数据结构 Project：服务器集群负载均衡优化器

数据结构

2025 年秋季学期

1 简要信息

项目名称：服务器集群负载均衡优化器

背景描述：随着大语言模型、AIGC 应用的迅速发展，云平台通常会部署成规模的服务器/GPU 集群，作为集中算力平台，用于同时为众多用户提供计算服务 (包括大语言模型的推理、训练以及其它计算密集型任务)。在真实运行过程中，不同节点之间的负载往往极不均衡：

- 有的节点任务队列堆积，GPU 长期处于高负载，资源紧张；
- 有的节点则相对空闲，资源利用率较低。

为了提高整体资源利用率、减少任务等待时间，需要在服务器之间迁移部分任务以实现负载均衡。然而，任务迁移并非零成本：

- 迁移过程中需要在节点之间传输程序代码、模型参数、中间状态等；
- 迁移过程中会占用网络带宽，因此在任务迁移优化中需要显式考虑网络约束。

因此，本项目的目标是设计并实现一个服务器集群负载均衡优化器：在给定任务信息、服务器拓扑结构和网络带宽约束的前提下，自动生成任务迁移方案，平衡各节点负载，并最小化总迁移成本。

考察知识点：此项目主要涉及图结构的表示与遍历、最短路径计算等基础数据结构与算法在负载均衡与任务迁移场景中的实际应用。

提交截止日期： 2025.12.28 23:59

2 Project 要求

2.1 基本要求

2.1.1 实现基础数据结构

(1) 节点信息数据结构

- 记录服务器节点集合：包括节点编号、节点可用 GPU 数量和当前已使用的 GPU 数量；
- 记录节点之间的网络连接：包括是否连通，以及在两节点之间迁移 “单个 GPU 占用” 所需要的成本。

(2) 任务信息数据结构

- 记录每个任务的编号、初始所在节点；
- 记录每个任务所需的 GPU 数量。

2.1.2 为多个任务生成迁移方案，实现负载平衡

在不考虑迁移成本和网络带宽的前提下，给定：

- 各节点的最大可用 GPU 数及当前 GPU 占用情况；
- 每个任务的初始所在节点。

系统需要输出一份任务迁移方案，使得：

- 每个任务最终被分配到某个节点上 (可以留在原节点，也可以迁移到其他节点)；
- 任意节点上的任务总 GPU 占用量不超过该节点的最大可用 GPU 数；
- 本阶段不要求迁移成本最小化，只需给出一份满足容量约束的可行方案。

2.2 高级要求

在完成基础要求的前提下，引入迁移成本优化目标：尽量减少总迁移成本 (所有任务从原节点迁移到目标节点的迁移成本之和)。

*** 注**：不强制要求全局最优解，但需要在报告中：说明的优化策略设计思路；对比基础要求与高级要求的总迁移成本，并分析差异。

2.3 附加要求

在高级要求的基础上，进一步引入网络带宽限制这一现实约束：

- 为每条网络链路（边）设定带宽容量，表示在同一时间内最多可迁移的任务数量 (单个任务不可进行拆分)；

- 当某条链路已达到带宽上限时，当前时间轮次下，不允许通过该链路继续迁移新的任务。

*** 注**: 本阶段同样不要求求解理论上的全局最优，只需：给出在带宽约束下仍然可行的迁移方案；但需要在报告中：对比高级要求与附加要求在：总迁移成本、迁移轮次数量（完成负载平衡所需的时间步数）等指标上的差异，并进行简单分析。

3 实现细节

3.1 输入格式

输入共包含集群规模、节点信息、网络链路信息、任务信息四部分，依次输入，具体格式如下：

3.1.1 集群与任务基本信息

第一行包含 3 个正整数（空格分隔）：

```
N M T
```

- **N**：服务器节点数量 ($1 \leq N \leq 50$)
- **M**：网络链路数量
- **T**：任务数量

节点默认编号为 $1 \sim N$，任务默认编号为 $1 \sim T$

3.1.2 节点信息

接下来输入 N 行，每行描述一个节点的信息：

```
node_id  capacity
```

- **node_id**：节点编号（$1 \sim N$）
- **capacity**：该节点的最大可用 GPU 总数（正整数）

3.1.3 网络链路信息（图的边信息）

接下来输入 M 行，每行描述一条网络连接（无向链路）：

```
u  v  path_cost  bandwidth
```

- **u, v**：相连的两个节点编号（$1 \sim N$）

- **path_cost**：在这条链路上迁移 “单个 GPU 占用” 所需的成本（用于高级要求中的迁移成本统计与优化）
- **bandwidth**：该链路的带宽容量（正整数），表示在同一时间单位内最多可通过的任务总量

