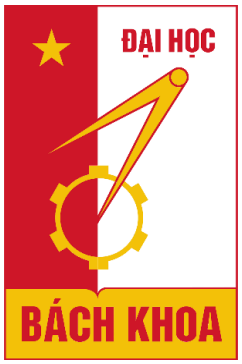
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



BÁO CÁO BÀI TẬP LỚN

Đề tài: Lưu trữ và xử lý dữ liệu tiền ảo theo thời gian thực

Lớp: 151392

Học phần: Lưu trữ và xử lý dữ liệu lớn

Mã học phần: IT4931

Giảng viên hướng dẫn: TS. Trần Văn Đặng

Danh sách thành viên nhóm 5:

Họ và tên	Mã số sinh viên
Nguyễn Quốc Dũng	20215329

Hà Nội, tháng 6 năm 2024

CHƯƠNG 1. GIỚI THIỆU BÀI TOÁN

1.1 Giới thiệu

Trong thế giới tài chính hiện đại, tiền ảo hay crypto là một dạng tiền điện tử rất còn mới mẻ và đang đà phát triển. Việc lưu trữ và xử lý dữ liệu tiền ảo thời gian thực là rất quan trọng. Điều này không chỉ giúp các nhà đầu tư có được cái nhìn sâu sắc và kịp thời về thị trường, mà còn là cơ sở để phát triển các chiến lược giao dịch và quản lý rủi ro hiệu quả. Báo cáo này sẽ trình bày phương pháp và các công cụ lưu trữ dữ liệu lớn trong quá trình xây dựng hệ thống xử lý và phân tích dữ liệu tiền ảo trong thời gian thực.

1.2 Vấn đề

Giá tiền ảo thay đổi trong thời gian thực và thường khó dự đoán bằng các phương pháp truyền thống, tạo ra một thách thức lớn cho các nhà đầu tư và nhà phân tích. Sự biến động không thể dự đoán của giá tiền ảo là do nhiều yếu tố - từ tin tức kinh tế, báo cáo tài chính, đến hành động của các nhà đầu tư khác. Hơn nữa, các yếu tố dẫn đến thay đổi giá cổ phiếu phức tạp và đa dạng, bao gồm cả các yếu tố cơ bản của công ty và yếu tố kỹ thuật trên thị trường. Điều này đòi hỏi cần có phương pháp tiên tiến và đa dạng hơn trong việc lưu trữ và phân tích dữ liệu để có thể hiểu và dự báo chính xác các xu hướng giá cổ phiếu, đặc biệt là trong môi trường giao dịch nhanh và biến động cao của ngày nay.

CHƯƠNG 2. CÁC NGUỒN DỮ LIỆU & CẤU TRÚC

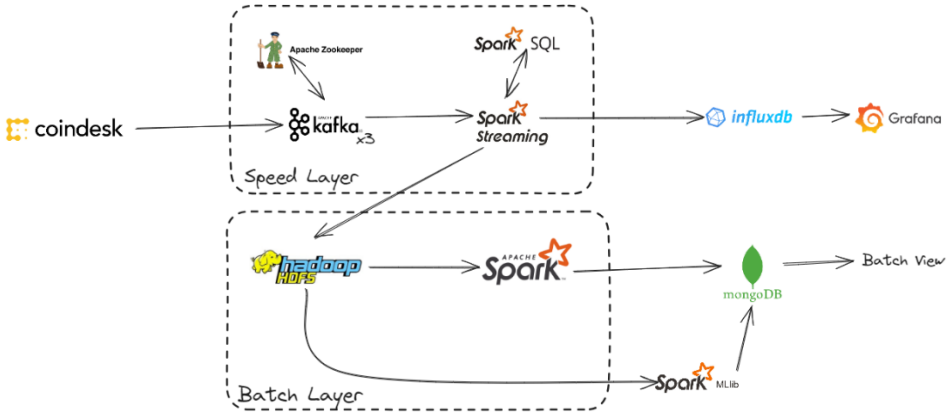
Nguồn dữ liệu cho việc phân tích và xử lý tiền thời gian thực trong dự án này chủ yếu đến từ nguồn là CoinDesk và dữ liệu được crawl trực tiếp từ API của trang web. Dữ liệu này bao gồm thông tin chi tiết về các mã chứng khoán, được cấu trúc như sau:

- ISO: Mã ISO của đồng tiền.
- Name: Tên đồng tiền.
- Date_time: Thời gian xuất hiện của bản ghi.
- Current_price: Giá hiện tại.
- Open: Giá mở cửa.
- High: Giá cao nhất.
- Low: Giá thấp nhất.
- Close: Giá đóng cửa

Thông tin từ nguồn dữ liệu này đóng vai trò quan trọng trong việc cung cấp dữ liệu chính xác và cập nhật cho hệ thống phân tích.

