**Develop Economic based scheduling algorithm for cloud computing**

The **Shortest Job First Scheduling (SFS)** algorithm, often called **SJF**, selects the task with the shortest processing time to execute next. In a **cloud computing environment**, SFS aims to improve **response time** and **throughput** by prioritizing shorter tasks first.

**Steps to Implement SFS in CloudSim**

1. **Initialize CloudSim** with data centers, VMs, and tasks.
2. **Sort tasks (cloudlets)** by their length (shortest to longest).
3. **Allocate tasks to VMs** in the sorted order.
4. **Execute the simulation** and analyze performance.

**Java Implementation of SFS Algorithm in CloudSim**

**1. Required Dependencies**

* Download and configure **CloudSim** in your Java environment.
* Use an IDE like Eclipse or IntelliJ and import the CloudSim JARs.

**SFS Scheduling Code in Java**

import org.cloudbus.cloudsim.\*;

import org.cloudbus.cloudsim.core.CloudSim;

import java.util.\*;

public class SFSSchedulingAlgorithm {

public static void main(String[] args) {

try {

// Step 1: Initialize CloudSim

int numUsers = 1;

Calendar calendar = Calendar.getInstance();

boolean traceFlag = false;

CloudSim.init(numUsers, calendar, traceFlag);

// Step 2: Create Datacenter and DatacenterBroker

Datacenter datacenter = createDatacenter("Datacenter\_1");

DatacenterBroker broker = new DatacenterBroker("Broker");

// Step 3: Create VMs and Cloudlets (Tasks)

List<Vm> vmList = createVMs(5, broker.getId()); // Create 5 VMs

List<Cloudlet> cloudletList = createCloudlets(10, broker.getId()); // Create 10 Cloudlets

// Step 4: Sort Cloudlets by length using SFS (Shortest Job First)

cloudletList.sort(Comparator.comparingLong(Cloudlet::getCloudletLength));

// Step 5: Submit VMs and Cloudlets to the broker

broker.submitVmList(vmList);

broker.submitCloudletList(cloudletList);

// Step 6: Bind Cloudlets to VMs (Simple Round-Robin Binding)

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

Cloudlet cloudlet = cloudletList.get(i);

Vm vm = vmList.get(i % vmList.size()); // Assign in round-robin fashion

broker.bindCloudletToVm(cloudlet.getCloudletId(), vm.getId());

}

// Step 7: Start Simulation

CloudSim.startSimulation();

// Retrieve results after simulation ends

List<Cloudlet> newList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

// Print Cloudlet Execution Results

printCloudletResults(newList);

} catch (Exception e) {

e.printStackTrace();

}

}

private static Datacenter createDatacenter(String name) {

List<Host> hostList = new ArrayList<>();

List<Pe> peList = new ArrayList<>();

int mips = 1000;

peList.add(new Pe(0, new PeProvisionerSimple(mips))); // Create 1 Processing Element (PE)

int hostId = 0;

int ram = 2048; // Host memory (RAM)

long storage = 1000000; // Storage

int bw = 10000; // Bandwidth

Host host = new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList,

new VmSchedulerTimeShared(peList)

);

hostList.add(host);

String arch = "x86"; // System architecture

String os = "Linux"; // Operating System

String vmm = "Xen"; // Virtual Machine Monitor

double time\_zone = 10.0; // Time zone

double costPerSec = 3.0; // Cost per second

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(

arch, os, vmm, hostList, time\_zone, costPerSec, 0.05, 0.1, 0.1

);

Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), new LinkedList<>(), 0);

} catch (Exception e) {

e.printStackTrace();

}

return datacenter;

}

private static List<Vm> createVMs(int numVMs, int brokerId) {

List<Vm> vmList = new ArrayList<>();

for (int i = 0; i < numVMs; i++) {

Vm vm = new Vm(i, brokerId, 1000, 1, 2048, 10000, 10000, "Xen", new CloudletSchedulerTimeShared());

vmList.add(vm);

}

return vmList;

}

private static List<Cloudlet> createCloudlets(int numCloudlets, int brokerId) {

List<Cloudlet> cloudletList = new ArrayList<>();

for (int i = 0; i < numCloudlets; i++) {

long length = (long) (1000 + Math.random() \* 9000); // Random length between 1000 and 10000

