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:

Hệ thống lưu trữ và phân tích dữ liệu việc làm

Nhóm:	**18**
Mã lớp:	**154050**
Giảng viên hướng dẫn:	**TS. Trần Việt Trung**
Sinh viên thực hiện:	**20207601 – Phạm Ngọc Hải**
	20207654 – Lê Đình Hoàng Anh
	20215363 – Trịnh Văn Hậu
	20215408 – Nguyễn Trung Kiên
	20215295 – Lê Trọng Bảo An

Hà Nội, tháng 12 năm 2024

MỤC LỤC

I. Đặt vấn đề

1. Tổng quan bài toán

Trong bối cảnh toàn cầu hóa và cách mạng công nghiệp 4.0, thị trường lao động đang trải qua những thay đổi sâu sắc. Công nghệ hiện đại, tự động hóa và trí tuệ nhân tạo đang dần thay thế nhiều công việc truyền thống, đồng thời tạo ra những nghề nghiệp mới đòi hỏi kỹ năng chuyên môn cao. Cạnh tranh việc làm trở nên khốc liệt hơn, không chỉ giữa các cá nhân mà còn giữa các khu vực và quốc gia. Trong bối cảnh này, việc nắm bắt được các xu hướng và dự báo chính xác về nhu cầu lao động trở nên vô cùng quan trọng để các cá nhân và tổ chức thích nghi và phát triển.

Phân tích dữ liệu việc làm đóng vai trò then chốt trong việc giải quyết các thách thức và tận dụng cơ hội từ thị trường lao động. Thông qua việc sử dụng dữ liệu, các tổ chức có thể xác định xu hướng tuyển dụng, nhu cầu về kỹ năng và sự chuyển dịch của các ngành nghề, từ đó xây dựng chiến lược tuyển dụng và đào tạo phù hợp. Đối với cá nhân, việc phân tích dữ liệu giúp họ hiểu rõ các yêu cầu của thị trường, từ đó định hướng nghề nghiệp và nâng cao năng lực cạnh tranh. Đồng thời, phân tích dữ liệu cũng hỗ trợ các nhà hoạch định chính sách thiết kế các chương trình giáo dục và đào tạo hiệu quả, giảm tình trạng thất nghiệp và thúc đẩy sự phát triển bền vững của nguồn nhân lực. Nhờ vậy, phân tích dữ liệu không chỉ là công cụ mà còn là động lực quan trọng thúc đẩy sự phát triển toàn diện của thị trường lao động hiện đại..

2. Phân tích bài toán trong xử lý dữ liệu lớn

Trong thời đại số hóa và sự mở rộng nhanh chóng của thị trường lao động, bài toán phân tích dữ liệu việc làm trở thành một thách thức quan trọng cần được giải quyết bằng các hệ thống xử lý dữ liệu lớn. Để đạt được hiệu quả trong việc phân tích và khai thác dữ liệu, cần giải quyết các vấn đề cốt lõi như sau:

- **Lưu trữ và xử lý dữ liệu khối lượng lớn:** Dữ liệu việc làm không chỉ có số lượng lớn mà còn phát sinh liên tục theo thời gian. Việc sử dụng Hadoop HDFS đảm bảo khả năng lưu trữ phân tán và mở rộng linh hoạt, giúp hệ thống lưu trữ dữ liệu một cách bền vững và hiệu quả. Đồng thời, Apache Spark cung cấp khả năng xử lý nhanh chóng trên tập dữ liệu khổng lồ, hỗ trợ cả xử lý theo lô (batch) và thời gian thực (streaming).
- **Xử lý dữ liệu không đồng nhất:** Dữ liệu việc làm thường bao gồm các thông tin đa dạng như tiêu đề công việc, yêu cầu kỹ năng, mức lương và vị trí địa lý, với nhiều định dạng khác nhau như CSV, JSON hoặc văn bản

phi cấu trúc. Thêm vào đó, dữ liệu có thể bị thiếu thông tin, trùng lặp hoặc không thống nhất giữa các nguồn. Việc áp dụng các quy trình tiền xử lý tự động trong Spark giúp làm sạch, chuẩn hóa và chuyển đổi dữ liệu, từ đó tạo ra đầu vào chất lượng cao cho quá trình phân tích.