CHƯƠNG 3. KIẾN TRÚC HỆ THỐNG

Phân tích tiền ảo thời gian thực đòi hỏi một hệ thống có khả năng xử lý dữ liệu lớn một cách nhanh chóng và chính xác. Chương này sẽ mô tả chi tiết về kiến trúc hệ thống và các công nghệ được sử dụng.



3.1 Thu thập dữ liệu

Dữ liệu được thu thập real-time từ CoinDesk. Sau đó được chuyển đến Kafka, một nền tảng xử lý stream-data, để quản lý và phân phối dữ liệu đầu vào.

3.2 Tầng xử lý dữ liệu

Tầng xử lý dữ liệu được chia thành hai lớp chính: Batch Layer và Speed Layer.

3.2.1 Speed Layer

Lớp Speed xử lý dữ liệu thời gian thực sử dụng Spark Streaming và Spark SQL để cung cấp thông tin cập nhật liên tục và nhanh chóng. Dữ liệu sau khi transform được chuyển đến InfluxDB và Hadoop HDFS để lưu trữ.

3.2.2 Batch Layer

Lớp Batch xử lý dữ liệu lớn thông qua Hadoop HDFS và Apache Spark để thực hiện các tính toán theo lô (batch processing) với tần suất là 1 lần / ngày. Kết quả sau khi xử lý xong sẽ được lưu trữ vào MongoDB.

3.3 Phân tích và trực quan hoá

Để phân tích dữ liệu, MLlib được sử dụng để cài đặt mô hình học máy với dữ liệu được lưu trữ tại Hadoop HDFS.

Kết quả cuối cùng, đối với Speed Layer, dữ liệu tại InfluxDB sẽ được trực quan hóa dưới dạng stream thông qua Grafana. Đối với Batch Layer dữ liệu từ MongoDB sẽ được hiện thị dưới dạng Batch View vào cuối mỗi ngày

Như vậy, hệ thống sử dụng kết hợp giữa xử lý theo lô (batch processing) và xử lý theo dòng dữ liệu (stream processing). Với mục tiêu, cung cấp cái nhìn sâu sắc và cập nhật liên tục về thị trường tiền ảo.

CHƯƠNG 4. CHI TIẾT HƠN VỀ DỰ ÁN

4.1 Thu thập dữ liệu

```python
URL = 'https://production.api.coindesk.com/v2/tb/price/ticker?assets='


2 usages  J Johnson +1
def default(args):
    if len(args) == 0:
        raise Exception("Missing argument...")

    final_url = URL + args
    result = get(final_url)
    json_data = result.json()

    individual_symbols = args.split(',')

    results = []

    for s in individual_symbols:
        coin_data = json_data['data'][s]
        results.append({
            "iso": coin_data['iso'],
            "name": coin_data['name'],
            "date_time": datetime.now().strftime("%Y-%m-%d %H:%M:%S%z"),
            "current_price": coin_data['ohlc']['c'],
            "open": coin_data['ohlc']['o'],
            "high": coin_data['ohlc']['h'],
            "low": coin_data['ohlc']['l'],
            "close": coin_data['ohlc']['c']
        })

    return results
```

