面向公平性的云边微服务系统部署方法

Deployment Methods for Cloud - Edge Microservice Systems Oriented towards Fairness

A. 场景描述

在某些**高实时性应用**服务中，用户分布广泛，既包括城市中心的用户，也涵盖偏远地区的用户，他们通过各种终端设备接入系统以请求服务。为了提升服务质量，本研究采用**云-边协同计算架构**，即在多个地理位置部署**边缘服务器**，同时依托**云服务器**提供计算支持，以减少计算任务的远程传输延迟，提高系统的整体响应效率。

在该架构下，如何确保**不同用户在资源受限的环境下获得公平的服务体验**是核心挑战。**用户服务体验的核心衡量指标是响应时间**，即用户从发送请求到接收到服务响应所经历的时间。本研究中的**公平性问题**主要体现在如何在**资源受限的环境**下，确保不同用户在获得计算资源和网络资源时能够享受**相对公平的响应时间**。具体而言，公平性问题可划分为以下两个层面：

1. 同优先级用户间的公平性

在**相同优先级**的用户群体内，地理位置不同导致其**网络条件存在显著差异**。接近边缘服务器的用户通常具有较低的传输延迟，而远离边缘服务器的用户则可能面临更高的传输延迟，从而影响**同优先级用户的响应时间公平性**。

- **公平性目标**：在相同优先级的用户之间，尽可能**均衡响应时间**，避免因地理因素导致服务体验的不均衡，从而实现相对公平的服务分配。

2. 跨优先级用户的公平性

在实际应用中，用户通常按照不同的**服务优先级**进行分类，例如**普通用户**、**VIP 用户**等。**高优先级用户**通常为付费用户，他们期望获得**更短的响应时间**和**更稳定的服务体验**。因此，在资源受限的边缘计算环境中，**系统无法保证所有用户均能获得最优的服务质量**，必须在公平性与服务等级保障之间进行权衡。

- **公平性目标**：确保在资源受限的环境下，**高优先级用户的平均响应时间优于低优先级用户**，即优先满足高优先级用户的计算和网络需求，以保障其**始终享有更优的服务体验**，同时在保证优先级机制有效性的前提下，尽可能避免低优先级用户的极端服务劣化。

总结

本研究旨在设计一种**面向公平性的云-边微服务部署优化策略**，以提高云-边协同计算的公平性和资源利用效率。在**资源受限环境**下，实现以下优化目标：

1. **同一优先级用户的公平性**：确保**相同优先级的用户**在不同地理位置和网络条件下，获得尽可能**均衡**的服务响应时间，避免因网络环境差异导致体验不均衡。

2. **跨优先级用户的服务保障**：在资源受限情况下，优先保障高优先级用户的服务质量，使其平均响应时间始终优于低优先级用户，确保资源分配符合优先级策略。
3. **微服务的动态部署与资源优化**：根据不同服务器的计算能力、网络延迟、带宽等因素，合理部署微服务实例，最大化资源利用率，并降低计算和通信成本。

B. 系统建模

本文研究**云-边协同环境**下的**公平性优化微服务部署问题。**系统由一组**边缘服务器**（记作 S_{edge}）和**云服务器**（记作 S_{cloud}）组成，用户请求的服务将在这些服务器上处理。边缘服务器由系统完全控制，而云服务器按**按需计费**模式提供服务资源。

定义 1（用户）

设 U 为用户集合，每个用户 $u_i \in U$ 具有以下属性：

$$u_i = \langle L_i, W_i, Q_i \rangle$$

其中：

- L_i：用户的**优先级**（取值为 $1, 2, 3, \cdots$），数值越大表示用户的重要程度越高
- W_i：用户的**权重**，与 L_i 成正相关
- $Q_i =< D_i^{in}, D_i^{out}, p_i >$：表示用户的请求集合
  - D_i^{in} 为该请求需要**上传**数据量（MB）；
  - D_i^{out} 为该请求**返回结果**的数据量（MB）；
  - p_i 为处理该请求需要的**计算单位数。**

用户请求的服务需要被分配到某台**边缘服务器**或**云服务器**进行处理。

定义2（服务器）

a. 边缘服务器

每个**边缘服务器** $s_j^{edge} \in S_{edge}$ 具有以下属性：

$$s_j^{edge} =< R_j^e, b_j^e, c_{fixed_j}^e >$$

其中：

- $R_j^e =< r_j^{CPU}, r_j^{RAM}, \cdots >$：表示边缘服务器 s_j 提供的可用计算资源（包括CPU、RAM 等）；
- b_j^e：表示边缘服务器 s_j 的最大可用带宽，本研究中设定服务器的上下行带宽总量相同；
- $c_{fixed_j}^e$：表示租赁服务器 s_j 的固定成本。