- **Phân tích và trực quan hóa dữ liệu:** Các công nghệ hiện đại như Apache Kafka và Apache Spark đóng vai trò quan trọng trong việc xử lý dữ liệu theo cả lô (batch) và thời gian thực (streaming). Spark SQL cho phép truy vấn, tổng hợp và phân tích dữ liệu nhanh chóng, trong khi ClickHouse đóng vai trò lưu trữ và tối ưu hóa cho các truy vấn phân tích tốc độ cao. Đặc biệt, công cụ Apache Superset hỗ trợ trực quan hóa dữ liệu một cách trực quan và dễ hiểu thông qua các biểu đồ và bảng điều khiển (dashboard), giúp người dùng cuối có thể nhận diện xu hướng thị trường lao động, các kỹ năng phổ biến và sự thay đổi nhu cầu tuyển dụng.

Với sự tích hợp giữa Hadoop HDFS và các công nghệ xử lý dữ liệu như Apache Spark, hệ thống này không chỉ đáp ứng khả năng lưu trữ và xử lý dữ liệu lớn mà còn cung cấp các kết quả phân tích chính xác và kịp thời. Doanh nghiệp có thể nắm bắt nhu cầu tuyển dụng và tối ưu hóa chiến lược nhân sự. Người lao động có thể hiểu rõ các xu hướng kỹ năng cần thiết để nâng cao khả năng cạnh tranh. Cơ quan quản lý có thể xây dựng chính sách phát triển nguồn nhân lực dựa trên dữ liệu đáng tin cậy và phân tích chi tiết.

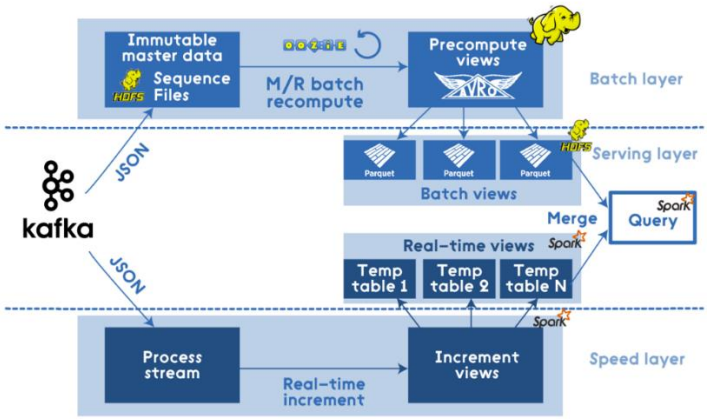
Hệ thống này không chỉ giải quyết vấn đề lưu trữ và xử lý dữ liệu lớn mà còn mở ra cơ hội dự đoán và đánh giá xu hướng thị trường lao động, từ đó thúc đẩy sự phát triển bền vững của thị trường lao động trong tương lai.

3. Phạm vi và giới hạn báo cáo

- **Phạm vi:** Báo cáo tập trung vào việc phân tích dữ liệu việc làm trong bối cảnh xử lý dữ liệu lớn, với mục tiêu phân tích xu hướng thị trường việc làm, phân tích yêu cầu theo ngành nghề, phân tích việc làm theo khu vực, quốc gia và sự chênh lệch, cơ hội trên thị thị trường việc làm ở các khu vực khác nhau.
- **Giới hạn:** Báo cáo sử dụng bộ dữ liệu có sẵn được public trên Kaggle (đường dẫn đến bộ dữ liệu: Link), không bao gồm việc thu thập dữ liệu từ thực địa hoặc các nguồn không chính thống. Kết quả phân tích mang tính tham khảo và đề xuất, không đảm bảo độ chính xác tuyệt đối do các hạn chế về dữ liệu và phương pháp phân tích.

II. Kiến trúc và thiết kế

1. Kiến trúc Lambda



Lambda Architecture là một mô hình kiến trúc dữ liệu phổ biến được thiết kế để **xử lý khối lượng lớn dữ liệu** một cách hiệu quả, đảm bảo **độ trễ thấp** trong truy xuất kết quả và **khả năng mở rộng** linh hoạt cho hệ thống. Mô hình này đặc biệt phù hợp trong các bài toán đòi hỏi cả **xử lý theo lô (batch)** và **xử lý theo thời gian thực (streaming)**, đảm bảo dữ liệu luôn chính xác và kịp thời.

