BIG DATA ANALYTICS TECHNOLOGY

Coursework

Module Learning Outcomes Assessed:

1. Knowledge of big data technologies and principles

2. Apply knowledge on Hadoop, Spark, Kafka, Flink to provide scalable solutions for important big data problems

3. Knowledge on streaming data processing applications

4. Build up skills on learning new technologies and trends on big data processing

Tasks:

The coursework consists of five tasks listed below. You are required to implement and test them in either a CentOS or Ubuntu environment or Docker
For Tasks 1–4, prepare a detailed report describing the steps you followed and the results you obtained. Submit a single report containing your answers for these four tasks. Additionally, provide the source files used for Tasks 1–4 in a compressed ZIP file.

Task 5 is a self-study component that requires you to conduct an in-depth review and provide a summary of the key findings.

Task 1: In-Degree Distribution Analysis using Apache Hadoop and Apache Spark

Objective

Analyze and compare the performance of **Apache Hadoop (MapReduce)** and **Apache Spark** for computing **in-degree distribution** on large-scale graph datasets from the **Stanford SNAP repository**.

Datasets (choose at least three, plus one large dataset for scalability testing)

Use real-world graph datasets from SNAP such as:

- **soc-Pokec** – Social network (Slovakia, friendships) https://snap.stanford.edu/data/soc-Pokec.html
- **email-EuAll** – Email communication network https://snap.stanford.edu/data/email-EuAll.html
- **cit-Patents** – US patent citation network https://snap.stanford.edu/data/cit-Patents.html
- **web-BerkStan** – Web graph of Berkeley and Stanford https://snap.stanford.edu/data/web-BerkStan.html
- **soc-LiveJournal1** – Large social network (for scalability testing) https://snap.stanford.edu/data/soc-LiveJournal1.html

Part 1: Implementation and Performance Comparison

1. Implement **in-degree distribution computation** separately using:
    - **Apache Hadoop (MapReduce)**
    - **Apache Spark**
2. Run experiments on at least **three datasets** from the list.
3. Record and analyze the following:
    - In-degree distribution plots (e.g., scatter or log-log plots)
    - Performance metrics:
        - Execution time
        - Memory usage
        - CPU utilization
        - Disk I/O and network overhead
4. Compare both systems based on:

- Correctness of results
- Execution performance
- System design and data processing approach

Part 2: Scalability and Optimization Analysis

1. Extend your implementation to a **larger dataset (soc-LiveJournal1)** to test scalability.
2. Evaluate how performance metrics change as dataset size increases. Identify bottlenecks such as:
   - Disk I/O
   - Memory usage
   - Network shuffle overhead
3. Apply **one optimization** to each system (e.g., data partitioning, caching, or configuration tuning) and measure its effect.
4. Write a **critical analysis** discussing:
   - Why Hadoop and Spark show different performance patterns
   - Which system is better suited for processing **large-scale graph data**
   - How experimental findings align with theoretical complexity and system design principles

Task 2: Real-Time IoT Sensor Data Analytics using Apache Kafka

Objective

Design and implement a **real-time IoT sensor data streaming pipeline** using **Apache Kafka**. The goal is to stream, process, and visualize live traffic sensor data to analyze urban traffic behavior and system performance.

Part 1: Setup and Environment Configuration

1. **Install and Configure Kafka**
    - Set up **Apache Kafka** (and **Zookeeper**, if required) on your local or cloud-based Ubuntu environment.
    - Create Kafka topics for:
        - Raw sensor data ingestion
        - Processed or aggregated data outputs
2. **Integration and Tools**
    - (Optional) Configure **Kafka Connect** for data ingestion or external system integration.
    - Prepare a **Grafana** dashboard environment for visualization.
    - Verify message flow by sending sample test data to Kafka topics.

Part 2: Data Source and Preprocessing

1. **Dataset Selection**
   Use the **Traffic count data collected from the several GRIDSMART optical traffic detectors deployed by the City of Austin (2025)**:
   https://data.austintexas.gov/Transportation-and-Mobility/Camera-Traffic-Counts/sh59-i6y9/about_data

2. **Data Preparation**
    - Download and explore the dataset (CSV format).
    - Clean or preprocess the data if necessary (e.g., handle missing values, convert timestamps).
    - Convert the dataset into a Kafka-friendly format (e.g., JSON or Parquet).
    - Simulate **real-time streaming** by publishing sensor readings periodically (e.g., every 5 seconds) using a Kafka **producer** script.

Part 3: Streaming Data Processing and Analysis

1. **Data Ingestion**
    - Use a **Kafka producer** to continuously send sensor readings to a Kafka topic.
2. **Processing and Computation**
    - Develop a **Kafka consumer** or a **Kafka Streams application** to compute and update real-time metrics:
        - **Hourly average vehicle count per sensor**
        - **Daily peak traffic volume across all sensors**
        - **Daily sensor availability (%)** — based on data presence or gaps
3. **Output and Persistence**
    - Write processed data to another Kafka topic for visualization.
    - Optionally processed results persist into a storage system (e.g., PostgreSQL, HDFS, or MinIO) for historical analysis.

