CloudSim Plus Datacenter Simulation

This code provides a detailed example of how to use the CloudSim Plus framework to simulate a cloud computing environment with multiple datacenters. The simulation includes VM allocation policies and migration strategies based on CPU utilization thresholds.
Here's a step-by-step breakdown of the main components and logic within the `InterDatacenterMigration1` class:

1. **Constants and Configuration**:
   - Several constants are defined to configure hosts, VMs, and cloudlets.
   - These constants include CPU capacity, memory, storage, bandwidth, utilization thresholds for migration, etc.
2. **Main Method**:
   - The main method initializes and starts the simulation by creating an instance of `InterDatacenterMigration1`.
3. **Constructor**:
   - The constructor sets the log level, validates configuration settings, and initializes the CloudSimPlus simulation.
   - It creates datacenters, brokers, VMs, and cloudlets, and starts the simulation.
4. **Datacenter and VM Allocation Policy Creation**:
   - Datacenters are created with specific characteristics (public or private).
   - A VM allocation policy (`VmAllocationPolicyMigrationFirstFitStaticThreshold`) is used to manage VM migrations based on CPU utilization thresholds.
5. **Brokers and VM/Cloudlet Creation**:
   - Brokers are created for each datacenter to manage VM and cloudlet submissions.
   - VMs and cloudlets are created and submitted to the brokers.
6. **VM and Cloudlet Creation Methods**:
   - `createAndSubmitVms`: Creates VMs with specific configurations and submits them to the broker.
   - `createAndSubmitCloudlets`: Creates cloudlets with specific configurations and submits them to the broker.
7. **Utilization Model**:
   - A dynamic utilization model is used for cloudlets to simulate changing CPU usage over time.
8. **Event Listeners**:
   - Listeners are added to handle events such as VM creation and migration.
   - These listeners help monitor and manage the state of the simulation.
9. **Printing Results**:
   - After the simulation, results are printed, including cloudlet execution details and host state history.

The following example demonstrates how to create a CloudSim Plus simulation for VM migration between datacenters. The setup includes defining hosts, VMs, and cloudlets, and configuring their

resource requirements. The VM allocation policy triggers migrations based on predefined CPU utilization thresholds.
If you have any specific questions or need further clarification on any part of the code, feel free to ask!

The migration policy in the provided CloudSim Plus example is handled by the `VmAllocationPolicyMigrationFirstFitStaticThreshold` class. This policy decides when and where to migrate VMs based on CPU utilization thresholds. Here's a detailed breakdown of how it works:

Key Components

1. **Thresholds for VM Migration:**
   - **Under-utilization Threshold:** If the CPU usage of a host is below this threshold, it will trigger the migration of VMs to balance the load.
   - **Over-utilization Threshold:** If the CPU usage of a host exceeds this threshold, it will also trigger the migration of VMs to avoid overloading the host.
2. **VM Selection Policy:**
   - The `VmSelectionPolicyMinimumUtilization` policy is used to select which VM to migrate. It selects the VM with the minimum CPU utilization, aiming to have the least impact on the system's performance during migration.

Migration Policy in Action

1. **Creation and Setting Up:**
   - A VM allocation policy is created using the `createVmAllocationPolicy` method, which sets up initial high thresholds to place VMs on hosts. After all VMs are created, these thresholds are adjusted to the defined over-utilization threshold to allow migration triggers.

```java
private VmAllocationPolicyMigration createVmAllocationPolicy() {
    final var vmSelection = new VmSelectionPolicyMinimumUtilization();
    final var vmAllocation = new
VmAllocationPolicyMigrationFirstFitStaticThreshold(vmSelection, 0.9);

vmAllocation.setUnderUtilizationThreshold(HOST_UNDER_UTILIZATION_THRESHOLD_FOR_VM_MIGRAT
    return vmAllocation;
}
```

2. **Triggering Migration:**
   - When the simulation starts, VMs are allocated to hosts. If a host's CPU utilization goes above the `HOST_OVER_UTILIZATION_THRESHOLD_FOR_VM_MIGRATION` (0.78 in this case) or below the `HOST_UNDER_UTILIZATION_THRESHOLD_FOR_VM_MIGRATION` (0.1 in this case), the migration policy triggers.
3. **Migration Process:**
   - The migration process involves identifying under-utilized or over-utilized hosts and selecting VMs for migration based on the minimum utilization policy. The selected VM is then migrated to another host that can accommodate it without violating the thresholds.

```java
private void onVmsCreatedListener(final DatacenterBrokerEventInfo info) {
    final var broker = info.getDatacenterBroker();
    final String vmIds = broker.getVmCreatedList()
          .stream()
          .mapToLong(Vm::getId)
          .mapToObj(Long::toString)
          .collect(joining(", "));

    System.out.printf("# %d VMs submitted to %s have been created. VMs: %s.%n",
broker.getVmCreatedList().size(), broker, vmIds);
    datacenterList.stream()
                  .map(dc ->
(VmAllocationPolicyMigrationFirstFitStaticThreshold)dc.getVmAllocationPolicy())
                  .forEach(policy ->
policy.setOverUtilizationThreshold(HOST_OVER_UTILIZATION_THRESHOLD_FOR_VM_MIGRATION));
    broker.removeOnVmsCreatedListener(info.getListener());
}
```