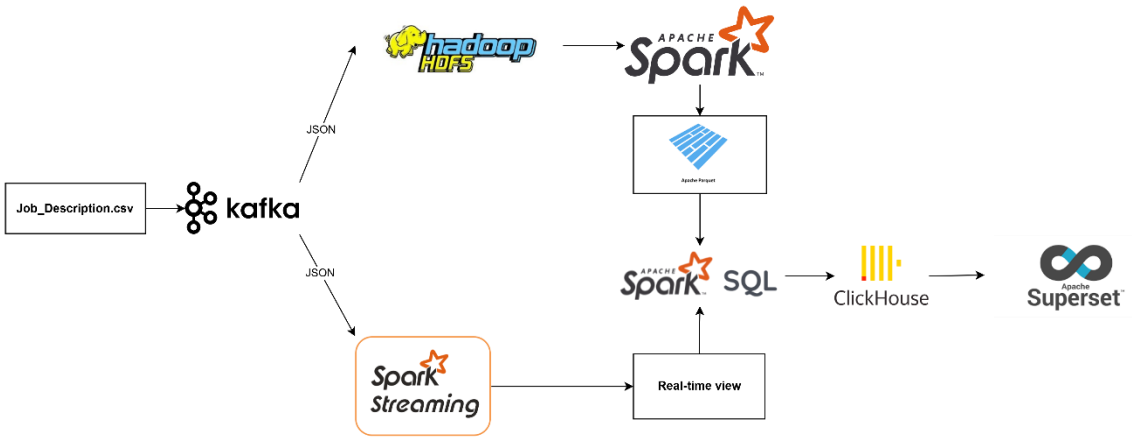
Lambda Architecture bao gồm **ba lớp chính** với các vai trò cụ thể như sau:

- **Batch Layer (Lớp xử lý theo lô):** Lớp này chịu trách nhiệm đảm bảo **tính chính xác và toàn vẹn dữ liệu**, nhờ khả năng xử lý toàn bộ dữ liệu gốc với độ trễ cao hơn nhưng đảm bảo kết quả chính xác.
- **Speed Layer (Lớp xử lý thời gian thực):** Xử lý dữ liệu mới phát sinh trong thời gian thực, giúp bù đắp cho độ trễ của Batch Layer. Lớp này tạo ra các **dữ liệu view tạm thời (real-time views)** để cung cấp kết quả nhanh chóng nhưng có thể kém chính xác hơn so với batch views. Dữ liệu được xử lý với **độ trễ cực thấp** để đáp ứng các yêu cầu phân tích tức thời. Tuy nhiên, kết quả từ Speed Layer chỉ mang tính tạm thời và sẽ được thay thế khi Batch Layer cập nhật.
- **Serving Layer (Lớp phục vụ kết quả):** Tổng hợp và phục vụ kết quả từ cả **Batch Layer** và **Speed Layer** cho người dùng cuối, đảm bảo rằng người dùng luôn nhận được dữ liệu mới nhất và chính xác nhất.

Lý Do Lựa Chọn Lambda Architecture:

- Đảm bảo tính chính xác và cập nhật của dữ liệu
- Khả năng mở rộng và chịu lỗi cao
- Xử lý dữ liệu linh hoạt
- Hiệu suất cao trong truy vấn và trực quan hóa dữ liệu