Part 4: Visualization and Reporting

1. **Visualization**

- Connect **Grafana** to Kafka (directly or through a data sink such as a database).
- Create real-time dashboards including:
  - Time-series charts for traffic flow per sensor
  - Bar or line charts for daily peak and average volumes
  - Gauge indicators or tables for sensor availability and health

2. **Reporting**
   - Document each step in detail:
     - Environment setup
     - Kafka topic design and configuration
     - Producer and consumer logic
     - Metric Computation Methodology
     - Grafana dashboard setup and interpretation
   - Include screenshots of results, sample data visualizations, and performance observations.
   - Package all **code, configuration files, and documentation** into a labeled ZIP file for submission.

Task 3: Graph Databases using Neo4j and Docker

Objective

Learn how **graph databases** work, when to use them, and how to implement a simple real-world example using **Neo4j** in a **Docker container**.

Part 1 — Understanding Graph Databases

1. In your own words, explain **two or three real-world scenarios** where using a **graph database** is better than using a traditional relational database.
    - Example domains: social networks, fraud detection, recommendation systems, knowledge graphs, supply chains, etc.
2. For each scenario, describe briefly:
    - What kind of data would be stored (nodes and relationships)
    - Why graph structure helps (what's easier or faster to do)
    - One example of a useful query (e.g., find connections, shortest path, related items)

Part 2 — Implementation with Neo4j (Docker Setup)

Choose **one scenario** from Part 1 and build a small demo using **Neo4j in Docker**.

Follow these steps:

1. **Run Neo4j with Docker**
2. docker pull neo4j:latest
3. docker run -d \
4. --name neo4j-graphdb \
5. -p7474:7474 -p7687:7687 \
6. -e NEO4J_AUTH=neo4j/test123 \

7. neo4j:latest

Access Neo4j Browser at: http://localhost:7474
Login: **neo4j / test123**

8. **Create a small dataset** (10–30 nodes)
    - Example (Social Network):
    - CREATE (:User {name:'Alice'})-[:FOLLOWS]->(:User {name:'Bob'}),
    - (:User {name:'Bob'})-[:FOLLOWS]->(:User {name:'Carol'}),
    - (:User {name:'Carol'})-[:FOLLOWS]->(:User {name:'Alice'});
9. **Write three Cypher queries** that show interesting insights from your graph, such as:
    - Find all users connected to a specific person
    - Find the most connected node
    - Find shortest paths or indirect connections
10. **Show your results**
    - Take screenshots of the Neo4j Browser view and query results
    - Briefly explain what each query demonstrates

Task 4: Watermarks in Real-Time Stream Processing using Apache Kafka and Apache Flink (Docker Setup)

Objective

Learn how **watermarks** help manage **out-of-order events** in real-time streaming systems, and explore their impact on **accuracy and performance** using **Apache Kafka** and **Apache Flink** running in **Docker containers**.

Part 1 — Understanding Watermarks

1. In your own words, explain:
    - What are **watermarks** in stream processing?
    - Why are they needed when processing **real-time event streams** (such as social media data)?
2. Describe and compare **three main types** of watermark generation strategies:
    - **Periodic watermarks**
    - **Punctuated watermarks**
    - **Event-time-based watermarks**
3. Discuss briefly how each method handles **late or out-of-order events**, and which method might be best for real-world social media streams.

Part 2 — Implementation using Docker (Kafka + Flink)

Step 1 — Environment Setup

1. Install **Docker** and **Docker Compose** on your Ubuntu system.
2. Create a docker-compose.yml file that runs:
    - **Kafka** and **Zookeeper**
    - **Flink JobManager** and **Flink TaskManager**

3. Verify containers are running correctly:
4. docker ps

Step 2 — Data and Kafka Topics

1. Download two small social media datasets from:
   🔗 https://github.com/luminati-io/Social-media-dataset-samples
2. Create **two Kafka topics**, each for one dataset (initially one partition each).
3. Use a Kafka producer (inside the container or from your host machine) to send tweet data to the topics.

Step 3 — Flink Stream Processing

1. Write **two Flink streaming applications** that:
   - Read data from the two Kafka topics
   - Count the number of tweets containing a **specific hashtag** every **15 seconds**
   - Use **Flink's built-in watermarks** for handling **out-of-order events**
2. Run both applications inside the **Flink Docker environment** (use docker exec or Flink Web UI).
3. Compare and discuss:
   - **Accuracy** of hashtag counts
   - **Performance** (latency, throughput, CPU/memory usage)

Step 4 — Scaling Experiment (Optional Bonus)

1. Update each Kafka topic to **two partitions**.
2. Enable **Kafka-partition-aware watermarking** in Flink.
3. Observe how results change in terms of accuracy and performance under higher parallelism.

Task 5: Self Study – Big Data Technologies in Large Language Model Development

Investigate the latest research and trends regarding the application of big data technologies in the development, verification, and deployment of large language models (LLMs). Specifically, explore how big data concepts and technologies are utilized in the following areas:

- **Development**: Examine the role of big data in training LLMs, including data collection, preprocessing, and handling large-scale datasets.
- **Verification**: Investigate methodologies for validating LLMs, focusing on data-driven approaches for model evaluation and performance assessment.
- **Deployment**: Analyze the deployment strategies of LLMs in production environments, considering aspects like scalability, real-time processing, and integration with big data systems.

Prepare a comprehensive summary report based on your findings, referencing a minimum of five recent research papers published in top-tier international conferences. Ensure that your report includes proper academic citations and provides a critical analysis of the current trends and challenges in this domain.