Lab 1: iFog Simulator Installation and Configuration Setup
Aim:
To install and configure the iFog Simulator for fog computing experimentation.
Algorithm:

1. Initialize the Java environment and install JDK.
2. Download the iFogSim framework from the official repository.
3. Open the project in an IDE (Eclipse or IntelliJ).
4. Configure required libraries and dependencies using Maven or manually.
5. Run the provided sample applications to confirm setup.

Program:
java
```
public class iFogSetup {
    public static void main(String[] args) {
        System.out.println("Starting iFog Simulator setup...");
        // Example: Configuring CloudSim
        CloudSim.init();
        System.out.println("iFog Simulator setup completed successfully!");
    }
}
```
Result:
Thus the installation and configuration of the iFog Simulator for fog computing experimentation has been done successfully

Lab 2: Implementation of Fog Nodes with Different Configuration Setup
Aim:
To Implement the Fog Nodes with Different Configuration Setup
Algorithm:

1. Define the fog node parameters, including CPU, RAM, bandwidth, and storage.
2. Create multiple fog nodes with varying configurations.
3. Connect the fog nodes to the cloud and edge devices.
4. Simulate the performance of fog nodes with different workloads.

Program:
java
```
public class FogNodeSetup {
    public static void main(String[] args) {
        // Example of creating a FogNode
        FogNode fogNode1 = new FogNode("Node1", 1000, 2048, 10000, 100);
        FogNode fogNode2 = new FogNode("Node2", 2000, 4096, 20000, 200);
            System.out.println("Fog Node 1 Configuration: " + fogNode1);
        System.out.println("Fog Node 2 Configuration: " + fogNode2);
    }
}

// Hypothetical FogNode class for demonstration
class FogNode {
    String name;
    int mips, ram, storage, bw;
    public FogNode(String name, int mips, int ram, int storage, int bw) {
        this.name = name;
        this.mips = mips;
```

```
        this.ram = ram;
        this.storage = storage;
        this.bw = bw;
    }
    @Override
    public String toString() {
        return "Name: " + name + ", MIPS: " + mips + ", RAM: " + ram + "MB, Storage: " + storage + "GB,
Bandwidth: " + bw + "Mbps";
    }
}
```

Result:

Thus the above program has been executed successfully

Lab 3: Demonstration of Various Fog Simulators

Aim:

To demonstrate the various Fog Simulators

Algorithm:

1. Research and identify key fog simulators like iFogSim, FogTorch, and EdgeCloudSim.
2. Set up each simulator and note their unique features.
3. Compare simulators based on ease of use, performance metrics, and features.
4. Record findings in a tabular format.

Comparison of iFogSim, FogTorch, and EdgeCloudSim

Feature	**iFogSim**	**FogTorch**	**EdgeCloudSim**
Purpose	Fog computing simulation	Fog and edge computing with AI/ML focus	Edge and cloud computing simulation
Main Focus	IoT devices, fog nodes, cloud integration	AI/ML workloads in fog environments	Edge-cloud interaction, task offloading
Supported Technologies	Java, SimJava	Torch deep learning framework	Java, SimJava
Use Cases	Smart cities, healthcare, IoT applications	AI-based applications (e.g., autonomous vehicles, smart sensors)	Real-time applications, multimedia streaming, smart cities
Task Offloading	Yes (Fog nodes to cloud)	Yes (fog to cloud or edge devices)	Yes (Edge to cloud)
AI/ML Support	Limited	Yes (AI/ML applications at the edge)	Limited
Simulation Focus	Resource management, task scheduling, latency	AI/ML task offloading and performance	Task offloading, resource management, hybrid edge-cloud systems
Energy Consumption	Yes	Yes	Yes
Latency Simulation	Yes	Yes	Yes
Scalability	High (large-scale fog environments)	Moderate (focused on AI tasks)	High (edge-cloud hybrid networks)
Customizability	High (various fog architectures)	Moderate (specialized for AI/ML)	High (supports large-scale edge-cloud simulations)

Feature	iFogSim	FogTorch	EdgeCloudSim
Performance Metrics	Latency, energy consumption, resource utilization	Energy consumption, task offloading time, AI performance	Task execution time, energy consumption, latency, network communication
Community & Documentation	Active (wide use in academic research)	Niche (AI-specific focus, smaller community)	Solid (used for edge-cloud research)
Best For	General fog computing and IoT applications	AI/ML workloads in fog and edge computing	Hybrid edge-cloud systems and real-time applications

This table summarizes the key differences and features of the three simulation frameworks to help you choose the best fit for your specific use case.