Hình 4.1 Code crawl dữ liệu

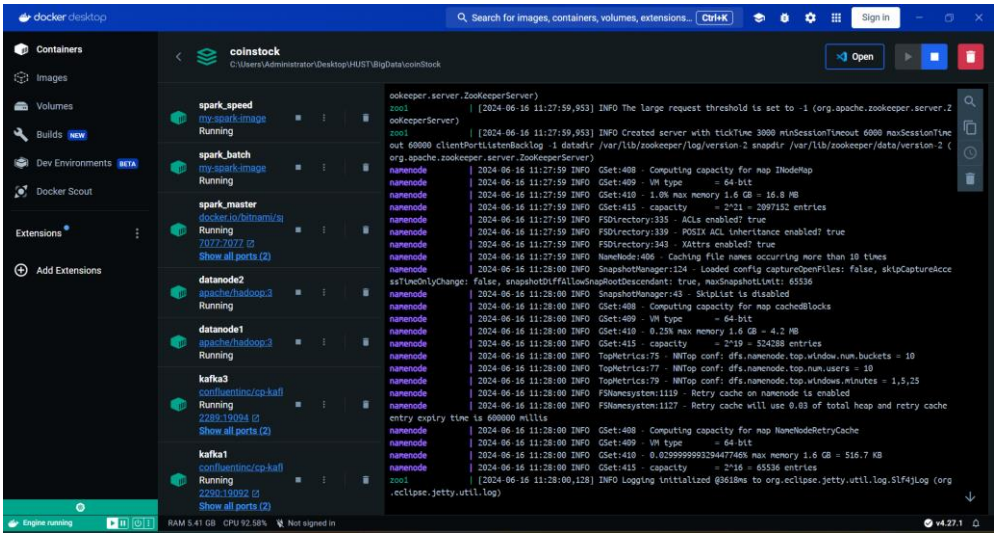
Hình 4.2 Kết quả dữ liệu sau khi crawl

Dữ liệu được crawl sẽ được sử dụng làm một nguồn producer cung cấp dữ liệu cho hệ thống.

4.2 Sử dụng Docker để thiết lập các thành phần hệ thống trên môi trường phân tán

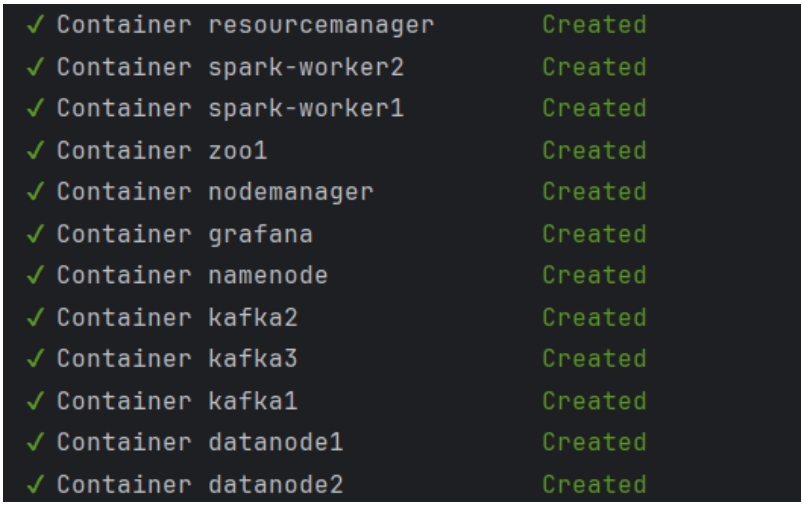
Để giả lập môi trường phân tán, các thành phần trong hệ thống đều được khởi chạy trên Docker thông qua docker-compose.yml. Trong đó có một image

phục vụ cho spark-submit được tạo lại có cài sẵn các thư viện cần thiết cho việc execute python file.



Hình 4.3 Các thành phần hệ thống được khởi chạy trên Docker

4.3 Thiết lập và sử dụng Multi Kafka brokers cùng ZooKeeper



Hình 4.4 Khởi động các container

```
zoo1:
  image: confluentinc/cp-zookeeper:7.3.2
  hostname: zoo1
  container_name: zoo1
  ports:
    - "2181:2181"
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_SERVER_ID: 1
    ZOOKEEPER_SERVERS: zoo1:2888:3888
  networks:
    - broker-kafka
kafka1:
  image: confluentinc/cp-kafka:7.3.2
  hostname: kafka1
  container_name: kafka1
  ports:
    - "9092:9092"
    - "19092"
  environment:
    KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka1:19092,EXTERNAL://${DOCKER_HOST_IP:-127.0.0.1}:9092,DOCKER://host.docker.internal:29092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT,DOCKER:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
    KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181"
    KAFKA_BROKER_ID: 1
    KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
    KAFKA_AUTHORIZER_CLASS_NAME: kafka.security.authorizer.AclAuthorizer
    KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: "true"
    KAFKA_NUM_PARTITIONS: 3
  depends_on:
    - zoo1
```

Hình 4.5 Zookeeper và broker Kafka

3 broker kafka được cấu hình tại: kafka1:19092, kafka2:19093, kafka3:19094.