2. Kiến trúc chi tiết của hệ thống



2.1. Batch layer

- **Thu thập và Lưu trữ**:
  - Dữ liệu được thu thập từ Kafka và lưu trữ dưới dạng immutable trên hệ thống tệp phân tán HDFS, đảm bảo dữ liệu luôn sẵn sàng cho việc tái tính toán.
- **Tái Tính Toán Dữ Liệu (Batch Recompute) với Apache Spark:**
  - Apache Spark được sử dụng cho quá trình tái tính toán toàn bộ dữ liệu. Spark tận dụng khả năng xử lý song song và in-memory, giúp tăng tốc độ xử lý.
  - Quá trình bao gồm các bước:
    - Tiền xử lý dữ liệu: Làm sạch, loại bỏ dữ liệu trùng lặp và chuẩn hóa dữ liệu.
    - Chạy các tác vụ tính toán: Sử dụng Spark SQL, ví dụ như tổng số lượng việc làm theo từng thành phố.
- **Lưu Trữ Kết Quả - Precompute Views:**
  - Kết quả sau khi tính toán được lưu dưới dạng Parquet hoặc Avro trên HDFS, giúp tối ưu hóa truy vấn và nén dữ liệu.
  - Các batch views chứa dữ liệu tổng hợp theo từng khoảng thời gian (ngày, tuần), phục vụ cho các báo cáo từ lớp Serving Layer.

2.2. Speed layer

- **Thu thập Dữ liệu Thời Gian Thực**:
  - Dữ liệu được thu thập từ các nguồn streaming như Apache Kafka, nơi dữ liệu đầu vào thường ở định dạng JSON. Kafka đóng vai trò như một hàng đợi tin nhắn (message queue), giúp truyền tải dữ liệu nhanh chóng đến lớp xử lý.
- **Xử Lý Luồng Dữ Liệu (Stream Processing) với Apache Spark Streaming**:
  - Quá trình xử lý bao gồm các bước:

        - Nạp dữ liệu từ Kafka: Spark Streaming kết nối với Kafka để lấy dữ liệu theo thời gian thực.
        - Tiền xử lý dữ liệu: Loại bỏ dữ liệu không hợp lệ và chuẩn hóa dữ liệu.
        - Tính toán theo thời gian thực: Áp dụng các phép tính như tổng hợp, phân nhóm
- **Tạo và Cập Nhật Real-time Views**:
    - Real-time Views: Kết quả xử lý được lưu trữ tạm thời trong các bảng tạm (Temp Tables) hoặc in-memory tables, cho phép truy vấn nhanh chóng.
    - Incremental Updates: Dữ liệu được cập nhật liên tục (incremental updates) để phản ánh những thay đổi mới nhất từ luồng dữ liệu.

2.3. Serving layer

- **Hợp nhất dữ liệu bằng SparkSQL**:
    - SparkSQL được sử dụng để merge dữ liệu từ Batch views Real-time views.
    - Kết quả hợp nhất sẽ tạo ra một bảng analytical view với dữ liệu được cập nhật liên tục.
- **Lưu trữ kết quả vào ClickHouse**:
    - Kết quả từ SparkSQL được insert vào ClickHouse. ClickHouse cung cấp khả năng lưu trữ dạng cột (columnar storage), giúp tăng tốc độ truy vấn và tối ưu hóa không gian lưu trữ.
- **Trực quan hóa dữ liệu với Apache Superset**:
    - Apache Superset kết nối trực tiếp với ClickHouse, cho phép truy vấn và tạo các biểu đồ trực quan.
    - Các biểu đồ phổ biến trên Apache Superset:
        - Time Series Chart: Hiển thị xu hướng số lượng công việc qua thời gian (total_jobs_over_time).
        - Bar Chart: Thể hiện số lượng công việc theo từng thành phố (job_count_by_city).
        - Pie Chart: Phân tích tỷ lệ phần trăm công việc theo ngành nghề (job_category_distribution).

III. Chi tiết triển khai

1. Kafka

```
zookeeper:
  image:                                    confluentinc/cp-zookeeper:7.3.2
  environment:
    ZOOKEEPER_CLIENT_PORT:                                              2181
```

```
    ZOOKEEPER_TICK_TIME: 2000
  networks:
    - mynetwork

kafka-server:
  image: confluentinc/cp-kafka:7.3.2
  ports:
    - "9092:9092"
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-
server:29092,PLAINTEXT_HOST://localhost:9092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
  depends_on:
    - zookeeper
  networks:
    - mynetwork
```