Result:
Thus the comparison different fog simulators were done successfully

Lab 4: Implementation of Application Models using iFog
Aim:
To implement the Application Models using iFogSim
Algorithm:
1. Define the application model components: sensors, actuators, and processing modules.
2. Create the application using AppModule classes in iFogSim.
3. Configure the application module's interaction with fog nodes and devices.
4. Simulate the deployment of the application on fog infrastructure.

Program: (JAVA Code)

ApplicationModel.java > Application

```java
import java.util.ArrayList;
import java.util.List;

class Application {
    private String name;
    private List<AppModule> modules;
    private List<Edge> edges;

    public Application(String name, List<AppModule> modules) {
        this.name = name;
        this.modules = modules;
        this.edges = new ArrayList<>();
    }

    Tabnine | Edit | Test | Explain | Document
    public void addModule(AppModule module) {
        modules.add(module);
    }

    Tabnine | Edit | Test | Explain | Document
    public void addEdge(String fromModule, String toModule, int bandwidth, int latency) {
        AppModule from = null, to = null;
        for (AppModule module : modules) {
            if (module.getName().equals(fromModule)) {
                from = module;
            }
            if (module.getName().equals(toModule)) {
                to = module;
            }
        }
        if (from != null && to != null) {
            edges.add(new Edge(from, to, bandwidth, latency));
        }
    }
```

```java
    @Override
    public String toString() {
        return "Application{name='" + name + "', modules=" +
         modules + ", edges=" + edges + "}";
    }
}

class AppModule {
    private String name;
    private int cpuRequirement;
    private int memoryRequirement;

    public AppModule(String name, int cpuRequirement, int memoryRequirement) {
        this.name = name;
        this.cpuRequirement = cpuRequirement;
        this.memoryRequirement = memoryRequirement;
    }

    Tabnine | Edit | Test | Explain | Document
    public String getName() {
        return name;
    }

    Tabnine | Edit | Test | Explain | Document
    @Override
    public String toString() {
        return "AppModule{name='" + name +
        "', cpuRequirement=" +
         cpuRequirement + ", memoryRequirement=" +
          memoryRequirement + "}";
    }
}
```

```java
class Edge {
    private AppModule fromModule;
    private AppModule toModule;
    private int bandwidth;
    private int latency;

    public Edge(AppModule fromModule,
    AppModule toModule, int bandwidth, int latency) {
        this.fromModule = fromModule;
        this.toModule = toModule;
        this.bandwidth = bandwidth;
        this.latency = latency;
    }

    Tabnine | Edit | Test | Explain | Document
    @Override
    public String toString() {
        return "Edge{from=" + fromModule.getName() +
        ", to=" + toModule.getName() + ", bandwidth=" +
        bandwidth + ", latency=" + latency + "}";
    }
}

public class ApplicationModel {
    Run | Debug | Tabnine | Edit | Test | Explain | Document
    public static void main(String[] args) {
        Application app = new Application(name:"MyApp", new ArrayList<>());
        AppModule sensorModule = new AppModule(name:"Sensor", cpuRequirement:100, memoryRequirement:500);
        AppModule processingModule = new AppModule(name:"Processor", cpuRequirement:2000, memoryRequirement:1024);
        app.addModule(sensorModule);
        app.addModule(processingModule);

        app.addEdge(fromModule:"Sensor", toModule:"Processor", bandwidth:100, latency:10);
        System.out.println("Application Model Created: " + app);
    }
}
```