4.4 Xử lý dữ liệu thời gian thực sử dụng Apache Spark Streaming

```
if __name__ == "__main__":
    spark = (
        SparkSession.builder.appName("KafkaInfluxDBStreaming")
        .master("spark://spark-master:7077")
        .config("spark.jars.packages", ",".join(packages))
        .getOrCreate()
    )

    spark.sparkContext.setLogLevel("INFO")

    stockDataframe = spark \
        .readStream \
        .format("kafka") \
        .option( key: "kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVERS) \
        .option( key: "subscribe", KAFKA_TOPIC_NAME) \
        .load()

    stockDataframe = stockDataframe.select(col("value").cast("string").alias("data"))
    inputStream = stockDataframe.selectExpr("CAST(data as STRING)")

    stock_price_schema = StructType([
        StructField( name: "iso", StringType(), nullable: True),
        StructField( name: "name", StringType(), nullable: True),
        StructField( name: "current_price", DoubleType(), nullable: True),
        StructField( name: "open", DoubleType(), nullable: True),
        StructField( name: "high", DoubleType(), nullable: True),
        StructField( name: "low", DoubleType(), nullable: True),
        StructField( name: "close", DoubleType(), nullable: True),
        StructField( name: "date_time", StringType(), nullable: True)
    ])
```

Hình 4.6 Code Spark Streaming

```
24/06/16 12:02:07 INFO DAGScheduler: Job 52 finished: call at /opt/bitnami/spark/python/lib/py4j-0.10.9.5-src.zip/py4j/clientserver.py:617, took 0.537789 s
["BTC", "Bitcoin", 66442.7846359773, 66280.83071709584, 66681.81572379959, 65846.13470482016, 66442.7846359773, "2024-06-16 19:02:06"]
---------------------
["ETH", "Ethereum", 3551.6567677362136, 3536.6071042008684, 3589.6116053697947, 3535.836775962719, 3551.6567677362136, "2024-06-16 19:02:06"]
---------------------
["ETHFI", "ether.fi", 3.9510062371, 3.8335795795, 4.0711812159, 3.7963240585, 3.9510062371, "2024-06-16 19:02:06"]
---------------------
["DOGE", "Dogecoin", 0.1357487928, 0.1369720102, 0.13759965, 0.1348752542, 0.1357487928, "2024-06-16 19:02:06"]
---------------------
["ZETA", "ZetaChain", 1.0343858023, 1.0380019553, 1.0501979546, 1.0030871901, 1.0343858023, "2024-06-16 19:02:06"]
---------------------
Batch processed 17 done!
```

Hình 4.7 Xử lý dữ liệu thời gian thực sử dụng Apache Spark Streaming

Từ hình 4.7, log của dòng dữ liệu (stream processing) đang được hiển thị theo các mini-batch.

4.5 Thực hiện các thao tác chuyển đổi và truy vấn dữ liệu bằng Spark SQL

```python
# Remove duplicates
df = df.dropDuplicates()

# Basic Statistics for each stock
basic_stats = df.groupBy("iso").agg(
    F.mean("open").alias("avg_open"),
    F.mean("high").alias("avg_high"),
    F.mean("low").alias("avg_low"),
    F.mean("close").alias("avg_close"),
    F.stddev("close").alias("std_dev_close"),
    F.max("high").alias("historical_high"),
    F.min("low").alias("historical_low")
)

# Show basic statistics
basic_stats.show()

# Write basic_stats DataFrame to MongoDB
(basic_stats.write.format("mongo").mode("append")
    .option( key: "database",  value: "bigdata").option( key: "collection",  value: "basic_stats").save())

# Filter out Bitcoin data and prepare it as a feature
btc_df = df.filter(df.iso == 'BTC').select("date_time", df.current_price.alias("btc_price"))

# Join BTC data with the main dataframe on date_time
df = df.join(btc_df, on="date_time")

# Assemble features for ML model
assembler = VectorAssembler(inputCols=["btc_price"], outputCol="features")
df = assembler.transform(df)
```

Hình 4.8 Các thao tác trên dữ liệu sử dụng Spark SQL

Hình 4.8, là một số thao tác trong quá trình xử lý dữ liệu với Spark SQL như: dropDuplicate(), filter(), tính max, min, mean, std-dev, v.v

