Experiment No 09

AIM

To simulate and evaluate the performance of different orchestrator policies in a **two-tier edge-cloud computing environment** using the **EdgeCloudSim simulator**. The simulation measures **task completion, failure rates, service times, network delays, and server utilization** across various scenarios and device counts to understand the impact of different task scheduling and orchestration strategies.

TOOLS

- **Simulator:** EdgeCloudSim
- **Software:** Eclipse IDE

THEORY

Edge-Cloud Computing Overview

In modern IoT systems, billions of devices generate massive amounts of data. Processing all of this in the **cloud** alone leads to issues such as **latency, bandwidth overload, and single-point dependency**. To overcome these problems, **edge-cloud computing** integrates both **edge devices** (closer to data sources) and **cloud servers** (highly powerful and centralized).

- **Edge devices** provide **low-latency** responses by processing tasks locally.
- **Cloud servers** handle **heavy workloads** and long-term storage.
- An **orchestrator** decides whether a task should run on the edge or the cloud to optimize performance.

Thus, effective orchestration policies are essential to balance **speed, resource usage, energy efficiency, and reliability**.

EdgeCloudSim Simulator

EdgeCloudSim is a simulation tool designed to model and evaluate **edge-cloud computing scenarios**. It extends CloudSim by incorporating features specific to **edge computing**, such as mobility, networking, and orchestration.

The simulator is built around **five main modules**:

1. **Mobility Module**
    - Tracks the location of mobile edge devices and clients.
    - Provides default nomadic mobility model (users move randomly), but can be customized by extending the MobilityModel class.

2. **Load Generator Module**
    - Generates tasks according to predefined patterns.
    - By default, tasks follow a **Poisson distribution** (active/idle cycles).
    - Custom load generation can be added by extending the LoadGeneratorModel class.
3. **Networking Module**
    - Simulates **transmission delays** in WLAN and WAN networks.
    - Default model uses a **single server queue** for both uploads and downloads.
    - Advanced users can extend the NetworkModel class for realistic network models.
4. **Core Simulation Module**
    - Loads configurations and runs the simulation scenarios.
    - Provides a **logging mechanism** to save results, usually in **CSV format**.
    - Supports different parameters like number of devices, task sizes, mobility patterns, etc.
5. **Edge Orchestrator Module**
    - Implements different orchestration strategies (e.g., random, latency-aware, resource-aware).
    - Decides whether a task runs on **edge or cloud**.
    - Greatly influences system performance.

Extensibility

- **Factory Design Pattern** is used for modularity.
- Users can easily **plug in custom implementations** of mobility, networking, load generation, and orchestration policies by defining their own **Scenario Factory**.
- This allows researchers to test new strategies without modifying the simulator's core.

Parameters Measured in Simulation

- **Task Completion Rate** – Percentage of tasks successfully executed.
- **Failure Rate** – Number of tasks dropped or delayed beyond tolerance.
- **Service Time** – Average time to complete a task.

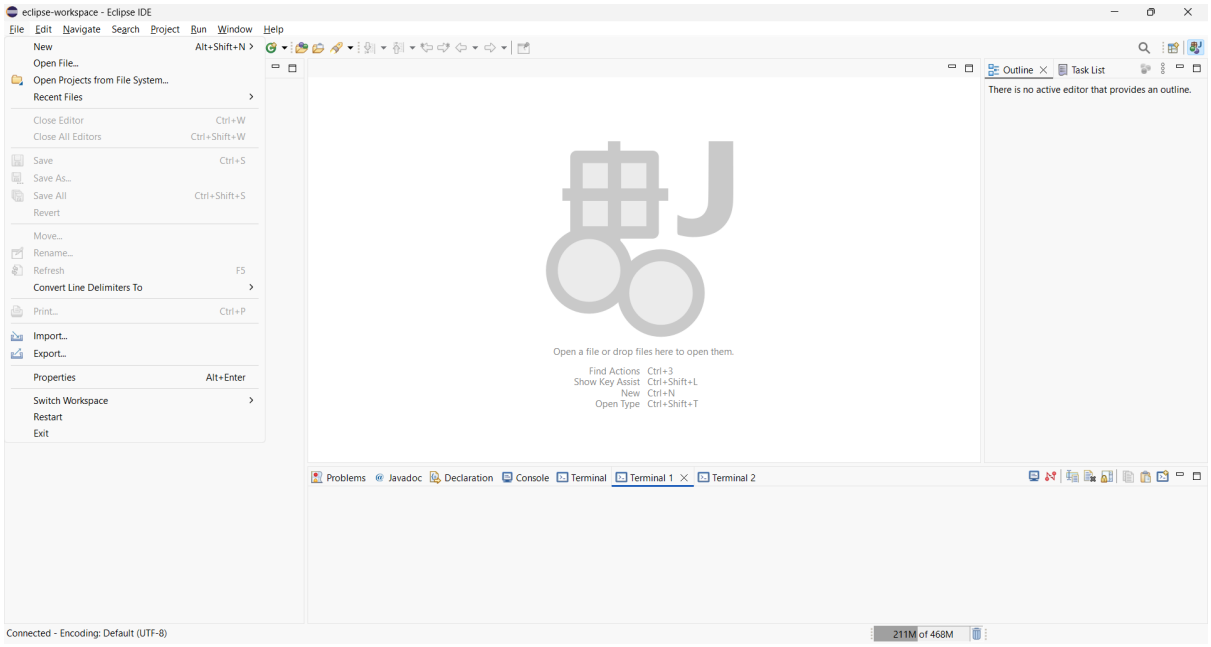
- **Network Delay** – Transmission delay across WLAN/WAN.
- **Server Utilization** – Edge and cloud resource usage.

