

**《计算机算法课程实践》**

**实验报告**

**实验（ 三 ）**

**年 级： 2024**

**专 业： 计算机技术**

**姓 名： 刘兴宇**

**学 号： 2020240739**

# 1 实验内容

本实验旨在设计并实现一个动态任务调度场景，用以对比和分析两种不同的贪心策略：**即时贪心**与**延迟批量贪心**。

实验的核心目标是探究在任务动态到达的环境下，决策的**时效性**（即立即决策还是延迟决策）以及决策所依据的**信息范围**（单个任务还是任务批次）对调度结果（特别是最大完成时间）产生的影响。

实验通过一个C++程序进行模拟，该程序接受以下可配置参数：

* **机器数量 (m)**：处理任务的并行机器总数。
* **初始任务序列 (static\_tasks)**：在 t=0 时刻已存在的一组任务。
* **动态任务序列 (dynamic\_tasks)**：在不同时间点依次到达的一组任务，每个任务包含到达时间和处理时间。
* **延迟阈值 (k)**：策略B中用于触发批量调度的缓存任务数量。

# 2 设计思想

程序整体采用C++语言编写，具备良好的结构和可扩展性。为了清晰地展示算法过程，程序会详细输出每一步的决策日志。其核心设计思想分为公共模块和两种独立的策略实现。

#### ****2.1. 公共模块设计****

* **数据结构**：定义了DynamicTask结构体来存储动态任务的到达时间和处理时间。使用std::vector<int>来追踪每台机器的当前总负载。
* **辅助函数**：设计了print\_loads用于打印当前所有机器的负载状态，以及find\_min\_load\_machine\_index用于查找并返回当前负载最小的机器索引。这两个函数被两种策略复用。
* **初始状态分配**：对于在 t=0 时刻存在的静态任务，统一采用\*\*LPT（Longest Processing Time，最长处理时间优先）\*\*的贪心思想进行初始分配。即将静态任务按处理时间降序排序，然后依次分配给当前负载最小的机器。这为两种策略的比较提供了一个公平且高效的初始状态。

#### ****2.2. 策略A：即时贪心****

该策略体现了“**立即响应**”的贪心思想。

* **机制**：当一个动态任务到达时，系统不进行任何等待或缓冲。
* **决策**：立即遍历所有机器，找出当前总负载最小的一台，并将该任务分配给它。
* **特点**：决策速度快，实现简单，每个决策仅基于当前瞬间的系统状态（局部信息）。

#### ****2.3. 策略B：延迟批量贪心****

该策略引入了“**延迟决策**”和“**批量处理**”的思想，试图通过等待来获取更全面的信息。

* **机制**：当一个动态任务到达时，先不分配，而是将其存入一个**缓存区 (buffer)**。
* **触发条件**：当缓存区中的任务数量达到预设的**阈值 k**，或者所有动态任务均已到达时，触发一次批量调度。
* **决策**： 对缓存区内的所有任务，按照**处理时间降序**进行排序（应用LPT思想）。将排序后的任务，按照从大到小的顺序，依次分配给当前负载最小的机器。清空缓存区，等待下一批任务。
* **特点**：通过牺牲即时性，换取了在一个“任务批次”内的全局视角，从而有机会做出更优的负载均衡决策。

# 3 程序效果

程序的运行效果，输入及输出的相关要求和具体执行结果如下所示：

## 3.1 输入

3

5

7 5 9 3 4

3

2 6

5 8

9 2

2

## 3.2 输出

--- Experiment Start ---

## Strategy A: Immediate Greedy

---

t=0: Static tasks assigned.

