import java.util.ArrayList;

import java.util.List;

// Federated computing task scheduler

public class FederationComputeScheduler {

private List<EdgeComputeNode> edgeNodes;

public FederationComputeScheduler(List<EdgeComputeNode> edgeNodes) {

this.edgeNodes = edgeNodes;

}

// Schedule tasks to available edge devices

public void scheduleTask(Task task) {

// Get the available edge devices

List<EdgeComputeNode> availableNodes = getAvailableNodes();

if (availableNodes.isEmpty()) {

System.out.println("No available nodes for task scheduling.");

return;

}

// Simple polling scheduling strategy to distribute tasks to available edge devices

int nodeIndex = 0;

for (int i = 0; i < task.getSubtasks().size(); i++) {

EdgeComputeNode node = availableNodes.get(nodeIndex);

node.executeTaskLocally(task.getSubtasks().get(i));

System.out.println("Task subtask " + (i + 1) + " scheduled on node: " + node.getName());

// Loop to select the next available node

nodeIndex = (nodeIndex + 1) % availableNodes.size();

}

System.out.println("Task scheduled successfully.");

}

// Get the available edge devices

private List<EdgeComputeNode> getAvailableNodes() {

List<EdgeComputeNode> availableNodes = new ArrayList<>();

for (EdgeComputeNode node : edgeNodes) {

if (node.isAvailable()) {

availableNodes.add(node);

}

}

return availableNodes;

}

import java.util.ArrayList;

import java.util.List;

import java.util.PriorityQueue;

// Federated computing task scheduler

public class FederationComputeScheduler {

private List<EdgeComputeNode> edgeNodes;

private PriorityQueue<Task> taskQueue;

public FederationComputeScheduler(List<EdgeComputeNode> edgeNodes) {

this.edgeNodes = edgeNodes;

this.taskQueue = new PriorityQueue<>((t1, t2) -> t2.getPriority() - t1.getPriority());

}

// Schedule tasks to available edge devices

public void scheduleTasks(List<Task> tasks) {

for (Task task : tasks) {

taskQueue.add(task);

}

while (!taskQueue.isEmpty()) {

Task currentTask = taskQueue.poll();

List<EdgeComputeNode> availableNodes = getAvailableNodes();

if (availableNodes.isEmpty()) {

System.out.println("No available nodes for task scheduling: " + currentTask.getName());

// Put the task back to the queue and wait for the next scheduling

taskQueue.add(currentTask);

} else {

// Select the task with the highest priority

EdgeComputeNode selectedNode = selectNode(availableNodes);

assignTaskToNode(currentTask, selectedNode);

}

}

System.out.println("All tasks scheduled successfully.");

}

// Get the available edge devices

private List<EdgeComputeNode> getAvailableNodes() {

List<EdgeComputeNode> availableNodes = new ArrayList<>();

for (EdgeComputeNode node : edgeNodes) {

if (node.isAvailable()) {

availableNodes.add(node);

}

}

return availableNodes;

}

// Select edge devices, considering resources and priorities

private EdgeComputeNode selectNode(List<EdgeComputeNode> availableNodes) {

// In practical applications, the optimal node can be selected based on resource utilization and network latency

return availableNodes.get(0);

}

// Assign tasks to edge devices

private void assignTaskToNode(Task task, EdgeComputeNode node) {

node.executeTaskLocally(task);

System.out.println("Task '" + task.getName() + "' scheduled on node: " + node.getName());

}

}

}