Output:

```
Application Model Created: Application{name='MyApp', modules=[AppModule{name='Sensor', cpuRequirement=100, memoryRequirement=500}, AppModule{name='Processor', cpuRequirement=2000, memoryRequirement=1024}], edges=[Edge{from=Sensor, to=Processor, bandwidth=100, latency=10}]}
```

Reason:
The string output starts with **Application {name='MyApp',** which confirms that the Application object named "MyApp" was successfully instantiated. The output confirms that both "Sensor" and "Processor" modules were successfully added to the application with the correct CPU and memory requirements and it also proves that an edge (data link) was successfully created between the "Sensor" and "Processor" modules with a bandwidth of 100 and a latency of 10.
Result: Thus we have successfully implemented Application Models using iFogSim

Lab 5: Simulation of Application Models using iFog Master-Worker Application Models
Aim: To simulate application Models using iFog Master-Worker Application Models
Algorithm:

1. Define master and worker nodes in the application model.
2. Configure task allocation for the master to distribute workloads to workers.

3. Simulate the processing of tasks in a distributed environment.
4. Analyze performance metrics such as latency and energy consumption.

Program:

```java
class FogNode {
    private String name;
    @SuppressWarnings("unused")
    private int cpuPower;
    @SuppressWarnings("unused")
    private int memory;
    @SuppressWarnings("unused")
    private int storage;
    @SuppressWarnings("unused")
    private int bandwidth;

    public FogNode(String name, int cpuPower, int memory, int storage, int bandwidth) {
        this.name = name;
        this.cpuPower = cpuPower;
        this.memory = memory;
        this.storage = storage;
        this.bandwidth = bandwidth;
    }

    Tabnine | Edit | Test | Explain | Document
    public void assignTask(FogNode worker, Task task) {
        System.out.println(name + " is assigning task: " + task.getTaskName() + " to " + worker.getName());
    }

    Tabnine | Edit | Test | Explain | Document
    public String getName() {
        return name;
    }
}
```

```java
class Task {
    private String taskName;
    private int taskSize;

    public Task(String taskName, int taskSize) {
        this.taskName = taskName;
        this.taskSize = taskSize;
    }

    Tabnine | Edit | Test | Explain | Document
    public String getTaskName() {
        return taskName;
    }

    Tabnine | Edit | Test | Explain | Document
    public int getTaskSize() {
        return taskSize;
    }
}

public class MasterWorkerSimulation {
    Run | Debug | Tabnine | Edit | Test | Explain | Document
    public static void main(String[] args) {
        // Node setup
        FogNode master = new FogNode(name:"Master", cpuPower:4000, memory:8192, storage:50000, bandwidth:1000);
        FogNode worker1 = new FogNode(name:"Worker1", cpuPower:2000, memory:4096, storage:20000, bandwidth:500);
        FogNode worker2 = new FogNode(name:"Worker2", cpuPower:2000, memory:4096, storage:20000, bandwidth:500);

        // Task distribution
        Task task1 = new Task(taskName:"Task1", taskSize:500);
        Task task2 = new Task(taskName:"Task2", taskSize:700);

        master.assignTask(worker1, task1);
        master.assignTask(worker2, task2);
        System.out.println(x:"Simulation Completed");
    }
}
```

Output:

```
Master is assigning task: Task1 to Worker1
Master is assigning task: Task2 to Worker2
Simulation Completed
```

Reason:

"**Master**" with CPU: 4000, Memory: 8192, Storage: 50000, Bandwidth: 1000.
"**Worker1**" with CPU: 2000, Memory: 4096, Storage: 20000, Bandwidth: 500.
"**Worker2**" with CPU: 2000, Memory: 4096, Storage: 20000, Bandwidth: 500.

Result: We have successfully executed and the simulation has been completed

Lab 6: Simulation of Application Models using iFog Master Sequential Unidirectional Application Models

Aim: To simulate Application Models using iFog Master Sequential Unidirectional Application Models

Algorithm:

1. Configure master and worker nodes in a unidirectional task flow.
2. Define task priorities and sequential execution order.