系统中**用户终端**与**边缘服务器**之间的**网络带宽**和**通信延迟**分别表示为：

$$B^e = \{b_{ij}^e | \forall u_i \in U, \forall s_j \in S_{edge}\}$$

$$D^e = \{t_{d_{ij}}^e | \forall u_i \in U, \forall s_j \in S_{edge}\}$$

其中：

- 用户 u_i 与服务器 s_j 之间的网络带宽将按照用户数据量与优先级按需分配，计算公式如下：

$$b_{ij}^e = \frac{W_i \cdot (D_i^{in} + D_i^{out})}{\sum_{u_k \in U} x_{ij} \cdot W_i \cdot (D_i^{in} + D_i^{out})} \cdot b_j^e$$

  x_{ij} 为用户与服务器之间的连接关系，其具体定义在见下面**定义 3**。
- 通信延迟取决于实际情况。

b. 云服务器

云服务器 S_{cloud} 资源丰富，在云服务器上处理**每个单位计算请求**的价格为 p_{net}。

系统中**用户终端**与**云服务器**之间的**网络带宽**和**通信延迟**分别表示为：

$$B^c = \{b_i^c | \forall u_i \in U\}$$

$$D^c = \{t_{d_i}^e | \forall u_i \in U\}$$

定义3 （用户与服务器连接）

用户与服务器连接方案 **X** 描述了系统中**用户与服务器的连接关系**：

$$X = \{x_{ij} | \forall u_i \in U, s_j \in S, x_{ij} \in \{0, 1\}\}$$

其中， x_{ij} 表示用户 u_i 是否发送请求到服务器 s_j ：

$$x_{ij} = \begin{cases} 1, & \text{用户} u_i \text{发送请求到服务器} s_j \\ 0, & \text{否则} \end{cases}$$

定义4（微服务）

设 m 为系统提供的微服务（在本研究的场景中，系统只提供一种微服务），

$$m = \langle P^m, r^m \rangle$$

其中：

- P^m：表示 m 的**计算能力**，即**每个服务实例**单位时间可处理的计算单位数。
- r^m：表示运行 m 的服务实例所需的**计算资源**（如 CPU、RAM等）。

微服务部署方案 Y 描述了**微服务实例在服务器上的部署状态**：

$$Y = \{y_j | s_j \in S_{edge}, y_j \in \mathbb{N}\}$$

其中，y_j 表示在边缘服务器 s_j 上运行的**服务实例数量**。在微服务架构中，通过在多个节点上运行服务的多个实例来实现负载均衡，每个节点可以托管服务的多个实例。

C. 问题公式化

1）响应时间

设 t_{ij} 为连接到服务器 s_j 的用户 u_i 获取服务的响应时间，包括三部分，数据**传输延迟** $t_{send_{ij}}$、**处理延迟** $t_{p_{ij}}$ **及传播延迟** $t_{d_{ij}}$：

$$t_{ij} = t_{send_{ij}} + t_{p_{ij}} + t_{d_{i,j}}$$

由于边缘服务器与云服务器的差异，其响应时间的计算方式也有所不同，如下。

a. 边缘服务器响应时间 $t_{ij}^e = t_{send_{ij}}^e + t_{p_{ij}}^e + t_{d_{i,j}}^e$

- **传输延迟** $t_{send_{ij}}^e = \frac{D_i^{in} + D_i^{out}}{b_{ij}^e}$
  - 表示传输**数据量**与用户服务器之间的**带宽**比
- **处理延迟** $t_{p_{ij}}^e = \frac{p_i}{P_{ij}}$
  - 为处理请求需要的**总计算单位**与**服务实例分配给用户**的**计算能力**之比
  - 式中的 P_{ij} 为边缘服务器 s_j 上的服务实例分配给用户 u_i 的计算能力，计算公式为：

    $$P_{ij} = \beta \cdot p_i \cdot W_i$$

    - β 为动态调整比例因子，用来控制总分配不能超过服务器能力。
    - 通过该分配方式，用户能够根据其需求获得**相应的计算能力**，同时，通过**加权机制**，**高优先级用户**可获得**更多**计算资源，从而体现出模型对不同优先级用户计算能力分配的差异化策略。
- **传播延迟** $t_{d_{ij}}^e$
  - 即前面定义的用户与**边缘服务器**之间的基础**传播延迟**