Initial loads: Machine1=9, Machine2=10, Machine3=9

Time=2: Task (a=2, p=6) has arrived

Current machine loads: Machine1=9, Machine2=10, Machine3=9

Assigned to Machine1 (minimum load, 9 -> 15)

Time=5: Task (a=5, p=8) has arrived

Current machine loads: Machine1=15, Machine2=10, Machine3=9

Assigned to Machine3 (minimum load, 9 -> 17)

Time=9: Task (a=9, p=2) has arrived

Current machine loads: Machine1=15, Machine2=10, Machine3=17

Assigned to Machine2 (minimum load, 10 -> 12)

---

\*\* Strategy A finished \*\*

Final loads: Machine1=15, Machine2=12, Machine3=17

Maximum Completion Time (Makespan): 17

============================================

## Strategy B: Delayed Batch Greedy

---

t=0: Static tasks assigned.

Initial loads: Machine1=9, Machine2=10, Machine3=9

Time=2: Task (a=2, p=6) has arrived

Threshold k=2 not reached, task is buffered.

Time=5: Task (a=5, p=8) has arrived

Buffer size reached 2, triggering batch dispatch...

Sorted buffer (descending by processing time): p=8 p=6

Current loads: Machine1=9, Machine2=10, Machine3=9

Assigning task p=8 to Machine1 (9 -> 17)

Current loads: Machine1=17, Machine2=10, Machine3=9

Assigning task p=6 to Machine3 (9 -> 15)

Loads after this batch: Machine1=17, Machine2=10, Machine3=15

Time=9: Task (a=9, p=2) has arrived

Buffer size reached 1, triggering batch dispatch...

Sorted buffer (descending by processing time): p=2

Current loads: Machine1=17, Machine2=10, Machine3=15

Assigning task p=2 to Machine2 (10 -> 12)

Loads after this batch: Machine1=17, Machine2=12, Machine3=15

---

\*\* Strategy B finished \*\*

Final loads: Machine1=17, Machine2=12, Machine3=15

Maximum Completion Time (Makespan): 17

# 4 算法分析

#### ****4.1. 算法对比与复杂度****

|  |  |  |
| --- | --- | --- |
| **特性** | **策略A: 即时贪心** | **策略B: 延迟批量贪心** |
| **决策时机** | **立即**：任务到达即决策。 | **延迟**：等待缓存区满或任务流结束。 |
| **信息范围** | **局部**：仅关心当前到达的单个任务。 | **半全局**：关心一个批次内的所有任务。 |
| **决策依据** | 简单的最小负载原则。 | 对批次应用**LPT**启发式策略。 |
| **性能潜力** | 容易陷入局部最优，可能导致负载不均。 | 通过LPT策略，通常能获得更好的负载均衡效果。 |
| **总复杂度** | O(Ns​logNs​+Nd​⋅m) | O(Ns​logNs​+Nd​(logk+m)) |

#### ****4.2. 深入理解时效性影响****

本次实验的核心在于理解“时效性”对贪心算法决策质量的影响。时效性在这里指系统做出调度决策的快慢程度。

**决策信息与时效性的关系**

**时效性越高，信息越少**。策略A（即时贪心）追求极致的时效性，在任务到达的瞬间就必须决策。此时，它所掌握的唯一“未来”信息就是这个刚到达的任务本身，对后续可能到来的任务一无所知。

**延迟换取信息**。策略B（延迟批量贪心）通过牺牲部分时效性（等待k个任务），换取了更大的信息视界。在决策时，它不仅知道这k个任务各自的属性，还能通过排序比较它们，从而做出更合理的安排。

**阈值 k 的角色：时效性的调节器**

延迟阈值 k 是控制这种权衡的**关键**。k 值越小，系统时效性越高，越接近策略A，响应快但决策质量可能不高。k 值越大，系统时效性越低，任务等待时间越长，但批量决策的信息更充分，越有可能获得优化的完工时间。然而，过大的 k 值也可能导致机器长时间空闲，等待缓存区填满，反而降低效率。

# 5 总结

本次实验加深了对贪心算法时效性的理解，并为解决实际负载均衡问题提供了有益的参考。在真实系统设计中，对k值的选择需要根据具体的业务场景和优化目标（是优先响应用户，还是优先提升系统效率）来决定。