在基础要求中：只需要利用 u 和 v 建立图的连通关系；不要求使用 path_cost 和 bandwidth，读入之后不使用。

在高级要求中：path_cost 作为边权，用于计算任务从一个节点到另一个节点的迁移代价（例如最短路径）；bandwidth 在本阶段仍可忽略（不参与计算）。

在附加要求中：同时使用 u, v, path_cost, bandwidth：bandwidth 用作链路容量限制，在规划迁移路径和时间轮次时需要检查是否超过带宽上限。

3.1.4 任务信息

最后输入 T 行，每行描述一个任务的信息：

```
task_id  start_node  demand
```

- **task_id**：任务编号（1 ∼ T）
- **start_node**：任务初始所在的节点编号（1 ∼ N）
- **demand**：该任务所需的 GPU 数量（正整数）

说明：某个节点的初始负载可由所有初始在该节点的任务的 demand 之和计算得到。

3.2 输出格式

3.2.1 基础要求：可行的负载平衡方案

在基础要求中，只需保证：

- 所有任务都有最终归属节点；
- 任意节点最终负载不超过其容量。

1）任务最终分配结果

输出 T 行，第 i 行对应任务 i 的最终信息，格式为：

```
task_id  start_node  end_node
```

其中：

- **task_id**：任务编号（1 ∼ T），与输入一致；
- **start_node**：任务初始所在节点编号；
- **end_node**：任务最终所在节点编号（可以与 start_node 相同，表示未迁移）。

说明：基础要求不要求迁移成本最优，end_node 只需使整体分配满足容量约束即可。

2）各节点最终负载

接着输出 N 行，每行一个节点的最终负载情况：

```
node_id  final_load
```

其中：

- **node_id**：节点编号（1 ∼ N）；
- **final_load**：该节点上所有任务 demand 之和（应不超过 capacity）。

3.2.2 高级要求：在容量约束下优化迁移成本

高级要求中，在满足基础要求所有约束的基础上，需要统计并尽量优化总迁移成本。

1）任务最终分配结果（含迁移成本）输出 T 行，第 i 行对应任务 i 的最终信息，格式为：

```
task_id  start_node  end_node  migration_cost
```

其中：

- **task_id**：任务编号（1 ∼ T）；
- **start_node**：任务初始所在节点编号；
- **end_node**：任务最终所在节点编号；
- **migration_cost**：该任务从 start_node 迁移到 end_node 的路径代价（最短路径上各链路 path_cost × 该任务 demand 之和；若未迁移则为 0）。

2）各节点最终负载

同基础要求，输出 N 行：

```
node_id  final_load
```

3）总迁移成本

最后输出 1 行，为所有任务的总迁移成本：

```
total_migration_cost
```

其中 total_migration_cost = 所有任务的 migration_cost 之和。

3.2.3 附加要求：在带宽约束下的迁移方案

在附加要求中，在高级要求的基础上进一步考虑链路带宽容量限制，迁移过程可能需要分成多个时间轮次（time steps）完成。

在保持高级要求输出内容的基础上，可以额外输出如下信息：

1）完成负载平衡所需的总时间轮次数

```
time_steps
```

其中 time_steps：在带宽约束下，完成全部任务迁移所需的时间轮次数量。

2）按时间步输出迁移计划

如需更详细展示迁移过程，可以输出每一轮中发生的迁移，例如：

```
time_step  task_id  from_node  to_node
```

含义：

- **time_step**：第几轮迁移（从 1 开始编号）；
- **task_id**：本轮迁移的任务编号；
- **from_node**：任务迁移前所在节点；
- **to_node**：任务迁移后所在节点。