4.6 Xử lý dữ liệu theo lô sử dụng Apache Spark

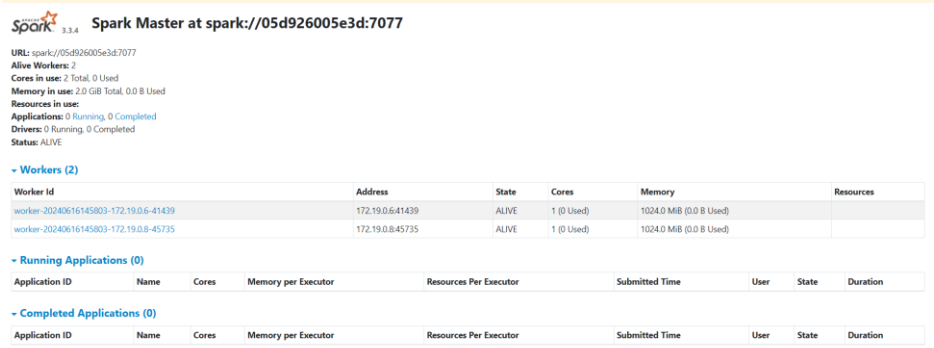
iso	avg_open	avg_high	avg_low	avg_close	std_dev_close	historical_high	historical_low
ZETA	1.037028455982353	1.0501979546	1.0030871901000002	1.0390431499705883	0.0026301764273285 19	1.0501979546	1.0030871901
ETH	3557.7287430509555	3606.8120380819387	3535.9136161289066	3584.5206784170923	0.7719423636426 21	3606.81203808194	3535.913616128907
TON	8.080760642229 41	8.1105445807	7.749244813900002	7.985538694376472	0.0074685041736882 35	8.1260772589	7.7492448139
DOGE	0.13654611282941176	0.13759965000000002	0.1348752542	0.13652190748823526	3.817778846327358E-5	0.13759965	0.1348752542
INJ	26.09377705765882	26.66472427680001	24.4675209321	25.010940715082363	0.0347728120805732 84	26.6647242768	24.4675209321
OKB	45.82810460593529	46.4435857202	45.757484751341 19	45.87240244494118	0.0312467112396151 11	46.4435857202	45.6662294656
RUNE	4.7262784346588225	4.7802550631	4.583909833199998	4.6946240740823 54	0.0026160253387851 65	4.7802550631	4.5839098332
PEPE	1.206378823529411...	1.228439999999999...	1.146670000000000...	1.212631764705882...	3.955729277598176E-9	1.22844E-5	1.14667E-5
SUN	0.012365409535294122	0.0124604455	0.012197545499999999	0.012302962888235293	6.087625291204116E-6	0.0124604455	0.0121975455
BTC	66186.50609890305	66728.5772907998	65846.13470482014	66550.38219961179	9.056651833870994	66728.5772907998	65846.13470482016
ETHFI	3.8886222592000004	4.071181215899999	3.8456058790999994	4.0082637735352 93	0.0030552873228 97...	4.0711812159	3.8456058791
SOL	144.61886221682622	146.62768550881395	143.05450288730887	146.18444374370134	0.0953751247929921 5	146.62768550881 39	143.05450288730884
WIF	2.4010517801470583	2.5789173342	2.3102194782999996	2.5447591039117 64	0.0016380840985 05...	2.5789173342	2.3102194783
BNB	606.1912657254236	611.511162389	601.958141428	609.8952116566942	0.1330253764227 9835	611.511162389	601.958141428
AVA	0.7566172723823529	0.7679110846999998	0.7320058395999999	0.7452492991176471	4.1065118114963 79...	0.7679110847	0.7320058396
SHIB	2.079458235294117...	2.103240000000000...	2.042279999999999...	2.091164705882352...	9.6556406860876 61E-9	2.10324E-5	2.04228E-5