Cloudlet cloudlet = new Cloudlet(i, length, 1, 300, 300, new UtilizationModelFull(), new UtilizationModelFull(), new UtilizationModelFull());

cloudlet.setUserId(brokerId);

cloudletList.add(cloudlet);

}

return cloudletList;

}

private static void printCloudletResults(List<Cloudlet> list) {

String indent = " ";

System.out.println("========== OUTPUT ==========");

System.out.println("Cloudlet ID" + indent + "STATUS" + indent +

"Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent + "Finish Time");

for (Cloudlet cloudlet : list) {

System.out.print(indent + cloudlet.getCloudletId() + indent + indent);

if (cloudlet.getStatus() == Cloudlet.SUCCESS) {

System.out.println("SUCCESS" + indent + indent + cloudlet.getResourceId() +

indent + indent + cloudlet.getVmId() +

indent + indent + cloudlet.getActualCPUTime() +

indent + indent + cloudlet.getExecStartTime() +

indent + indent + cloudlet.getFinishTime());

}

}

}

}

**Explanation of the Code**

1. **Initialization**: CloudSim is initialized with a data center and a broker.
2. **VM and Cloudlet Creation**: 5 VMs and 10 cloudlets are created. Cloudlets have random lengths to simulate varied tasks.
3. **SFS Scheduling**: Cloudlets are sorted by length (shortest first) and assigned to VMs.
4. **Execution**: The simulation is executed, and results are printed.

**Output Example**

========== OUTPUT ==========

Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time

0 SUCCESS 1 0 1.5 0.0 1.5

1 SUCCESS 1 1 2.0 0.0 2.0

**FCFS Scheduling Algorithm in CloudSim**

import org.cloudbus.cloudsim.\*;

import org.cloudbus.cloudsim.core.CloudSim;

import java.util.\*;

public class FCFSSchedulingAlgorithm {

public static void main(String[] args) {

try {

// Step 1: Initialize CloudSim

int numUsers = 1; // Number of cloud users

Calendar calendar = Calendar.getInstance();

boolean traceFlag = false; // Disable event tracing

CloudSim.init(numUsers, calendar, traceFlag);

// Step 2: Create Datacenter and DatacenterBroker

Datacenter datacenter = createDatacenter("Datacenter\_1");

DatacenterBroker broker = new DatacenterBroker("Broker");

// Step 3: Create VMs and Cloudlets (Tasks)

List<Vm> vmList = createVMs(3, broker.getId()); // Create 3 VMs

List<Cloudlet> cloudletList = createCloudlets(6, broker.getId()); // Create 6 Cloudlets

// Step 4: Submit VMs and Cloudlets to the broker

broker.submitVmList(vmList);

broker.submitCloudletList(cloudletList);

// Step 5: Bind Cloudlets to VMs (First-Come, First-Serve)

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

Cloudlet cloudlet = cloudletList.get(i);

Vm vm = vmList.get(i % vmList.size()); // Assign cloudlets in a round-robin manner across VMs

broker.bindCloudletToVm(cloudlet.getCloudletId(), vm.getId());

}

// Step 6: Start the simulation

CloudSim.startSimulation();

// Step 7: Retrieve and print results

List<Cloudlet> resultList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

printCloudletResults(resultList);

} catch (Exception e) {

e.printStackTrace();

}

}

private static Datacenter createDatacenter(String name) {

List<Host> hostList = new ArrayList<>();

List<Pe> peList = new ArrayList<>();

int mips = 1000;

peList.add(new Pe(0, new PeProvisionerSimple(mips))); // One processing element (PE)

int hostId = 0;

int ram = 2048; // Host memory (RAM)

long storage = 1000000; // Storage capacity

int bw = 10000; // Bandwidth

Host host = new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList,

new VmSchedulerTimeShared(peList)

);

hostList.add(host);

String arch = "x86"; // System architecture

String os = "Linux"; // Operating system

String vmm = "Xen"; // Virtual Machine Monitor

double time\_zone = 10.0; // Time zone

double costPerSec = 3.0; // Cost per second

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(

arch, os, vmm, hostList, time\_zone, costPerSec, 0.05, 0.1, 0.1

);

Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), new LinkedList<>(), 0);

} catch (Exception e) {

e.printStackTrace();

}

return datacenter;