Summary

- **Initialization:** The migration policy is initialized with high thresholds to allow initial VM placement.
- **Adjustment:** After VM creation, the over-utilization threshold is adjusted to the defined value (0.78) to enable migration.
- **Triggering:** When a host is under or over-utilized, the migration policy identifies the VM to migrate using the minimum utilization policy.
- **Execution:** The selected VM is migrated to another host to balance the load.

This policy ensures that VMs are migrated to maintain balanced CPU utilization across hosts, preventing both under-utilization and over-utilization of resources.

Yes, you can use other VM allocation and selection policies in CloudSim Plus besides the `FirstFit` policy. Different policies can be tailored to meet specific requirements and improve the efficiency of resource allocation and migration in a cloud environment. Here are some alternative policies you can consider:

Alternative VM Allocation Policies

1. **VmAllocationPolicySimple**
   - Allocates VMs to the first host that has enough resources.
   - Easy to implement but may not be efficient in resource utilization.

```java
VmAllocationPolicy allocationPolicy = new VmAllocationPolicySimple();
```

2. **VmAllocationPolicyRoundRobin**

- Allocates VMs to hosts in a round-robin manner.
- Ensures a balanced load distribution across hosts.

```java
VmAllocationPolicy allocationPolicy = new VmAllocationPolicyRoundRobin();
```

3. **VmAllocationPolicyBestFit**
   - Allocates VMs to the host where it best fits based on available resources.
   - Tries to minimize resource fragmentation.

```java
VmAllocationPolicy allocationPolicy = new VmAllocationPolicyBestFit();
```

4. **VmAllocationPolicyWorstFit**
   - Allocates VMs to the host where it fits the worst, leaving larger gaps for future VMs.
   - Can be useful to avoid packing VMs too tightly.

```java
VmAllocationPolicy allocationPolicy = new VmAllocationPolicyWorstFit();
```

Alternative VM Selection Policies

1. **VmSelectionPolicyMaximumCpuUtilization**
   - Selects the VM with the maximum CPU utilization for migration.
   - Useful when trying to balance high-utilization VMs across hosts.

```java
VmSelectionPolicy selectionPolicy = new VmSelectionPolicyMaximumCpuUtilization();
```

2. **VmSelectionPolicyMinimumCpuUtilization**
   - Selects the VM with the minimum CPU utilization for migration.
   - Useful when trying to remove less impactful VMs first.

```java
VmSelectionPolicy selectionPolicy = new VmSelectionPolicyMinimumCpuUtilization();
```

3. **VmSelectionPolicyRandomSelection**
   - Selects a VM randomly for migration.
   - Simple and easy to implement, but may not always be efficient.

```java
VmSelectionPolicy selectionPolicy = new VmSelectionPolicyRandomSelection();
```

4. **VmSelectionPolicyMaximumRamUtilization**
   - Selects the VM with the maximum RAM utilization for migration.
   - Useful when balancing memory usage across hosts.

```java
VmSelectionPolicy selectionPolicy = new VmSelectionPolicyMaximumRamUtilization();
```

5. **VmSelectionPolicyMinimumMigrationTime**
   - Selects the VM that can be migrated with the minimum time.
   - Useful for reducing the impact of migration on the system.

```java
VmSelectionPolicy selectionPolicy = new VmSelectionPolicyMinimumMigrationTime();
```

Example of Using Different Policies

Here's how you can use a different VM allocation policy (`BestFit`) and VM selection policy (`MaximumCpuUtilization`) in your CloudSim Plus example:

```java
private VmAllocationPolicyMigration createVmAllocationPolicy() {
    final var vmSelection = new VmSelectionPolicyMaximumCpuUtilization();
    final var vmAllocation = new
VmAllocationPolicyMigrationBestFitStaticThreshold(vmSelection, 0.9);

vmAllocation.setUnderUtilizationThreshold(HOST_UNDER_UTILIZATION_THRESHOLD_FOR_VM_MIGRATION);
    return vmAllocation;
}
```

Summary

You have various options for VM allocation and selection policies in CloudSim Plus. By choosing the appropriate policy based on your specific requirements, you can improve resource utilization, balance load distribution, and minimize migration impact.