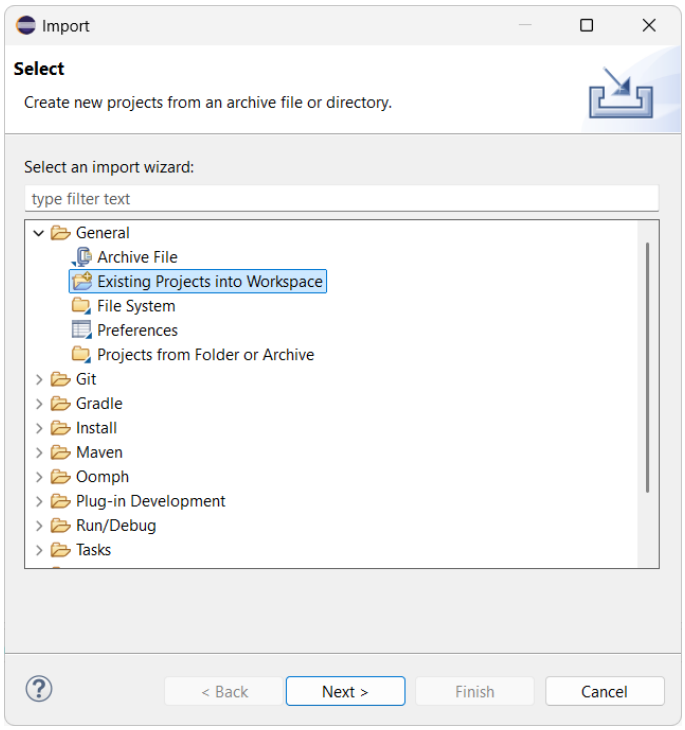
These metrics help compare different **orchestrator policies** and determine which strategy is best for specific scenarios.

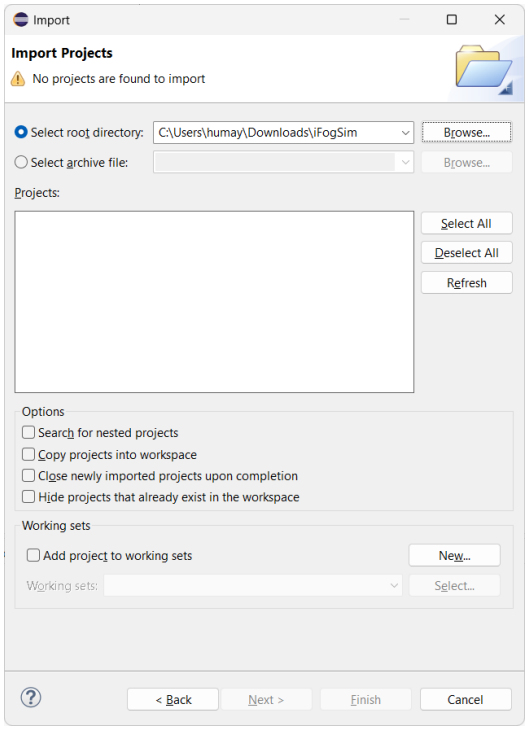
PROCEDURE

Step 1: Import the EdgeCloudSim Project

- Open **Eclipse IDE** → File → Import.



