# 数据结构 Project: 服务器集群负载均衡优化器

## 1 摘要

针对大规模计算集群中存在的负载不均衡、任务迁移成本高及网络带宽受限等问题，本项目基于图论与启发式搜索算法旨在设计一个服务器集群负载均衡优化器。首先利用 Floyd-Warshall 算法计算出图中任意两点之间的最短路径，建立迁移成本模型。然后在任务分配阶段，利用贪心策略生成初始可行分配方案，并采用模拟退火算法优化，在贪心解的基础上，通过随机调整任务的目标节点并以一定概率接受暂时较差的解，寻找全局更优解。针对网络带宽限制，将迁移过程划分为多个时间步，并在每一步中检查链路带宽使用情况，仅允许未超限的任务进行移动，从而生成每一步的迁移计划。实验表明，本方案能够快速生成优化的可行解，并能正确处理网络带宽竞争与排队等情况。

## 2 数据结构设计

### 2.1 图结构的表示

考虑到题目中服务器节点数量 $N \leq 50$，采用邻接矩阵实现 $O(1)$ 的查询效率。

```
const int MAXN = 55;
int dist[MAXN][MAXN];            // 存储最短路径成本
int adj_bandwidth[MAXN][MAXN];   // 存储链路带宽
int next_hop[MAXN][MAXN];        // 路由表
```

### 2.2 实体定义

节点：

```
struct Node {
    int id;
    int capacity;        // 节点的总容量限制
    int current_usage;   // 当前已分配的任务总负载
};
```

任务：

```cpp
struct Task {
    int id, start_node, demand; // 静态属性
    int end_node;               // 目标节点
    int migration_cost;         // 迁移代价

    // 模拟属性
    vector<int> path;           // 迁移路径序列
    size_t path_idx;            // 当前路径索引
    int current_pos_node;       // 当前所在位置
    bool finished;              // 完成标记
};
```

# 3 算法设计与实现

本程序的目标是在满足多重约束（节点容量、网络带宽）的前提下，解决一个 NP-Hard 组合优化问题。核心算法分为三个阶段：预处理、规划优化、过程模拟。

## 3.1 预处理：全图最短路径 (Floyd-Warshall)

为了计算任意任务从 *Start* 到 *End* 的最小迁移成本，首先计算全图的最短路径。

采用 Floyd 算法，一次运行即可求出全图最短路径和路由表，将图论路径搜索问题转化为 O(1) 的查表操作，在后续的模拟退火迭代中，计算成本函数时无需重复搜索路径。在此过程中，同时维护 `next_hop` 矩阵，用于后续重建具体的迁移路径。

虽然 Floyd 算法的时间复杂度为 $O(N^3)$，但鉴于 $N \leq 50$，耗时可忽略不计。

核心代码实现如下：

```cpp
// 三重循环计算全源最短路径
for (int k = 1; k <= N; ++k) {
    for (int i = 1; i <= N; ++i) {
        for (int j = 1; j <= N; ++j) {
            // 状态转移：若经过 k 中转更近，则更新
            if (dist[i][k] + dist[k][j] < dist[i][j]) {
                dist[i][j] = dist[i][k] + dist[k][j];
                next_hop[i][j] = next_hop[i][k]; // 维护路由表
            }
        }
    }
}
```

## 3.2 核心规划：混合优化策略

### 3.2.1 贪心策略初始化—基础要求

为避免算法从随机解开始搜索导致收敛缓慢，首先使用贪心算法生成一个初始解：

- 排序：将所有任务按需求量降序排列，解决大任务难以安放的问题。
- 分配：依次为每个任务寻找当前剩余容量足够、且迁移成本最低的目标节点。

### 3.2.2 时间控制模拟退火—高级要求

贪心算法容易陷入局部最优。为了寻找全局最优解，采用模拟退火算法，以随机扰动和概率接受差解的机制进行优化。传统的模拟退火使用固定迭代次数。但在测评环境中，计算能力不确定。故使用基于时钟的终止条件，使得程序在有限时间内进行数百万次迭代搜索，并配合重热机制，防止算法过早冻结在某个局部最优解上。

**Algorithm 1** 基于时间的模拟退火优化

初始化温度 $T = 2000.0$，时间限制 $TimeLimit = 1.8s$
当前解 $\leftarrow$ 贪心解
**while** 运行时间 $< TimeLimit$ **do**
  随机选择一个任务 $t$ 和一个新节点 $node_{new}$
  **if** $node_{new}$ 容量满足约束 **then**
    计算成本变化 $\Delta E = Cost_{new} - Cost_{old}$
    **if** $\Delta E < 0$ **or** $exp(-\Delta E/T) > Random(0, 1)$ **then**
      接受新解：更新任务目标与节点负载
      **if** 当前解优于全局最优 **then**
        更新全局最优解
      **end if**
    **end if**
  **end if**
  降温 $T = T \times 0.999$
  **if** $T < T_{min}$ (过冷) **then**
    $T = T_{start} \times 0.5$ **(**重热机制，防止过早收敛**)**
  **end if**
**end while**