Hình 4.9 Kết quả xử lý theo lô

```python
import pyhdfs
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.types import StructType, StructField, StringType, DoubleType

# Initialize Spark Session
spark = SparkSession.builder \
    .appName("Crypto Dependency Analysis") \
    .config("spark.mongodb.output.uri", "mongodb://root:admin@mongodb:27017/bigdata.stock2024") \
    .getOrCreate()

# Set up the HDFS client
hdfs = pyhdfs.HdfsClient(hosts="namenode:9870", user_name="hdfs")
directory = '/data'
if not hdfs.exists(directory):
    hdfs.mkdirs(directory)
files = hdfs.listdir(directory)
print("Files in '{}':".format(directory), files)

# Define the schema for the DataFrame
schema = StructType([
    StructField( name: "iso", StringType(),  nullable: True),
    StructField( name: "name", StringType(),  nullable: True),
    StructField( name: "current_price", DoubleType(),  nullable: True),
    StructField( name: "open", DoubleType(),  nullable: True),
    StructField( name: "high", DoubleType(),  nullable: True),
    StructField( name: "low", DoubleType(),  nullable: True),
    StructField( name: "close", DoubleType(),  nullable: True),
    StructField( name: "date_time", StringType(),  nullable: True)
])
```

Hình 4.10 Một đoạn code trong xử lý theo lô

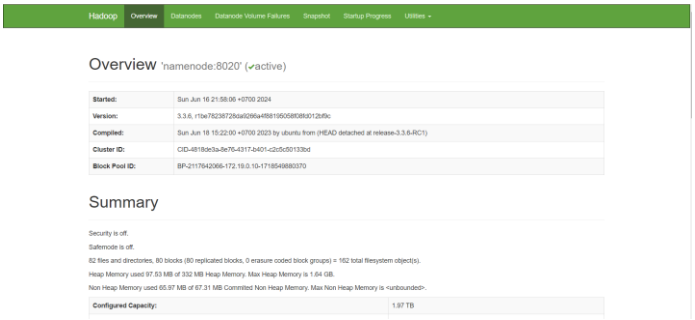
4.7 Thiết lập và sử dụng Multi-Node Spark Cluster



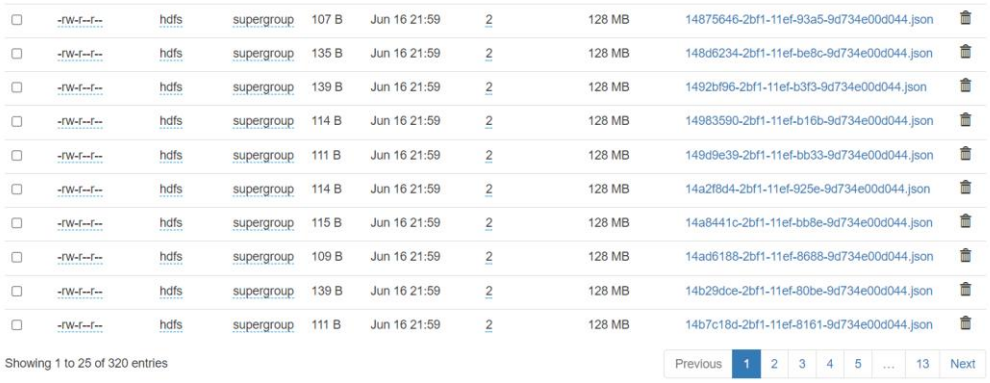
Hình 4.11 Giao diện hiển thị thông tin Spark Master

Từ Spark Master (hình 4.11) có thể thấy Apache Spark hoạt động với 2 worker node.

4.8 Lưu trữ dữ liệu trên Hadoop HDFS



Hình 4.12 Tổng quan của Hadoop HDFS



Hình 4.13 Dữ liệu được lưu trữ trong HDFS

4.9 Cài đặt mô hình máy học thông qua Spark Mllib

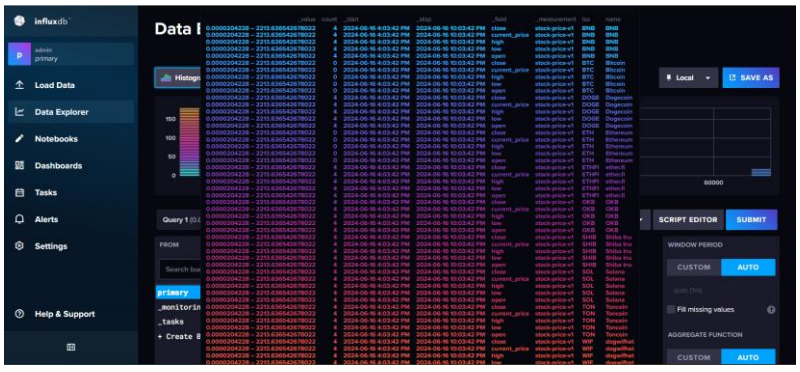
```
# K-Means Model
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans

input_cols = ["avg_price_change", "std_dev_price_change"]
vec_assembler = VectorAssembler(inputCols=input_cols, outputCol="features")
df_kmeans = vec_assembler.transform(basic_stats)

kmeans = KMeans().setK(2).setSeed(1).setFeaturesCol("features")
model = kmeans.fit(df_kmeans)

# Predict clusters for each crypto
predictions = model.transform(df_kmeans)
predictions.select("iso", "prediction").show()
```

```
+-----+----------+
|  iso|prediction|
+-----+----------+
| ZETA|         1|
|  ETH|         1|
|  TON|         0|
| DOGE|         1|
|  INJ|         0|
|  OKB|         1|
| RUNE|         0|
| PEPE|         1|
|  SUN|         0|
|  BTC|         1|
|ETHFI|         1|
|  SOL|         1|
|  BNB|         1|
|  WIF|         1|
|  AVA|         0|
| SHIB|         1|
+-----+----------+
```