b. 云服务器响应时间 $t_{ij}^c = t_{send_{ij}}^c + t_{p_{ij}}^c + t_{d_i}^c$

- **传输延迟** $t_{send_{ij}}^c = \frac{D_i^{in} + D_i^{out}}{b_i^c}$
  - 表示传输**数据量**与用户服务器之间的**带宽**比
- **处理延迟** $t_{p_{ij}}^c = \frac{p_i}{P_{cloud}}$
- 为处理请求需要的**总计算单位**与**云服务器**的**计算能力**之比
- **传播延迟** $t_{d_i}^c$
  - 即前面定义的用户与**云服务器**之间的基础**传播延迟**

即，用户 u_i 连接到服务器 s_j 请求服务的总响应时间为：

$$t_{ij} = \begin{cases} \frac{D_i^{in} + D_i^{out}}{b_{ij}^e} + \frac{p_i}{P_{ij}} + t_{d_{ij}}^e, & \text{用户发送请求到边缘服务器} s_j \\ \frac{D_i^{in} + D_i^{out}}{b_i^c} + \frac{p_i}{P_{cloud}} + t_{d_i}^c, & \text{用户发送请求到云服务器} \end{cases}$$

2）加权 Jain 公平性指数

设 F_{Jain} 为系统中所有用户的响应时间的加权 Jain 指数：

$$F_{\text{Jain}} = \frac{\left(\sum_{i=1}^{n} t_{ij}^{weight}\right)^2}{n \cdot \sum_{i=1}^{n} \left(t_{ij}^{weight}\right)^2}$$

其中，$t_{ij}^{weight} = t_{ij} \cdot W_i$ ，为用户 u_i 的加权响应时间；n 为用户总数。

- **公式解释**：
  - **分子**：所有用户加权响应时间的**总和平方**，反映响应时间的**集中程度**，值越大说明加权时间分布越均衡。
  - **分母**：所有用户加权响应时间的**平方和**，表示响应时间的离散程度，值越大说明时间差异较大。
  - 公式整体衡量了加权响应时间的均衡性，指数越**接近 1**，表明系统越公平。

- **加权作用**：
  - **影响分子**：**高权重**用户的**短响应时间**有助于**拉近用户间的加权响应时间**，提高公平性。
  - **影响分母**：**高权重**用户的**短响应时间**降低**分母增长速度**，使指数**更接近 1**。

- **作用分析**：
  - **提升同优先级用户的公平性**：加权 Jain 指数的最大化保证了**相同优先级**用户的加权响应时间接近，从而减少同级用户间的响应时间差异，提高系统内部的公平性。
  - **保障高优先级用户的低响应时间**：在资源受限的情况下，权重机制确保高优先级用户获得更多计算资源，使其响应时间更短。
  - **整体公平性优化**：Jain 指数越大，说明系统既保证了**相同优先级**用户的响应时间相近，又确保了**跨优先级**用户的响应时间符合优先级设定，即优先级越高的用户响应时间越短，确保了整个系统响应时间的公平性。

3）约束条件

1. 不同优先级用户的平均响应时间约束

各优先级用户的平均响应时间不能超过其最大响应时间限制：

$$\frac{1}{|U_{L_i}|} \sum_{u_j \in U_{L_i}} \sum_{s_k \in S} x_{jk} \cdot t_{jk} \leq T_{L_i}^{max}, \quad \forall L_i$$

其中，$T_{L_i}^{max}$ 为优先级 L_i 的用户的最大平均响应时间。

2. 成本约束

系统中的成本不得超过成本预算 C_{max}：

$$C_{edge} + C_{cloud} \leq C_{max}$$

其中：

- C_{edge} 为**边缘服务器**的总成本，计算公式为：

$$C_{edge} = \sum_{s_j \in S_{edge}} c_{fixed_j}^{e}$$

- C_{cloud} 为**云服务器**的总成本，计算公式为：

$$C_{cloud} = \sum_{u_i \in U, s_j \in S_{cloud}} p_{net} \cdot p_i \cdot x_{ij}$$

3. 用户与服务器的连接约束

为确保所有用户的请求都被处理，每个用户 u_i 必须连接到唯一一个服务器：

$$\sum_{\forall s_j \in S} x_{ij} = 1, \quad \forall u_i \in U$$

4. 服务实例计算能力约束

连接到同一服务器的所有用户的计算能力需求并不能超过**该服务器上**部署的服务实例的总计算能力：

$$\sum_{u_i \in U} x_{ij} \cdot p_{ij} \leq y_j \cdot p^m, \quad \forall s_j \in S_{edge}$$