}

private static List<Vm> createVMs(int numVMs, int brokerId) {

List<Vm> vmList = new ArrayList<>();

for (int i = 0; i < numVMs; i++) {

Vm vm = new Vm(i, brokerId, 1000, 1, 2048, 10000, 10000, "Xen", new CloudletSchedulerTimeShared());

vmList.add(vm);

}

return vmList;

}

private static List<Cloudlet> createCloudlets(int numCloudlets, int brokerId) {

List<Cloudlet> cloudletList = new ArrayList<>();

for (int i = 0; i < numCloudlets; i++) {

long length = (long) (1000 + Math.random() \* 4000); // Random length between 1000 and 5000

Cloudlet cloudlet = new Cloudlet(i, length, 1, 300, 300, new UtilizationModelFull(), new UtilizationModelFull(), new UtilizationModelFull());

cloudlet.setUserId(brokerId);

cloudletList.add(cloudlet);

}

return cloudletList;

}

private static void printCloudletResults(List<Cloudlet> list) {

String indent = " ";

System.out.println("========== OUTPUT ==========");

System.out.println("Cloudlet ID" + indent + "STATUS" + indent +

"Datacenter ID" + indent + "VM ID" + indent + "Time" + indent +

"Start Time" + indent + "Finish Time");

for (Cloudlet cloudlet : list) {

System.out.print(indent + cloudlet.getCloudletId() + indent);

if (cloudlet.getStatus() == Cloudlet.SUCCESS) {

System.out.println("SUCCESS" + indent + indent +

cloudlet.getResourceId() + indent + indent +

cloudlet.getVmId() + indent + indent +

cloudlet.getActualCPUTime() + indent + indent +

cloudlet.getExecStartTime() + indent + indent +

cloudlet.getFinishTime());

}

}

}

}

**Explanation of the Code**

1. **Initialization**:
   * CloudSim is initialized with one user and a calendar object.
2. **Datacenter and Broker Creation**:
   * A datacenter and broker are created to manage resources and tasks.
3. **VM and Cloudlet Creation**:
   * 3 VMs and 6 cloudlets (tasks) are created. Cloudlets are assigned random lengths.
4. **FCFS Scheduling Logic**:
   * Cloudlets are assigned to VMs in the order they were created (FCFS logic).
5. **Simulation Execution**:
   * The simulation is started, and results are retrieved and printed.

**Output**

========== OUTPUT ==========

Cloudlet ID STATUS Datacenter ID VM ID Time Start Time Finish Time

0 SUCCESS 1 0 3.5 0.0 3.5

1 SUCCESS 1 1 2.0 0.0 2.0

**Round Robin (RR) Scheduling Algorithm**

import org.cloudbus.cloudsim.\*;

import org.cloudbus.cloudsim.core.CloudSim;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.List;

public class RoundRobinSchedulingExample {

public static void main(String[] args) {

// Initialize CloudSim

int numUsers = 1; // Number of users in the simulation

Calendar calendar = Calendar.getInstance();

boolean traceFlag = false; // Disable event tracing

CloudSim.init(numUsers, calendar, traceFlag);

// Create Datacenter

Datacenter datacenter0 = createDatacenter("Datacenter\_0");

// Create Broker

DatacenterBroker broker = createBroker();

int brokerId = broker.getId();

// Create VMs list

List<Vm> vmList = new ArrayList<>();

int vmCount = 4; // Number of VMs

for (int i = 0; i < vmCount; i++) {

Vm vm = new Vm(i, brokerId, 1000, 1, 1024, 10000, 10000, "Xen", new CloudletSchedulerTimeShared());

vmList.add(vm);

}

// Submit VM list to broker

broker.submitVmList(vmList);

// Create Cloudlets (Tasks)

List<Cloudlet> cloudletList = new ArrayList<>();

int cloudletCount = 10; // Number of Cloudlets

for (int i = 0; i < cloudletCount; i++) {

Cloudlet cloudlet = new Cloudlet(i, 40000, 1, 300, 300,

new UtilizationModelFull(), new UtilizationModelFull(), new UtilizationModelFull());

cloudlet.setUserId(brokerId);

cloudletList.add(cloudlet);

}

// Submit Cloudlet list to broker

broker.submitCloudletList(cloudletList);