附加要求的扩展输出不作强制统一要求，在实验报告中说明自己设计的输出格式与含义即可，并分析“带宽约束”对迁移成本与完成时间的影响。

3.3 基本要求测试样例

输入样例 1：

```
3 2 4
1 4
2 3
3 3
1 2 1 10
2 3 1 10
1 1 3
2 1 2
3 2 1
4 3 1
```

输出样例 1：

```
1 1 1
2 1 3
3 2 2
4 3 2
1 3
2 2
3 2
```

3.4 高级要求测试样例

输入样例:

```
3 3 5
1 1
2 3
3 4
1 2 4 3
1 3 2 1
2 3 1 2
1 1 1
2 1 2
3 1 3
4 3 1
5 1 1
```

输出样例:

当前情况下，任务 1 和任务 5 起始节点一样，任务需求量一样，因此会出现两种方案 (给出任一一种即可)。

```
1 1 1 0
2 1 3 4
3 1 2 9
4 3 3 0
5 1 3 2
1 1
2 3
3 4
15
```

```
1 1 3 2
2 1 3 4
3 1 2 9
4 3 3 0
5 1 1 0
1 1
2 3
3 4
15
```

4 提交要求

作业通过 eLearning 进行提交，提交文件应为一个文件名为学号，后缀名为 zip 的压缩包，压缩包示例结构如下：

```
student-id.zip
    |__ report.pdf
    |__ code
        |__ main.cpp
        |__ readme.txt (可选，详见项目评分)
```

report.pdf 要求：长度不宜超过 10 页，正文字号行距等格式以清晰易读为佳，内容包括解释整体实现思路、介绍已实现功能的思路、展示测试用例及结果、描述实现过程中收获 (非必须)。若仅有报告，无代码/抄袭代码/乱写代码，PJ 将酌情扣分。

代码文件要求：在 code 文件夹中，所有代码文件需要包含必要的中文注释，所有代码文件请使用 UTF-8 编码提交。

*** 注**：在完成 PJ 过程中若参考了互联网资料或获得了同学的帮助，请在报告的末尾注明，如未注明但在评分时发现有直接抄袭互联网资料或其他同学提交的代码等情况，PJ 将酌情扣分。

5 项目评分

5.1 评分组成

本项目的评分包括代码评分和报告评分。

5.2 代码评分

代码评分时会在 Ubuntu 22.04 环境下，对源文件进行编译。请选择如下 3 种方式之一提交代码文件。

(1) 提交单个 C 语言源文件

对于该方式，请将所有代码放在一个源文件中，文件命名为 main.c。

编译时，使用 gcc11 对 main.c 直接编译，编译命令为：

```
gcc -Wall -std=c17 -O2 main.c -o main
```

(2) 提交单个 C++ 语言源文件

对于该方式，请将所有代码放在一个源文件中，文件命名为 main.cpp。

编译时，使用 g++11 对 main.cpp 直接编译，编译命令为：

```
g++ -Wall -std=c++20 -O2 main.cpp -o main
```

(3) 提交包含多个 C/C++ 语言源文件

对于该方式，请将所有代码文件放在 `code` 文件夹中，并在 `code` 文件夹中放置一个 `readme.txt` 文件作为编译命令的说明。`readme.txt` 文件内仅可包含用于编译的命令（每行一条编译命令），不能包含额外的文字说明，命令中只可以使用 `gcc`（版本为 gcc11）和 `g++`（版本为 g++11），其中用于编译的命令（不包含链接等命令）需使用 `-O2` 选项。编译时，会按照 `readme.txt` 中的命令依次执行。

如不能正常编译和运行程序，或程序不能正常读入相应的测试样例，均会有较大程度的扣分。对于每一个测试样例，程序运行时长不能超过 20 秒。

5.3 报告评分

在代码能正确编译运行，并按要求正确的前提下，报告能说明整个程序的设计思路即可获得全部分数，请不要在报告中加入冗长的说明或复制大段的代码，报告格式尽量规范。