5. 边缘服务器计算资源约束

每个边缘服务器上的服务实例所需总资源不超过服务器可提供的资源：

$$y_j \cdot r^m \leq R_j^{server}, \quad \forall s_j \in S_{edge}$$

6. 边缘服务器带宽约束

连接到边缘服务器的 s_j 的所有用户的带宽总和不得超过该边缘服务器的最大可用带宽：

$$\sum_{u_i \in U} x_{ij} \cdot b_{i,j}^e \leq b_j^e, \quad \forall s_j \in S_{edge}$$

4）问题定义

在由 **私有边缘集群** S_{edge} **和公共云** S_{cloud} 组成的微服务系统中，目标是在 **满足各项约束条件** 的前提下 **优化用户与服务器连接关系 X 与微服务部署方案 Y**，以最大化系统的**公平性。**本研究采用 **用户响应时间** 作为衡量用户服务体验的关键指标，即通过最大化系统所有用户响应时间的**加权 Jain 指数**，优化不同用户在系统中的服务获得情况，从而提升整体公平性。

$$max \quad F_{jain}$$

$$s.t. \quad \frac{1}{|U_{L_i}|} \sum_{u_j \in U_{L_i}} \sum_{s_k \in S} x_{jk} \cdot t_{jk} \leq T_{L_i}^{max}, \quad \forall L_i$$

$$C_{edge} + C_{cloud} \leq C_{max}$$

$$\sum_{\forall s_j \in S} x_{ij} = 1, \quad \forall u_i \in U$$

$$\sum_{u_i \in U} x_{ij} \cdot p_{ij} \leq y_j \cdot p^m, \quad \forall s_j \in S_{edge}$$

$$y_j \cdot r^m \leq R_j^{server}, \quad \forall s_j \in S_{edge}$$

$$\sum_{u_i \in U} x_{ij} \cdot b_{i,j}^e \leq b_j^e, \quad \forall s_j \in S_{edge}$$

D.优化问题求解

Greedy Simulated Annealing Fairness Optimization (GSAFO)

- **Greedy**（贪心）：初始分配用户到服务器
- **Simulated Annealing**（模拟退火）：优化分配方案
- **Fairness Optimization**（公平性优化）：最大化加权 Jain 指数

1. 首先为基于贪心策略的优化方法

它旨在按用户**优先级**分配服务器。贪心算法的核心思想是**逐步**选择当前最优的服务器，以最大化 Jain 公平性指数。

```
Algorithm GreedyAlgorithm
Require: Users, Servers, Constraints
Ensure: Optimized connection relation X, Best Jain index

1: X ← Initialize connection relation
2: Sort users by priority in descending order
3: attempt_count ← 0
4: valid_X ← false
5: while valid_X = false do
6:     X ← Initialize connection relation
7:     Initialize server computing capacity
8:     for each user u do
9:         best_server ← null
10:        best_jain ← -1
11:        potential_servers ← Get available servers for u
12:        for each server s in potential_servers do
13:            temp_X ← Copy X
14:            temp_X[u][s] ← 1
15:            jain_index ← Calculate weighted Jain index for temp_X
16:            if jain_index > best_jain then
17:                best_jain ← jain_index
18:                best_server ← s
19:            end if
20:        end for
21:        if best_server ≠ null then
22:            X[u][best_server] ← 1
23:            Update server computing capacity
24:        end if
25:    end for
26:    valid_X ← Check if X meets all constraints
27:    attempt_count ← attempt_count + 1
28:    if attempt_count > 100 then
29:        break
30:    end if
31: end while
```

```
32: X, best_jain ← SimulatedAnnealing(X)
33: return X, best_jain
```

算法分析

1. **用户排序**：按照用户的**优先级**对用户集合 U 进行**降序**排序，得到排序后的用户列表 U_{sorted}；优先级**较高**的用户将优先被分配服务器。
2. **服务器选择**：对于排序后的每个用户 $u_i \in U_{sorted}$，遍历所有**可用**服务器 S，计算将该用户分配到每个服务器时的加权 Jain 公平性指数。选择使得加权 Jain **公平性指数最大**且**满足服务器资源约束**的服务器 s_j。
3. **资源更新**：将用户 u_i 分配到选定的服务器 s_j 后，更新该服务器的可用资源。
4. **约束检查**：在完成所有用户的分配后，检查分配方案是否满足**所有约束条件**，如不同优先级用户的平均响应时间约束、成本约束等。如果满足，则输出该分配方案

