Big Data Analytics Project: Healthcare Data Processing Using MIMIC-III

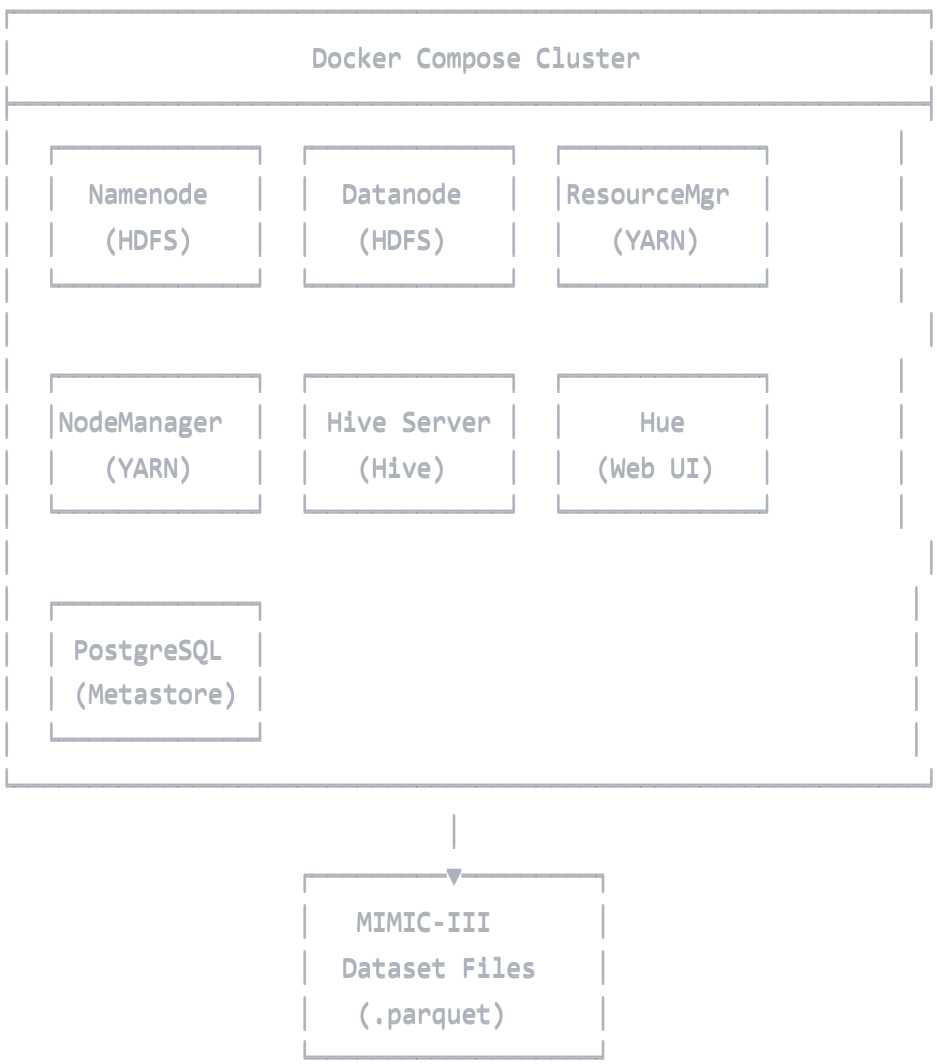
Project Overview

This project implements batch analytics on the dataset MIMIC-III with Hadoop ecosystem including Hive, and MapReduce. The objective is to perform large-scale data processing and analysis on records to extract meaningful insights.