2. Hadoop HDFS

```
hadoop-namenode:
  image: apache/hadoop:3.3.5
  container_name: hadoop-namenode
  hostname: hadoop-namenode
  user: root
  environment:
    - HADOOP_HOME=/opt/hadoop
  volumes:
    - ./hadoop-namenode:/opt/hadoop/data/nameNode
    - ./hadoop-config:/opt/hadoop/etc/hadoop
    - ./start-hdfs.sh:/start-hdfs.sh
  ports:
    - "9870:9870"
    - "9000:9000"
  command: [ "/bin/bash", "/start-hdfs.sh" ]
  networks:
    - mynetwork

hadoop-datanode-1:
  image: apache/hadoop:3.3.5
  container_name: hadoop-datanode-1
  hostname: hadoop-datanode-1
  user: root
  environment:
    - HADOOP_HOME=/opt/hadoop
  volumes:
    - ./hadoop-datanode-1:/opt/hadoop/data/dataNode
    - ./hadoop-config:/opt/hadoop/etc/hadoop
    - ./init-datanode.sh:/init-datanode.sh
```

```
  ports:
    - "9864:9864"
  depends_on:
    - hadoop-namenode
  command: ["/bin/bash", "/init-datanode.sh"]
  networks:
    - mynetwork

hadoop-datanode-2:
  image: apache/hadoop:3.3.5
  container_name: hadoop-datanode-2
  hostname: hadoop-datanode-2
  user: root
  environment:
    - HADOOP_HOME=/opt/hadoop
  volumes:
    - ./hadoop-datanode-2:/opt/hadoop/data/dataNode
    - ./hadoop-config:/opt/hadoop/etc/hadoop
    - ./init-datanode.sh:/init-datanode.sh
  ports:
    - "9865:9864"
  depends_on:
    - hadoop-namenode
  command: ["/bin/bash", "/init-datanode.sh"]
  networks:
    - mynetwork
```

3. Spark

```
spark-master:
  image: docker.io/bitnami/spark:3.5
  environment:
    - SPARK_MODE=master
    - SPARK_RPC_AUTHENTICATION_ENABLED=no
    - SPARK_RPC_ENCRYPTION_ENABLED=no
    - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
    - SPARK_SSL_ENABLED=no
    - SPARK_MASTER_HOSTNAME=spark-master
    - SPARK_MASTER_PORT=7077
  ports:
    - "4040:4040"
    - "6066:6066"
    - "7077:7077"
    - "8080:8080"
  networks:
    - spark-network

spark-worker-1:
  image: docker.io/bitnami/spark:3.5
  environment:
    - SPARK_MODE=worker
    - SPARK_MASTER_URL=spark://spark-master:7077
```

```
  - SPARK_WORKER_MEMORY=4G
  - SPARK_WORKER_CORES=4
  - SPARK_RPC_AUTHENTICATION_ENABLED=no
  - SPARK_RPC_ENCRYPTION_ENABLED=no
  - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
  - SPARK_SSL_ENABLED=no
 ports:
  - "8081:8081"
 depends_on:
  - spark-master
 networks:
  - spark-network

spark-worker-2:
 image: docker.io/bitnami/spark:3.5
 environment:
  - SPARK_MODE=worker
  - SPARK_MASTER_URL=spark://spark-master:7077
  - SPARK_WORKER_MEMORY=4G
  - SPARK_WORKER_CORES=4
  - SPARK_RPC_AUTHENTICATION_ENABLED=no
  - SPARK_RPC_ENCRYPTION_ENABLED=no
  - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
  - SPARK_SSL_ENABLED=no
 ports:
  - "8082:8082"
 depends_on:
  - spark-master
 networks:
  - spark-network
```

4. ClickHouse

5. Superset

```
superset:
 image: apache/superset:latest
 container_name: superset
 restart: always
 environment:
  - SUPERSET_SECRET_KEY=supersecretkey123
  -
DATABASE_URL=postgresql+psycopg2://superset:supersetpassword@postgres:5432/sup
erset
  - REDIS_URL=redis://redis:6379/0
 volumes:
  - ./superset-home:/app/superset_home
 depends_on:
  - postgres
  - redis
 ports:
```

```
    - "8088:8088"
  command: >
    bash -c "
    pip install clickhouse-connect psycopg2-binary pymongo && superset db upgrade &&
    superset fab create-admin --username admin --firstname Admin --lastname Admin --email
admin@example.com --password admin &&
    superset load_examples &&
    superset init &&
    superset run -h 0.0.0.0 -p 8088
    "
  networks:
    - mynetwork

postgres:
  image: postgres:13
  container_name: superset_postgres
  restart: always
  environment:
    POSTGRES_USER: superset
    POSTGRES_PASSWORD: supersetpassword
    POSTGRES_DB: superset
  volumes:
    - ./postgres-data:/var/lib/postgresql/data
  ports:
    - "5432:5432"
  networks:
    - mynetwork

redis:
  image: redis:latest
  container_name: superset-redis
  restart: always
  ports:
    - "6379:6379"
  networks:
    - mynetwork
```