复杂度分析

- **时间复杂度：**
  - 假设用户数量为 n，服务器数量为 m。
  - **排序**用户的时间复杂度为 $O(n \log n)$。
  - 对于每个用户，需要**遍历**所有服务器来计算加权 Jain 公平性指数，因此服务器选择的时间复杂度为 $O(n \cdot m)$。
  - 每次计算加权 Jain 公平性指数需要遍历所有用户，时间复杂度为 $O(n)$。
  - 因此，贪心算法的总时间复杂度为 $O(n^2 \cdot m + n \log n)$。
- **空间复杂度：**
  - 主要用于存储用户和服务器的信息，以及分配方案，空间复杂度为 $O(n \cdot m)$。

2. 模拟退火（SA）优化过程

它的目的是在贪心算法之后**进一步优化解决方案**，**避免陷入局部最优。**

```
Algorithm SimulatedAnnealing
Require: Connection relation X, Constraints
Ensure: Optimized connection relation best_X, Best Jain index

1: prev_jain ← Calculate weighted Jain index for X
2: best_jain ← prev_jain
3: best_X ← Copy X
4: temp ← Initial temperature
5: for iter from 1 to max_iters do
6:     improved ← false
7:     op ← Randomly select an operation
8:     new_X ← Copy X
9:    Apply operation op to new_X
10:    if new_X meets all constraints then
11:        new_jain ← Calculate weighted Jain index for new_X
12:        delta ← new_jain - prev_jain
13:        prob_accept ← exp(max(delta / temp, -500))
14:        if delta > (1e - 3 * temp / Initial temperature) or prob_accept > Random float
then
15:            X ← new_X
16:            prev_jain ← new_jain
17:            improved ← true
```

```
        end if
        if new_jain > best_jain then
            best_jain ← new_jain
            best_X ← Copy new_X
        end if
    end if
    temp ← temp * alpha
    if not improved then
        break
    end if
end for
return best_X, best_jain
```

算法分析

1. **初始解**：以**贪心算法**得到的分配方案作为**初始解** X_0。
2. **邻域搜索**：**随机**选择几种操作（交换两个用户的服务器分配、随机重新分配一个用户的服务器或子集重新分配）对当前解 X_k 进行扰动，得到新解 X_{k+1}。
3. **约束检查**：检查新解 X_{k+1} 是否满足所有约束条件。如果不满足，则舍弃该解，继续进行邻域搜索。
4. **接受准则**：计算新解的加权 Jain 公平性指数 $F_{Jain}(X_{k+1})$ 和当前解的加权 Jain 公平性指数 $F_{Jain}(X_k)$，并计算差值 $\Delta F = F_{Jain}(X_{k+1}) - F_{Jain}(X_k)$。如果 $\Delta F > 0$，则接受新解；否则，以一定的概率 $P = \exp(\frac{\Delta F}{T})$ 接受新解，其中 T 为当前温度。
5. **温度衰减**：随着迭代次数的增加，温度 T 逐渐降低，使得算法逐渐收敛到一个最优解。温度衰减公式为 $T_{k+1} = \alpha \cdot T_k$，其中 α 为温度衰减率，$0 < \alpha < 1$。

复杂度分析

- **时间复杂度**：
  - 假设最大迭代次数为 K，每次迭代需要进行邻域搜索、约束检查和接受准则判断。
  - 邻域搜索的时间复杂度为 $O(1)$，约束检查的时间复杂度为 $O(n \cdot m)$，接受准则判断的时间复杂度为 $O(1)$。
  - 因此，模拟退火算法的总时间复杂度为 $O(K \cdot n \cdot m)$。
- **空间复杂度**：主要用于存储当前解和新解，空间复杂度为 $O(n \cdot m)$。

GSAFO 算法总结

GSAFO（Greedy - Simulated Annealing for Fairness Optimization，基于贪心 - 模拟退火的公平性优化算法）旨在解决云 - 边协同环境下微服务部署的公平性问题。它先利用**基于贪心的优化方法**按用户**优先级**分配服务器，以**最大化加权Jain公平性指数**并考虑资源约束，快速得到初始可行解；之后借助**模拟退火算法**，通过随机操作搜索解空间，结合约束检查和温度衰减规则优化解，**跳出局部最优**，最终实现用户与服务器**连接关系**及**微服务部署方案**的优化，**提升系统公平性**，满足**响应时间、成本、资源等**多种约束 。