Hình 4.14 Thuật toán K-means và kết quả phân cụm

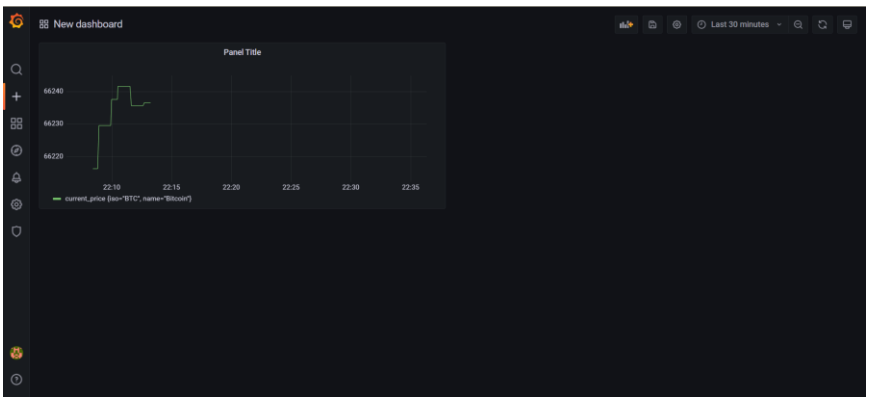
4.10 Triển khai và lưu trữ cơ sở dữ liệu NoSQL MongoDB

4.11 Triển khai và lưu trữ cơ sở dữ liệu time-series InfluxDB



Hình 4.15 Dữ liệu tại bucket primary trong InfluxDB

4.12 Trực quan hoá dữ dạng stream-data thông qua Grafana



Hình 4.16 Trực quan hoá dữ liệu với Grafana

Dữ liệu từ dạng time-series từ InfluxDB, sữ được trực quan hoá theo dạng stream thông qua nền tảng Grafana.

4.13 Thiết kế kiến trúc hệ thống theo mô hình lambda

Việc thiết kế và khởi tạo kiến trúc lambda đã giúp cho hệ thống xử lý dữ liệu phân tán linh hoạt và hiệu quả hơn, có thể đáp ứng được các yêu cầu khác nhau. Điển hình của kiến trúc là 2 nhánh Batch Layer và Speed Layer như hình 3.1. Chi tiết đã được mô tả trong chương 3.

CHƯƠNG 5. KẾT LUẬN

Trong dự án này, em đã thành công trong việc xây dựng một hệ thống xử lý và phân tích dữ liệu tiền ảo thời gian thực. Kết hợp các công cụ hỗ trợ xử lý dữ liệu lớn một cách nhanh chóng và chính xác gồm: Kafka, Apache Spark, Hadoop cùng với các cơ sở dữ liệu NoSQL như MongoDB hay time-series như InfluxDB.

Em cũng đã gặp phải nhiều thách thức, và các vấn đề khác nhau. Tuy nhiên, từ đây nhiều bài học và trải nghiệm trong lưu trữ và xử lý dữ liệu lớn đã được rút ra. Dự án là bước đầu cho việc ứng dụng dữ liệu lớn trong các lĩnh vực tài chính, đặc biệt là trong việc phân tích và dự báo thị trường tiền ảo của em.

Em hy vọng rằng, dự án có thể hoàn thiện hơn trong tương lai. Cùng với các thông tin, phân tích đa dạng, thực tế và hữu ích hơn nữa cho người sử dụng.