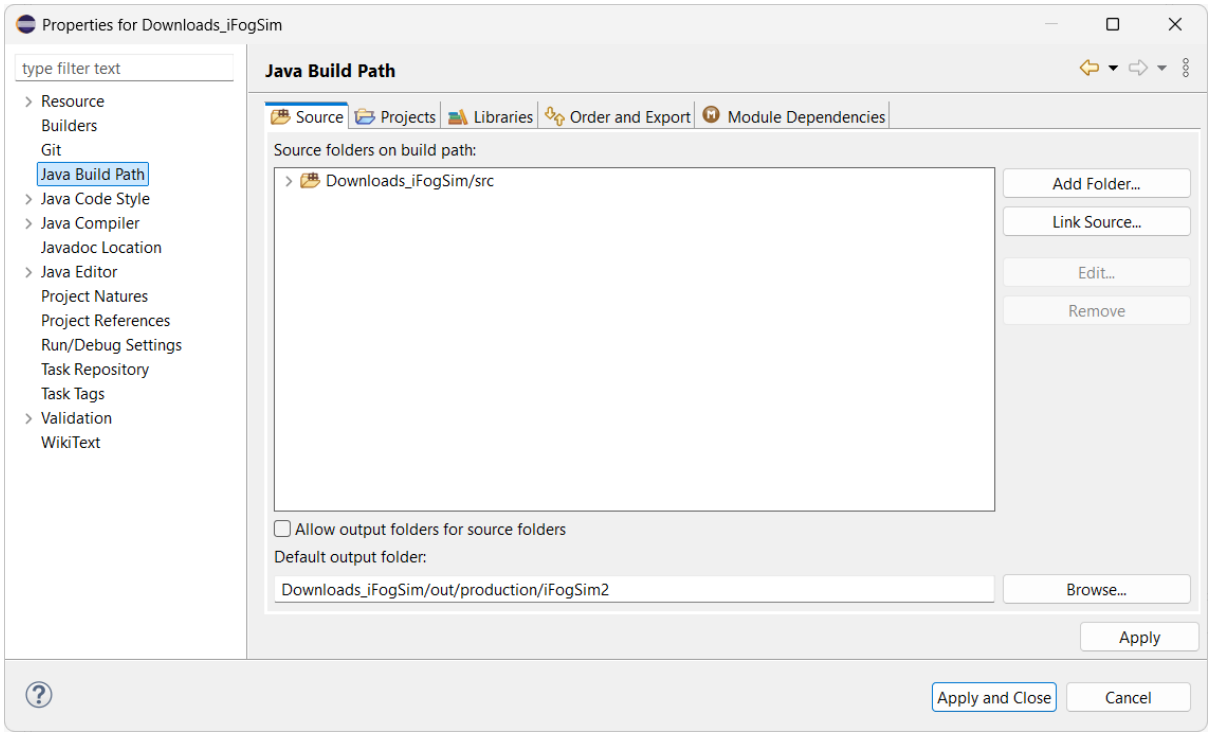




- Choose **Existing Projects into Workspace** and browse to the cloned EdgeCloudSim directory.
- Select the project and click **Finish**.

Step 2: Configure the Build Path

- Right-click on the imported project → Properties.



- Under **Java Build Path**, check if the src folder is listed under **Source** and required JARs under **Libraries**.
- Add missing libraries if necessary.