Project Architecture

Repository: https://github.com/yasmindawa/big_data_project

The project follows a distributed computing architecture leveraging containerized services:



Data Flow:

1. MIMIC-III parquet files → Namenode container
2. Files uploaded to HDFS distributed storage
3. Hive creates external tables pointing to HDFS locations
4. Analytics queries executed via Hive SQL and MapReduce
5. Results stored back in HDFS for further analysis

Architecture & Technology Stack

- **Hadoop Distributed File System (HDFS)** - Data storage
- **Apache Hive** - Data warehousing and SQL queries
- **MapReduce** - Distributed computing framework
- **Docker & Docker Compose** - Containerization and orchestration
- **Hue** - Web-based interface for Hadoop ecosystem
- **MIMIC-III Dataset** - Healthcare data in Parquet format

Environment Setup

1. Cluster Initialization

Step 1: Repository Setup

```bash
# Clone the Hadoop/Spark/Hive Docker Compose repository
git clone [repository-url]
```

Step 2: Start the Cluster

```bash
# Build and start the complete Hadoop ecosystem
docker-compose up -d
```

Step 3: Access Services

bash

```
# Access Hive Server
docker exec -it hive-server bash

# Access Namenode for file operations
docker exec -it namenode bash
```

2. Data Ingestion Pipeline

File Transfer to Namenode Container

bash

```
# Copy MIMIC-III parquet files to namenode container
docker cp [local-file-path] namenode:/[container-path]
```

HDFS Directory Structure Creation

bash

```
# Create warehouse directories for each dataset
hdfs dfs -mkdir -p /user/hive/warehouse/admissions
hdfs dfs -mkdir -p /user/hive/warehouse/diagnosis_full
hdfs dfs -mkdir -p /user/hive/warehouse/icustays
hdfs dfs -mkdir -p /user/hive/warehouse/patients
```

Data Upload to HDFS

bash

```
# Upload parquet files to respective HDFS directories
hdfs dfs -put admissions.parquet /user/hive/warehouse/admissions/
hdfs dfs -put diagnosis_full.parquet /user/hive/warehouse/diagnosis_full/
hdfs dfs -put icustays.parquet /user/hive/warehouse/icustays/
hdfs dfs -put patients.parquet /user/hive/warehouse/patients/
```

Database Schema Implementation

Hive Database Setup

sql

```sql
-- Create MIMIC database
CREATE DATABASE IF NOT EXISTS mimic;
USE mimic;
```

Table Definitions

1. Admissions Table

sql

```sql
CREATE EXTERNAL TABLE admissions (
    subject_id INT,
    hadm_id INT,
    admittime STRING,
    dischtime STRING,
    deathtime STRING,
    admission_type STRING,
    admission_location STRING,
    discharge_location STRING,
    insurance STRING,
    language STRING,
    religion STRING,
    marital_status STRING,
    ethnicity STRING,
    edregtime STRING,
    edouttime STRING,
    diagnosis STRING,
    hospital_expire_flag INT,
    has_chartevents_data INT
)
STORED AS PARQUET
LOCATION '/user/hive/warehouse/admissions/';
```

2. Diagnosis Table

sql

```sql
CREATE EXTERNAL TABLE diagnosis_full (
    subject_id INT,
    hadm_id INT,
    seq_num INT,
    icd9_code STRING,
    short_title STRING,
    long_title STRING
)
STORED AS PARQUET
LOCATION '/user/hive/warehouse/diagnosis_full/';
```

3. ICU Stays Table

sql

```sql
CREATE EXTERNAL TABLE icustays (
    row_id INT,
    subject_id INT,
    hadm_id INT,
    icustay_id INT,
    dbsource STRING,
    first_careunit STRING,
    last_careunit STRING,
    first_wardid INT,
    last_wardid INT,
    intime STRING,
    outtime STRING,
    los DOUBLE
)
STORED AS PARQUET
LOCATION '/user/hive/warehouse/icustays/';
```

4. Patients Table

sql

```sql
CREATE EXTERNAL TABLE patients (
    row_id INT,
    subject_id INT,
    gender STRING,
    dob STRING,
    dod STRING,
    dod_hosp STRING,
    dod_ssn STRING,
    expire_flag INT
)
STORED AS PARQUET
LOCATION '/user/hive/warehouse/patients/';
```