IV. Bài học kinh nghiệm

Kinh nghiệm 1: Xử lý nhiều nguồn dữ liệu đa dạng

Mô tả vấn đề

- **Context và background:** Hệ thống cần thu thập dữ liệu từ nhiều nguồn khác nhau như CSV, JSON, và dữ liệu phi cấu trúc.
- **Thách thức gặp phải:** Sự không đồng nhất về định dạng dữ liệu, thiếu dữ liệu và tốc độ ingest chậm gây ảnh hưởng tới pipeline.
- **Impact với hệ thống:** Pipeline bị gián đoạn và mất thời gian xử lý làm chậm trễ kết quả.

Các giải pháp đã thử

- **Approach 1:** Sử dụng Apache Spark để chuyển đổi và chuẩn hóa dữ liệu.
    - **Trade-offs:** Tốn thời gian xử lý với lượng dữ liệu lớn.
- **Approach 2:** Áp dụng Kafka để ingest dữ liệu streaming và tích hợp pre-processor.
    - **Trade-offs:** Phức tạp khi xử lý dữ liệu phi cấu trúc.

Giải pháp cuối cùng

- **Chi tiết giải pháp:** Sử dụng Kafka cho dữ liệu streaming và Spark để tiền xử lý. Lưu kết quả vào HDFS hoặc ClickHouse tùy vào yêu cầu.
- **Implementation details:**
    - Kafka làm nguồn chính để xử lý dữ liệu đa dạng.
    - Spark xử lý batch và streaming để chuẩn hóa dữ liệu.

Bài học rút ra

- **Technical insights:** Kết hợp Kafka và Spark giúp linh hoạt trong xử lý dữ liệu.
- **Best practices:** Chuẩn hóa dữ liệu ngay từ pipeline ingest.
- **Recommendations:** Sử dụng các công cụ xử lý batch và stream phù hợp.

Kinh nghiệm 2: State management trong Stream Processing

Mô tả vấn đề

- **Context và background:** Khi xử lý dữ liệu thời gian thực với Spark Streaming, quản lý state cho các phép toán aggregation và windowing trở nên phức tạp.
- **Thách thức gặp phải:** State lớn làm tăng độ trễ và tiêu tốn nhiều bộ nhớ.
- **Impact với hệ thống:** Giảm hiệu suất và có nguy cơ mất state khi job gặp sự cố.

Các giải pháp đã thử

- **Approach 1**: Sử dụng in-memory state mặc định của Spark Streaming.
    - **Trade-offs**: Phù hợp cho state nhỏ nhưng dễ gặp vấn đề khi state lớn.
- **Approach 2:** Ghi state ra HDFS/Checkpointing.
    - **Trade-offs:** Tăng độ bền nhưng giảm tốc độ truy cập state.

Giải pháp cuối cùng

- **Chi tiết giải pháp:**
    - Kết hợp checkpointing để lưu state trên HDFS.

    - Giảm kích thước state thông qua việc loại bỏ key không cần thiết bằng TTL (Time to Live).
- **Implementation details:**
    - Enable checkpointing: sparkContext.setCheckpointDir("hdfs://path/to/checkpoints").
    - Sử dụng TTL để giảm state size

Bài học rút ra

- **Technical insights:** Checkpointing là giải pháp bền vững cho state lớn.
- **Best practices:** Cấu hình TTL để giữ state ở mức tối thiểu.
- **Recommendations:** Monitor state size để tránh tràn bộ nhớ.