// Round Robin Scheduling logic

int vmIndex = 0;

for (Cloudlet cloudlet : cloudletList) {

Vm vm = vmList.get(vmIndex);

broker.bindCloudletToVm(cloudlet.getCloudletId(), vm.getId());

// Move to the next VM in a round-robin fashion

vmIndex = (vmIndex + 1) % vmList.size();

}

// Start Simulation

CloudSim.startSimulation();

// Get simulation results

List<Cloudlet> resultList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

// Print Results

printCloudletResults(resultList);

}

private static Datacenter createDatacenter(String name) {

// Create Hosts list

List<Host> hostList = new ArrayList<>();

int hostId = 0;

int ram = 2048; // Host memory (MB)

long storage = 1000000; // Host storage

int bw = 10000; // Bandwidth

// Create Host with a list of processing elements (PEs)

List<Pe> peList = new ArrayList<>();

peList.add(new Pe(0, new PeProvisionerSimple(1000))); // 1 core, 1000 MIPS

Host host = new Host(

hostId,

new RamProvisionerSimple(ram),

new BwProvisionerSimple(bw),

storage,

peList,

new VmSchedulerTimeShared(peList)

);

hostList.add(host);

// Create Datacenter characteristics

String arch = "x86"; // Architecture

String os = "Linux"; // Operating system

String vmm = "Xen"; // Virtual machine manager

double timeZone = 10.0; // Time zone

double costPerSec = 3.0; // Cost per second

double costPerMem = 0.05; // Cost per MB

double costPerStorage = 0.1; // Cost per storage

double costPerBw = 0.1; // Cost per bandwidth

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(

arch, os, vmm, hostList, timeZone, costPerSec, costPerMem, costPerStorage, costPerBw);

// Create Datacenter

Datacenter datacenter = null;

try {

datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), new ArrayList<>(), 0);

} catch (Exception e) {

e.printStackTrace();

}

return datacenter;

}

private static DatacenterBroker createBroker() {

DatacenterBroker broker = null;

try {

broker = new DatacenterBroker("Broker");

} catch (Exception e) {

e.printStackTrace();

}

return broker;

}

private static void printCloudletResults(List<Cloudlet> list) {

String indent = " ";

System.out.println("========== OUTPUT ==========");

System.out.println("Cloudlet ID" + indent + "STATUS" + indent +

"Data center ID" + indent + "VM ID" + indent + "Time" + indent + "Start Time" + indent + "Finish Time");

for (Cloudlet cloudlet : list) {

System.out.print(indent + cloudlet.getCloudletId() + indent + indent);

if (cloudlet.getStatus() == Cloudlet.SUCCESS) {

System.out.println("SUCCESS" + indent + indent + cloudlet.getResourceId() +

indent + indent + cloudlet.getVmId() + indent + indent +

cloudlet.getActualCPUTime() + indent + indent + cloudlet.getExecStartTime() +

indent + indent + cloudlet.getFinishTime());

}

}

}

}

**Explanation**

1. **Datacenter Setup**:
   * We create a single datacenter with one host and a simple VM scheduler.
   * Each host contains a processing element (PE) with 1000 MIPS.
2. **Broker Creation**:
   * The broker is responsible for submitting VMs and cloudlets to the datacenter and managing resource allocation.
3. **VM and Cloudlet Creation**:
   * We create 4 VMs and 10 cloudlets. Each cloudlet represents a task.
4. **Round Robin Scheduling**:
   * The scheduling logic iterates over the cloudlets and assigns them to VMs in a round-robin manner using modular arithmetic.
5. **Simulation and Results**:
   * After starting the simulation, we print the execution results of each cloudlet.

**Output**

========== OUTPUT ==========

Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time

0 SUCCESS 0 0 40.0 0.1 40.1

1 SUCCESS 0 1 40.0 0.1 40.1

2 SUCCESS 0 2 40.0 0.1 40.1

3 SUCCESS 0 3 40.0 0.1 40.1

4 SUCCESS 0 0 40.0 0.1 40.1

5 SUCCESS 0 1 40.0 0.1 40.1

6 SUCCESS 0 2 40.0 0.1 40.1

7 SUCCESS 0 3 40.0 0.1 40.1

8 SUCCESS 0 0 40.0 0.1 40.1

9 SUCCESS 0 1 40.0 0.1 40.1