Analytics Queries

Query 1: Average Length of Stay per Diagnosis

sql

```sql
SELECT
    d.short_title as diagnosis,
    AVG(DATEDIFF(a.dischtime, a.admittime)) as avg_length_of_stay_days
FROM admissions a
JOIN diagnosis_full d ON a.hadm_id = d.hadm_id
WHERE a.dischtime IS NOT NULL
    AND a.admittime IS NOT NULL
GROUP BY d.short_title
ORDER BY avg_length_of_stay_days DESC
LIMIT 20;
```

Query 2: Patient Demographics Analysis

sql

```sql
SELECT
    gender,
    COUNT(*) as patient_count,
    AVG(YEAR(CURRENT_DATE()) - YEAR(dob)) as avg_age
FROM patients
WHERE dob IS NOT NULL
GROUP BY gender;
```

Query 3: ICU Utilization Patterns

sql

```sql
SELECT
    first_careunit,
    COUNT(*) as total_stays,
    AVG(los) as avg_length_of_stay,
    MIN(los) as min_stay,
    MAX(los) as max_stay
FROM icustays
WHERE los IS NOT NULL
GROUP BY first_careunit
ORDER BY total_stays DESC;
```

MapReduce Implementation

Data Preparation for MapReduce

sql

```sql
-- Create table with MapReduce-compatible format
CREATE TABLE patients_for_mr (
    subject_id INT,
    gender STRING,
    age_at_death INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION '/user/hive/warehouse/patients_for_mr/';

-- Insert processed data
INSERT INTO patients_for_mr
SELECT
    subject_id,
    gender,
    CASE
        WHEN dod IS NOT NULL THEN YEAR(dod) - YEAR(dob)
        ELSE YEAR(CURRENT_DATE()) - YEAR(dob)
    END as age
FROM patients
WHERE dob IS NOT NULL;
```

MapReduce Job Execution

bash

```bash
# Create directories for MapReduce code
mkdir -p ~/avg-age-job
cd ~/avg-age-job

# Compile Java MapReduce classes
javac -classpath $HADOOP_CLASSPATH AverageAge.java
jar cf average-age.jar AverageAge*.class

# Execute MapReduce job
hadoop jar ~/avg-age-job/average-age.jar AverageAge \
    /user/hive/warehouse/patients_for_mr/patients_for_mr \
    /user/hive/warehouse/mimic/average_age_output

# View results
hdfs dfs -cat /user/hive/warehouse/mimic/average_age_output/part-r-00000
```

System Configuration & Troubleshooting

Web UI Configuration

Resource Manager Access Issue Resolution:

1. **Problem**: Resource Manager Web UI not accessible
2. **Solution**: Updated Docker Compose port mapping

yaml

```yaml
resourcemanager:
  ports:
    - "8088:8088"  # Added port mapping for Web UI access
```

Yarn Configuration Update:

xml

```xml
<!-- Added to yarn-site.xml -->
<property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>0.0.0.0:8088</value>
</property>
```

Hue Integration

Docker Compose Configuration for Hue:

```yaml
hue:
  image: gethue/hue:latest
  hostname: hue
  container_name: hue
  dns: 8.8.8.8
  ports:
    - "8888:8888"
  volumes:
    - ./hue-overrides.ini:/usr/share/hue/desktop/conf/hue-overrides.ini
  depends_on:
    - hue-metastore-postgresql
```

Data Persistence Configuration:

```yaml
hue-metastore-postgresql:
  volumes:
    - hue_postgres_data:/var/lib/postgresql/data
```

Data Processing Challenges & Solutions

Timestamp Data Type Issues

Challenge: Parquet files contained timestamp datatypes that caused compatibility issues with Hive.

Solution: Used Python pandas with specific Parquet export parameters to maintain timestamp compatibility:

```python
df.to_parquet('filename.parquet', engine='pyarrow', index=False, use_deprecated_int96_timestamp
```