## 3.3 过程模拟：生成具体迁移计划—附加要求

规划层只解决了起点和终点的问题，没有考虑带宽限制。为了满足附加要求中的带宽约束，将连续的迁移过程离散化为时间步，实现基于时间步的具体迁移计划：

- 资源竞争：使用 map<int, int> 记录当前时间步每条物理链路的占用情况。
- 排队机制：若某条链路当前占用量 ≥ 带宽上限，则后续申请该链路的任务本轮保持静止，直到下一轮次再次尝试。
- 状态更新：仅当获得带宽资源后，任务才移动到路径的下一步。

离散事件模拟中的带宽竞争逻辑如下：

```
// 生成物理链路的唯一 Key
int key = (u < v) ? (u * 1000 + v) : (v * 1000 + u);

// 检查当前时间步该链路是否已满
if (link_usage[key] < adj_bandwidth[u][v]) {
    link_usage[key]++;            // 占用一个带宽名额
    tasks_moved_indices.push_back(i); // 标记任务可移动
}
// 否则：任务本轮保持静止，不加入移动列表
```

# 4 实验结果与分析

本项目完成了基础、高级、附加要求，能够快速生成优化的可行解，同时能够正确处理网络带宽竞争与排队问题。

## 4.1 附加要求的输出格式说明

本程序设计的日志输出格式为：time_step task_id from_node to_node

time_step：时钟，表示该动作发生在第几轮。

task_id：正在进行移动的任务编号。

from_node：该任务在本轮移动前所在的节点。

to_node：该任务在本轮移动后到达的节点。

示例：输出行 1 2 1 3 表示：在第 1 个时间步，任务 2 成功抢占了带宽资源，从节点 1 移动到了节点 3。

## 4.2 输入数据

采用说明文档提供的高级要求的测试样例进行测试。

```
3 3 5          // N=3节点, M=3链路, T=5任务
1 1            // 节点1: 容量1
2 3            // 节点2: 容量3
```

```
3 4             // 节点3: 容量4
1 2 4 3         // 链路1-2: 迁移成本4, 带宽3
1 3 2 1         // 链路1-3: 迁移成本2, 带宽1
2 3 1 2         // 链路2-3: 迁移成本1, 带宽2
1 1 1           // 任务1: 在节点1, 需求1
2 1 2           // 任务2: 在节点1, 需求2
3 1 3           // 任务3: 在节点1, 需求3
4 3 1           // 任务4: 在节点3, 需求1
5 1 1           // 任务5: 在节点1, 需求1
```

## 4.3 实验输出结果

程序运行后输出：

```
1 1 1 0
2 1 2 6
3 1 3 6
4 3 3 0
5 1 2 3
1 1
2 3
3 4
15
4
1 2 1 3
2 2 3 2
2 3 1 3
3 5 1 3
4 5 3 2
```

输出结果解释：

```
//  1. 任务分配详情
1 1 1 0     // Task 1: 留在节点1, 成本 0
2 1 2 6     // Task 2: 1->3->2, 路径成本(2+1)*2 = 6
3 1 3 6     // Task 3: 1->3, 路径成本 2*3 = 6
4 3 3 0     // Task 4: 留在节点3, 成本 0
5 1 2 3     // Task 5: 1->3->2, 路径成本(2+1)*1 = 3
// 2. 节点最终负载
1 1         // Node 1: 负载1/1 (Task 1) -> 饱和
2 3         // Node 2: 负载3/3 (Task 2, 5) -> 饱和
```

```
3 4          // Node 3: 负载4/4 (Task 3, 4) -> 饱和
// 3. 核心指标
15           // 总迁移成本
4            // 完成迁移所需的时间步
// 4. 迁移过程日志
1 2 1 3      // T=1: Task 2 抢占链路 1->3
2 2 3 2      // T=2: Task 2 离开 3->2
2 3 1 3      // T=2: 链路 1->3 空闲, Task 3 通过
3 5 1 3      // T=3: 链路 1->3 空闲, Task 5 通过
4 5 3 2      // T=4: Task 5 到达终点
```

## 4.4 结果分析

注意到，本程序输出的任务分配方案与 PDF 提供的两种参考答案均不同，但经数学验证，三者均为全局最优解。

参考答案将大任务 Task 3 分配至 Node 2（远端），迁移成本为 $3 \times 3 = 9$；将 Task 2 分配至 Node 3（近端），成本 $2 \times 2 = 4$。本程序将大任务 Task 3 分配至 Node 3（近端），迁移成本为 $2 \times 3 = 6$；将 Task 2 分配至 Node 2（远端），成本 $3 \times 2 = 6$。

参考答案总成本：$0(\text{T1}) + 4(\text{T2}) + 9(\text{T3}) + 0(\text{T4}) + 2(\text{T5}) = 15$。

本程序总成本：$0(\text{T1}) + 6(\text{T2}) + 6(\text{T3}) + 0(\text{T4}) + 3(\text{T5}) = 15$。