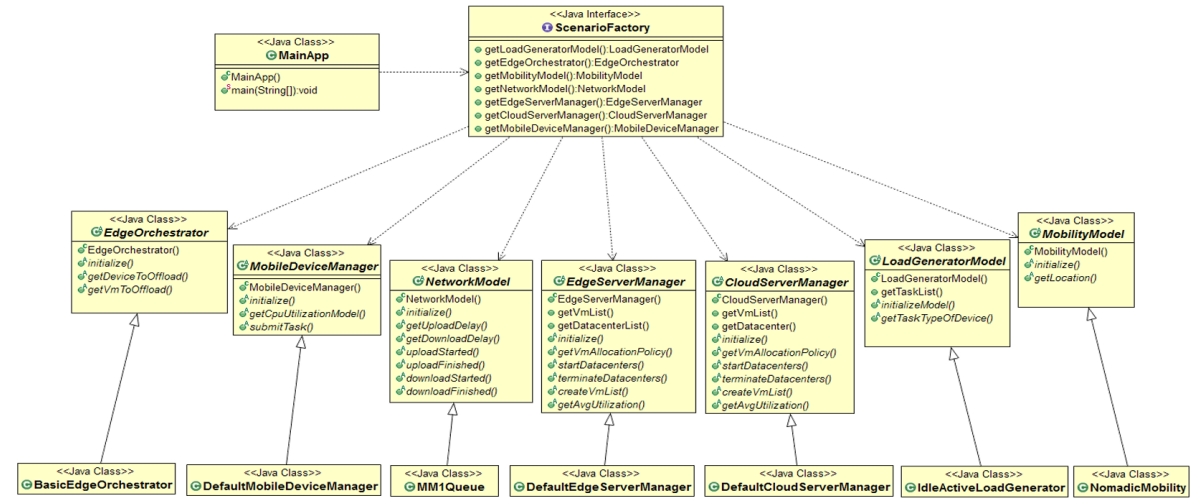
Step 3: Modify Simulation Configuration

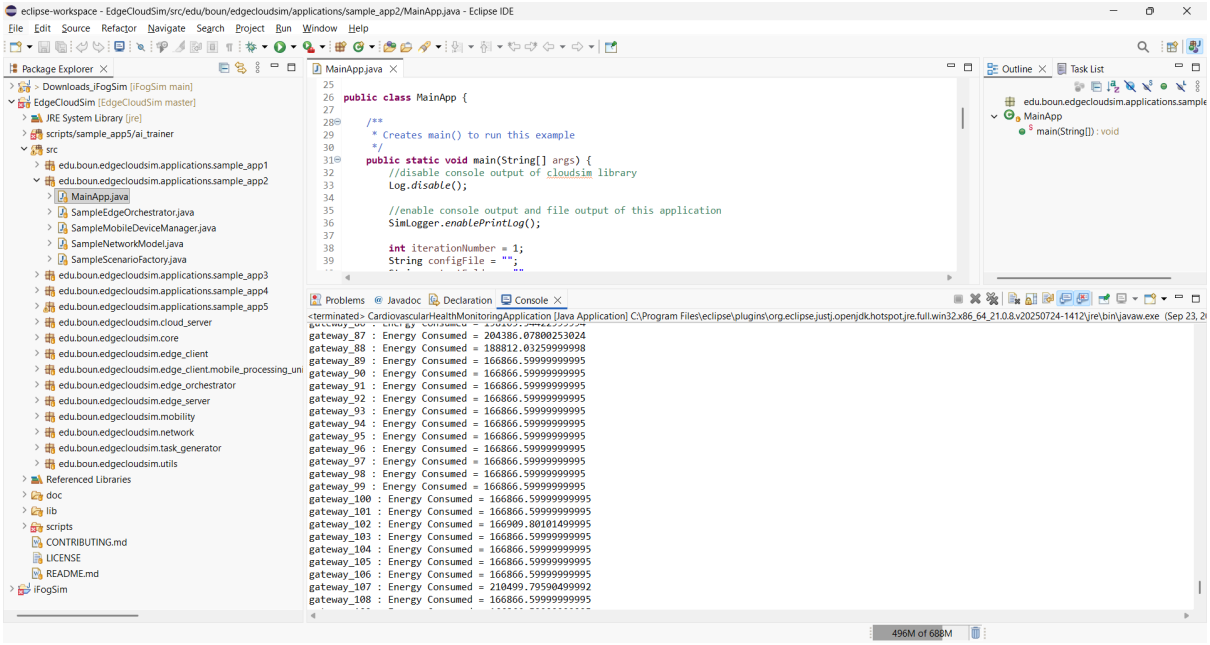
- Locate the **configuration files** inside EdgeCloudSim (e.g., scripts or config).
- Define parameters like:
    - Number of devices
    - Network type (WLAN/WAN)
    - Mobility model
    - Orchestration policy (e.g., latency-based, random, load-aware)

Step 4: Run the Simulation

- Navigate to the **main class** of the experiment (e.g., EdgeCloudSim.java).

- Right-click → Run As → **Java Application**.
- The simulator runs and outputs logs in **CSV format** for analysis.





CONCLUSION

The experiment successfully simulated different **orchestrator policies** in a **two-tier edge-cloud environment** using EdgeCloudSim.

Findings included:

- **Latency-aware orchestration** reduces response times significantly.
- **Resource-aware policies** balance workloads better and minimize task failures.
- **Random scheduling** is simple but leads to inefficiency in heavy workloads.

Overall, EdgeCloudSim proves to be a **powerful research tool**