-term_memory