Kinh nghiệm 3: Lựa chọn storage format trong Data Storage

Mô tả vấn đề

- **Context và background:** Hệ thống cần lưu trữ dữ liệu lớn trên HDFS để phục vụ cho batch và real-time queries.
- **Thách thức gặp phải:** Lựa chọn định dạng dữ liệu ảnh hưởng đến tốc độ truy vấn và dung lượng lưu trữ.
- **Impact với hệ thống:** Tốc độ đọc/ghi chậm và tốn tài nguyên lưu trữ.

Các giải pháp đã thử

- **Approach 1**: Sử dụng định dạng CSV.
    - **Trade-offs:** Đơn giản nhưng không hỗ trợ nén tốt và chậm khi truy vấn.
- **Approach 2:** Sử dụng Parquet với nén Snappy.
    - **Trade-offs:** Tăng tốc truy vấn nhưng cần nhiều tài nguyên xử lý hơn khi nén/giải nén.

Giải pháp cuối cùng

- **Chi tiết giải pháp:** Chọn Parquet cho batch storage trên HDFS. Kết hợp nén Snappy để tối ưu dung lượng.
- **Implementation details:**
    - Lưu batch data: write.format("parquet").option("compression", "snappy").

Bài học rút ra

- **Technical insights**: Parquet và Avro là lựa chọn tối ưu cho tốc độ và dung lượng.

- **Best practices:** Chọn định dạng dựa trên loại workload.
- **Recommendations:** Luôn bật nén để tiết kiệm dung lượng.

Kinh nghiệm 4: Cấu hình thời gian lưu log và kích thước data trong Kafka

Mô tả vấn đề

- **Context và background:** Hệ thống sử dụng Kafka để thu thập và truyền tải dữ liệu từ nhiều nguồn. Các logs lưu trữ trong Kafka chiếm dụng nhiều dung lượng ổ đĩa khi dữ liệu liên tục được ghi vào.
- **Thách thức gặp phải:**
  - Dung lượng ổ đĩa của brokers nhanh chóng bị đầy.
  - Dữ liệu cũ không cần thiết vẫn còn lưu trữ, làm giảm hiệu suất.
  - Cấu hình không hợp lý có thể gây mất dữ liệu hoặc ngắt quãng pipeline.
- **Impact với hệ thống:** Nếu logs không được xóa đúng thời gian hoặc giới hạn kích thước không phù hợp, có thể dẫn đến crash broker hoặc mất dữ liệu trong production.

Các giải pháp đã thử

- **Approach 1:** Giảm thời gian lưu log (log.retention.hours) xuống thấp hơn.
  - **Trade-offs:** Giải phóng dung lượng nhanh hơn nhưng có nguy cơ mất dữ liệu nếu consumers chưa xử lý kịp thời.
- **Approach 2:** Giới hạn kích thước log segment (log.segment.bytes) và tổng dung lượng lưu trữ (log.retention.bytes).
  - **Trade-offs:** Tối ưu kích thước lưu trữ nhưng cần tính toán kỹ dung lượng cho brokers.

Giải pháp cuối cùng

- **Chi tiết giải pháp:**
  - Thiết lập thời gian lưu trữ log (log.retention.hours) hợp lý: 48-72 giờ tùy vào tốc độ xử lý của consumers.
  - Giới hạn kích thước log segment: log.segment.bytes=1GB.
  - Giới hạn tổng kích thước logs trên mỗi broker: log.retention.bytes=100GB.
  - Kết hợp monitoring Kafka metrics để theo dõi trạng thái dung lượng ổ đĩa và tốc độ xử lý consumers.
- **Implementation details:**
  - File server.properties trên Kafka brokers:

- log.retention.hours=72
- log.segment.bytes=1073741824
- log.retention.bytes=107374182400

Bài học rút ra

- **Technical insights:** Cấu hình log.retention và log.segment.bytes là chìa khóa để tối ưu hiệu suất Kafka và đảm bảo tính ổn định.
- **Best practices:**
  - Tính toán dung lượng lưu trữ dựa trên tốc độ ghi dữ liệu và thời gian xử lý của consumers.
  - Sử dụng monitoring tools như Prometheus hoặc JMX Exporter để theo dõi dung lượng ổ đĩa và tốc độ xử lý logs.
- **Recommendations:** Cấu hình thời gian lưu trữ và kích thước logs một cách linh hoạt dựa trên SLAs và yêu cầu hệ thống.