3. Deploy the application model in the fog environment.
4. Measure throughput and task completion time.

Program:

```java
public class SequentialAppModel {
    Run | Debug | Tabnine | Edit | Test | Explain | Document
    public static void main(String[] args) {
        FogNode master = new FogNode(name:"Master", cpuSpeed:4000, ram:8192, storage:50000, networkBandwidth:1000);

        // Create two tasks
        Task task1 = new Task(name:"Task1", executionTime:300);  // Task1 takes 300ms
        Task task2 = new Task(name:"Task2", executionTime:400);  // Task2 takes 400ms

        master.executeTask(task1);
        master.executeTask(task2);
        System.out.println(x:"Unidirectional Sequential Execution Completed");
    }
}

class FogNode {
    String name;
    int cpuSpeed;
    int ram;
    int storage;
    int networkBandwidth;

    public FogNode(String name, int cpuSpeed, int ram, int storage, int networkBandwidth) {
        this.name = name;
        this.cpuSpeed = cpuSpeed;
        this.ram = ram;
        this.storage = storage;
        this.networkBandwidth = networkBandwidth;
    }

```

```java
    public void executeTask(Task task) {
        System.out.println(name + " is executing " +
        task.getName() + " which takes " + task.getExecutionTime() + " ms.");
        try {
            Thread.sleep(task.getExecutionTime());
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(task.getName() + " execution completed.");
    }
}

class Task {
    String name;
    int executionTime;

    public Task(String name, int executionTime) {
        this.name = name;
        this.executionTime = executionTime;
    }

    public String getName() {
        return name;
    }

    // Getter for task execution time
    public int getExecutionTime() {
        return executionTime;
    }
}
```

Output:

```
Master is executing Task1 which takes 300 ms.
Task1 execution completed.
Master is executing Task2 which takes 400 ms.
Task2 execution completed.
Unidirectional Sequential Execution Completed
```

Reason:
Tasks are e[…]ram does not move to Task2 before Task1 completes, confirming blocking execution due to Thread.sleep(). **300ms delay** confirms Task1 ran for the expected duration. **400ms delay** confirms Task2 ran as expected.
Result: Simulation of Application Models using iFog Master Sequential Unidirectional Application Models has been successfully executed

Lab 7: Design of Sensor Nodes and Simulate with Different Tuple Emission Rates
Aim: To Design Sensor Nodes and Simulate them with different tuple emission rates
Algorithm:

1. Define sensor nodes and their properties such as tuple emission rate.
2. Create fog nodes to process sensor data.

3. Simulate different tuple emission rates for varying workloads.
4. Observe the effect on system performance metrics.

Program:

```
public class SensorNodeSimulation {
    Run | Debug | Tabnine | Edit | Test | Explain | Document
    public static void main(String[] args) {
        SensorNode sensor = new SensorNode(name:"TempSensor", emissionRate:10);
        FogNode processor = new FogNode(name:"Processor",
        cpuSpeed:2000, ram:4096, storage:20000, networkBandwidth:500);
        sensor.connectTo(processor);
        sensor.emitTuples(numberOfTuples:10);
        System.out.println(x:"Sensor Node Simulation Completed");
    }
}

class SensorNode {
    String name;
    int emissionRate; // Tuples per second
    FogNode connectedNode;

    public SensorNode(String name, int emissionRate) {
        this.name = name;
        this.emissionRate = emissionRate;
    }

    Tabnine | Edit | Test | Explain | Document
    public void connectTo(FogNode fogNode) {
        this.connectedNode = fogNode;
        System.out.println(name + " connected to " + fogNode.getName());
    }

    public void emitTuples(int numberOfTuples) {
        System.out.println(name + " emitting " + numberOfTuples +
        " tuples/sec to " + connectedNode.getName());
        try {
            Thread.sleep(millis:1000); // 1 second per emission cycle
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Emitted " + numberOfTuples +
        " tuples to " + connectedNode.getName());
    }
}

class FogNode {
    String name;
    int cpuSpeed;
    int ram;
    int storage;
    int networkBandwidth;

    public FogNode(String name, int cpuSpeed, int ram, int storage, int networkBandwidth) {
        this.name = name;
        this.cpuSpeed = cpuSpeed;
```