本程序在此样例下为全局最优解，正确处理了最短路径计算，输出结果正确。

## 4.5 基础要求、高级要求、附加要求的对比与综合分析

本节以说明文档提供的高级要求的测试样例为例，综合对比并分析基础要求、高级要求与附加要求在迁移成本与完成时间上的差异。

### 4.5.1 对比场景

- 基础场景：仅考虑连通性，优先走直连边，忽略路径权重，按任务输入顺序，将其分配给第一个有剩余容量的节点。数据基于理论推演。
- 高级场景：假设无限带宽。使用 Floyd-Warshall 算法计算基于权重的全图最短路径，并采用降序贪心算法与模拟退火算法，寻找成本最低的分配组合。数据来源本程序 `optimizeAllocationSA` 阶段输出的 `migration_cost`。
- 附加场景：在高级场景的分配方案基础上，引入物理链路带宽限制。数据来源本程序 `simulateMigration` 阶段输出的 `total_time_steps`。

#### 4.5.2 基础方案结果推演

假设按输入顺序分配任务（T1, T2, T3, T4, T5）且走直连边：

- T1 (Dem 1) → Node 1 (满)。
- T2 (Dem 2) → Node 2 (直连 1-2)，成本 $4 \times 2 = 8$。
- T3 (Dem 3) → Node 3 (直连 1-3)，成本 $2 \times 3 = 6$。
- T4 (Dem 1) → Node 3 (原地)，成本 0。
- T5 (Dem 1) → Node 2 (直连 1-2)，成本 $4 \times 1 = 4$。

基础方案总成本：$0 + 8 + 6 + 0 + 4 = \mathbf{18}$。

#### 4.5.3 综合对比

表 1: 各阶段性能指标对比

| 指标 | 基础场景 | 高级场景 | 附加场景 |
|---|---|---|---|
| 路径策略 | 直连优先 | Floyd 最短路 | Floyd 最短路 |
| 分配策略 | 首次适应 | 模拟退火 | 模拟退火 |
| 总迁移成本 | 18 | 15 | 15 |
| 迁移轮次 | 1 (理想) | 1 (理想) | 4 (受带宽限制) |

#### 4.5.4 差异分析

1. 基础要求与高级要求的总迁移成本差异分析

   - 样例实验现象：总迁移成本从 18 降低至 15。
   - 原因：基础要求往往倾向于选择直连边，忽略了潜在的更优间接路径。高级要求使用了 Floyd-Warshall 算法，同时采用模拟退火算法，通过全局搜索避免将大任务分配到高成本节点，总迁移成本更低。

2. 高级要求与附加要求的总迁移成本与迁移轮次梳理差异分析

   - 总迁移成本一致性：

     本项目采用规划与执行分离的设计，在规划阶段，算法仅根据最短路径距离和任务大小确定源与目标，此时成本即被确定。在执行阶段，带宽限制仅决定任务何时移动，负责调度移动时间，不改变任务经过，故总成本保持不变。

   - 迁移轮次差异：

  - 在高级要求下，假设带宽无穷大，所有任务可在一个时间步内完成迁移。
  - 在附加要求下，在测试样例中总时间步增加至 4。
- 原因：带宽限制引入了排队延迟。在测试样例中，链路 $1 \to 3$ 带宽容量 $BW = 1$，而有 3 个任务（Task 2, 3, 5）需要通过该链路，故至少需要 3 个时间步才能让这 3 个任务依次通过。再加上部分任务需要走两步（$1 \to 3 \to 2$），最终耗时为 4 个时间步。这表明带宽限制会导致迁移轮次不变或增加。

# 5 多目标优化权衡思考

在完成附加要求的过程中，本程序采用成本优先策略，适用于对成本敏感的场景。

## 5.1 成本优先与时间优先

本程序当前采用的是成本优先策略，所有任务严格遵循 Floyd 算法计算出的最短路径进行迁移。然而，当多条最短路径重合在某一低带宽链路时，任务被迫排队等待，会导致总迁移轮次 ($T_{max}$) 增加。这种成本优先策略虽然保证了总迁移成本最低，但在高并发场景下可能导致较长的迁移时间。

## 5.2 时间优化算法设想

若实际场景对完成时间要求较高，可采用 K-最短路径负载均衡策略优化迁移轮次：

当最短路径（Path 1）拥塞时，动态评估次短路径（Path 2），当排队等待的代价超过绕路的额外成本时，选择绕行带宽更充裕的非最短路径。

$$\text{If } (WaitTime_{path1} > \text{Threshold}) \text{ then choose } Path_2$$

这种方法虽然会使总迁移成本有所上升，但能减少链路拥塞，缩短总完成时间。

# 6 总结与心得

在解决任务分配这一组合优化问题时，我体会到贪心算法与启发式算法的区别。贪心算法虽然速度快，但受限于视界；而模拟退火算法通过引入随机性和 Metropolis 准则，允许暂时接受差解以跳出局部陷阱。

在处理附加要求时，采用了规划与执行分离的设计模式。规划层专注于优化静态成本最短路径，执行层专注于处理动态约束带宽排队。这种分离降低了代码复杂度，也让我理解了理论最优成本与实际完成时间之间的辩证关系。