Indian Institute of Technology Jodhpur

Fundamentals of Distributed Systems

Assignment – 1

Total Marks:	20
Submission Deadline:	23 June 2025

1. Vector Clocks and Causal Ordering

[10 Marks]

Objective

To move beyond simple event ordering by implementing **Vector Clocks** to capture the causal relationships between events in a distributed system. You will apply this to build a causally consistent, multi-node key-value store.

Problem Description

You will build a distributed key-value store with three or more nodes. The key challenge is to ensure that writes to the store are **causally ordered**. If event B is causally dependent on event A (e.g., a value is read and then updated), all nodes must process event A before they process event B. Simple Lamport clocks are insufficient for this, so you will use Vector Clocks.

Technology Constraints

- **Programming Language:** The entire application logic for the nodes and client must be written exclusively in **Python**.
- **Containerization:** The system must be containerized and orchestrated solely using **Docker** and **Docker Compose**.

Tasks

Your implementation should cover the following tasks:

1. **Node Implementation with Vector Clocks:** Create a Python script for a node. Each node must maintain its own local key-value data and a Vector Clock.
2. **Vector Clock Logic:** Implement the rules for incrementing the clock on local events, including the clock in sent messages, and updating the local clock upon receiving a message.
3. **Causal Write Propagation:** Implement the Causal Delivery Rule. When a node receives a replicated write message, it must delay processing that write until the causal dependencies are met by checking the message's vector clock against its own. Messages that cannot be delivered must be buffered.
4. **Containerization and Networking:** Write a 'Dockerfile' for your node and a 'docker-compose.yml' file to run a 3-node system.
5. **Verification and Scenario Testing:** Create a client script and a specific test scenario to prove that your system maintains causal consistency, even when messages arrive out of order.

Deliverables

Your final submission must be a single Git repository containing all the required files organized in the exact structure shown below.

Folder and File Structure

```
vector-clock-kv-store/
|
|-- src/
|    |-- node.py
|    `-- client.py
|
|-- Dockerfile
|-- docker-compose.yml
`-- project_report.pdf
```

Project Report Requirements

You must submit a thorough **Project Report** that details your work.

- **Format:** The report must be a typed document (e.g., created in Microsoft Word, Google Docs, or LaTeX) and submitted as a single 'project_report.pdf' file. **Handwritten reports will not be accepted.**
- **Content:** The report should detail your architecture and implementation and use logs with screenshots to prove that your causal consistency test works correctly.
- **Video Link:** The report must include a publicly accessible link to a short (max 3-minute) video demonstrating the project.

2. Dynamic Load Balancing for a Smart Grid

[10 Marks]

Objective

To design and build a scalable system for a Smart Grid that dynamically balances Electric Vehicle (EV) charging requests across multiple substations based on their real-time load, complete with a comprehensive observability stack.

Problem Description

You will build a system that simulates a Smart Grid managing charging requests from a fleet of EVs. The primary challenge is to prevent overloading any single charging substation. The system must intelligently distribute incoming charging requests to the **least loaded** substation, ensuring grid stability and efficient resource usage. You will use an industry-standard monitoring stack to measure and visualize key performance indicators.

Technology Constraints

- **Programming Language:** All custom services must be written exclusively in **Python**.
- **Containerization & Orchestration:** The entire system must be defined, configured, and run using **Docker** and **Docker Compose**.

Tasks

Your implementation should cover the following tasks:

1. **Microservice Development:** Create two services: a 'charge_request_service' as the public entry point and a 'substation_service' that simulates charging. Instrument the substation to expose its current load as a Prometheus metric.
2. **Custom Dynamic Load Balancer:** Build a new service that acts as the grid's core logic. It must periodically poll the '/metrics' endpoint of each substation to get its current load and use this data to decide where to route new requests.
3. **Observability Stack:** Configure Prometheus to scrape the substation metrics and Grafana to visualize the load on a dashboard.
4. **Containerization & Orchestration:** Write 'Dockerfile's for all services and a 'docker-compose.yml' file to run the entire system, including multiple replicas of the substation service.
5. **Load Testing and Analysis:** Create a Python script to simulate a "rush hour" of EV charging requests and analyze the system's response on your Grafana dashboard.

Deliverables

Your final submission must be a single Git repository containing all the required files organized in the exact structure shown below.

Folder and File Structure

```
smart-grid-load-balancer/
|
|-- charge_request_service/
|    |-- main.py
|    `-- Dockerfile
|
|-- load_balancer/
|    |-- main.py
|    `-- Dockerfile
|
|-- substation_service/
|    |-- main.py
|    `-- Dockerfile
|
|-- load_tester/
|    `-- test.py
|
|-- monitoring/
```

```
|    |-- prometheus/
|    |    `-- prometheus.yml
|    `-- grafana/
|         `-- dashboard.json
|
|-- docker-compose.yml
`-- project_report.pdf
```

Project Report Requirements

You must submit a thorough **Project Report** that details your work.

- **Format:** The report must be a typed document and submitted as a single project_report.pdf file. **Handwritten reports will not be accepted.**
- **Content:** The report should detail your architecture and analyze the system's performance using screenshots from your Grafana dashboard during the load test.
- **Video Link:** The report must include a publicly accessible link to a short (max 3-minute) video demonstrating the project in action.