Output:

```
TempSensor connected to Processor
TempSensor emitting 10 tuples/sec to Processor
Emitted 10 tuples to Processor
Sensor Node Simulation Completed
```

Reason:
"**TempSensor connected to Processor**" verifies that the connection between sensor and processor was established.
"**TempSensor emitting 10 tuples/sec to Processor**" confirms the sensor correctly initiates tuple transmission.
"**Emitted 10 tuples to Processor**" confirms successful data transmission.
The presence of **Thread.sleep(1000);** ensures real-time simulation of tuple emission, proving sequential execution.

Result: We have successfully designed sensor nodes and simulated them using different tuple emission rates

Lab 8: Design of Mobile Edge Node using iFog

Aim: To design Mobile Edge Node using iFog

Algorithm:

1. Define the mobile edge node parameters such as mobility, computational capacity, and bandwidth.
2. Implement the mobility model for the edge node.
3. Simulate task offloading from the mobile edge node to fog or cloud nodes.
4. Measure latency, energy consumption, and task completion time.

Program:

```
public class MobileEdgeNode {
    @SuppressWarnings("unused")
    private String nodeName;
    private int cpuSpeed; // in MHz
    @SuppressWarnings("unused")
    private int memorySize; // in MB
    @SuppressWarnings("unused")
    private int bandwidth; // in Kbps
    private int battery; // in mAh
    @SuppressWarnings("unused")
    private Object mobilityModel;

    public MobileEdgeNode(String nodeName, int cpuSpeed,
    int memorySize, int bandwidth, int battery) {
        this.nodeName = nodeName;
        this.cpuSpeed = cpuSpeed;
        this.memorySize = memorySize;
        this.bandwidth = bandwidth;
        this.battery = battery;
        this.mobilityModel = null;
    }

    Tabnine | Edit | Test | Explain | Document
    public void setMobilityModel(Object mobilityModel) {
        this.mobilityModel = mobilityModel;
        System.out.println("Mobility Model Set: " + mobilityModel);
```

```java
    public static void main(String[] args) {
        MobileEdgeNode edgeNode = new MobileEdgeNode(nodeName:"MobileEdge1",
        cpuSpeed:3000, memorySize:2048, bandwidth:15000, battery:500);
        edgeNode.setMobilityModel(new RandomWaypointModel());
        Task mobileTask = new Task(taskName:"MobileTask1", taskSize:400);
        edgeNode.executeTask(mobileTask);

        System.out.println(x:"Mobile Edge Node Simulation Completed");
    }
}
class RandomWaypointModel {
    Tabnine | Edit | Test | Explain | Document
    @Override
    public String toString() {
        return "Random Waypoint Mobility Model";
    }
}
class Task {
    private String taskName;
    private int taskSize; // Task size in KB (simplified)

    public Task(String taskName, int taskSize) {
        this.taskName = taskName;
        this.taskSize = taskSize;
    }
    Tabnine | Edit | Test | Explain | Document
    public String getTaskName() {
        return taskName;
    }
    Tabnine | Edit | Test | Explain | Document
    public int getTaskSize() {
        return taskSize;
    }
}
```

Output:

```
Mobility Model Set: Random Waypoint Mobility Model
Executing task: MobileTask1 with size: 400
Task processed in: 133 seconds
Remaining battery: 490 mAh
Mobile Edge Node Simulation Completed
```

Reason:
"**Mobility Model Set: Random Waypoint Mobility Model**" confirms that the mobility model was assigned properly. "**Executing task: MobileTask1 with size: 400**" verifies that the task was assigned and started execution. "**Task processed in: 133 seconds**" confirms the correct processing time calculation using CPU speed. "**Remaining battery: 490 mAh**" confirms that battery consumption logic is implemented correctly. "**Mobile Edge Node Simulation Completed**" proves that all operations completed successfully.

Result: We have successfully designed Mobile Edge Node using iFog