Kinh nghiệm 5: Ghi file Parquet vào HDFS với AvroParquetWriter trong Java

Mô tả vấn đề

- **Context và background:** Trong hệ thống Java-based, sử dụng AvroParquetWriter để ghi dữ liệu vào HDFS dưới định dạng Parquet.
- **Thách thức gặp phải:**
  - Kích thước file không tối ưu khi ghi dữ liệu lớn, gây ra nhiều file nhỏ.
  - Khó kiểm soát kích thước file Parquet và quản lý tài nguyên.
  - Ghi file song song dẫn đến bất đồng bộ khi xử lý đa luồng.
  - Impact với hệ thống: Giảm hiệu suất đọc/ghi và tốn tài nguyên HDFS NameNode.

Các giải pháp đã thử

- **Approach 1:** Ghi dữ liệu trực tiếp mà không giới hạn kích thước file.
  - **Trade-offs:** Tạo ra rất nhiều file nhỏ, gây quá tải HDFS NameNode.
- **Approach 2:** Sử dụng AvroParquetWriter kết hợp RowGroupSize để kiểm soát kích thước file.
  - **Trade-offs:** Yêu cầu quản lý bộ nhớ và xử lý luồng dữ liệu hiệu quả.

Giải pháp cuối cùng

- **Chi tiết giải pháp:**

- **AvroParquetWriter:** Thiết lập RowGroupSize là 128MB để tối ưu kích thước file Parquet phù hợp với HDFS block size.
- **Ghi dữ liệu đa luồng:** Sử dụng ExecutorService trong Java để ghi song song dữ liệu, đảm bảo đồng bộ hóa khi cần thiết.
- **Snappy compression:** Kết hợp nén dữ liệu để giảm dung lượng lưu trữ mà vẫn duy trì tốc độ ghi.

Bài học rút ra

- **Technical insights:**
  - Tối ưu RowGroupSize là chìa khóa để kiểm soát kích thước file Parquet phù hợp với HDFS block size.
  - Sử dụng ExecutorService để ghi song song giúp tận dụng tối đa tài nguyên hệ thống.
- **Best practices:**
  - Kết hợp nén Snappy để cân bằng giữa tốc độ ghi và dung lượng lưu trữ.
  - Giới hạn số lượng thread song song tùy theo tài nguyên hệ thống (CPU và bộ nhớ).
- **Recommendations:**
  - Thường xuyên kiểm tra kích thước file và giám sát hiệu suất HDFS.
  - Tối ưu song song hóa dựa trên mức tài nguyên hiện có.

V. Kết luận

1. Kết luận

Qua quá trình thực hiện đề tài "Hệ thống lưu trữ và phân tích dữ liệu việc làm", nhóm đã đạt được những kết quả sau:

- Xây dựng thành công một hệ thống phân tích dữ liệu việc làm với kiến trúc hiện đại, tận dụng các công nghệ như Hadoop, Spark, ClickHouse và Superset.
- Thiết lập được quy trình thu thập, xử lý và phân tích dữ liệu tự động, giúp quản lý và khai thác hiệu quả lượng dữ liệu lớn về việc làm.
- Áp dụng thành công các kỹ thuật ETL (Extract-Transform-Load) thông qua Spark để xử lý dữ liệu thô và chuyển đổi thành thông tin có giá trị.
- Xây dựng được các dashboard trực quan thông qua Superset, giúp người dùng dễ dàng theo dõi và phân tích xu hướng việc làm.

2. Hướng phát triển

Để tiếp tục nâng cao hiệu quả của hệ thống, nhóm đề xuất các hướng phát triển sau:

- Mở rộng nguồn dữ liệu: Thu thập dữ liệu từ các nguồn phổ biến và uy tín như Careerviet, TopCV…
- Nâng cao khả năng phân tích: Tích hợp các mô hình học máy để dự đoán xu hướng việc làm, phân vùng việc làm phù hợp với kỹ năng và trình độ của ứng viên
- Cải thiện hiệu năng hệ thống: Tối ưu hóa quy trình ETL để giảm thời gian xử lý, áp dụng các kỹ thuật cache và phân vùng dữ liệu hiệu quả.

Phụ lục

Link github dự án: https://github.com/house3173/BigData_Project_20241