Adaptive Weighted Multi-Tenant Scheduler

Authors

Mou Yangyi-1306078
Taoying Jia-1306194

Wenzhou Kean University
Dr. Hamza Djigal
CPS 3250 W05&06
November 30 2025

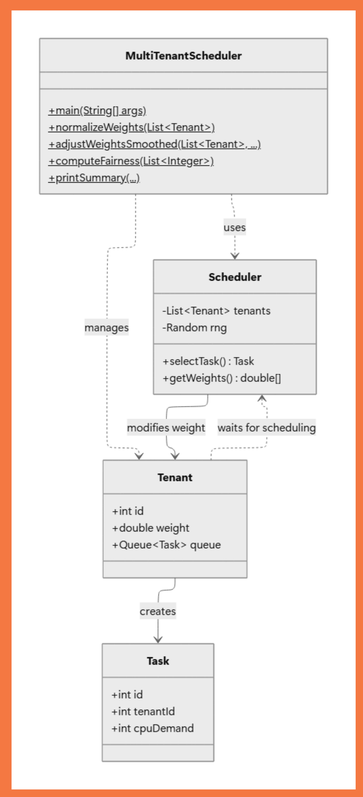
Introduction

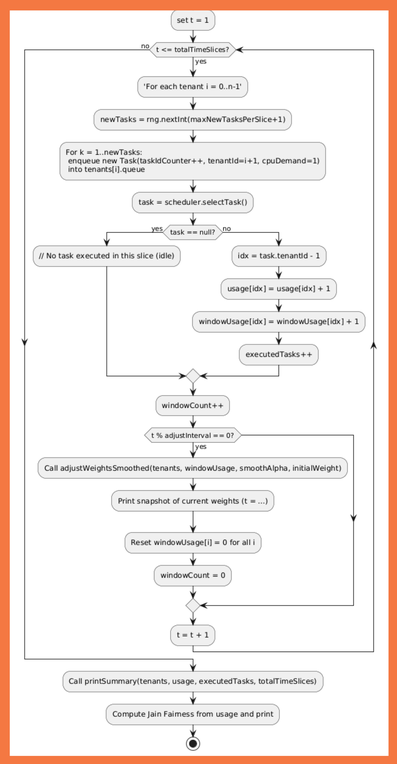
In modern cloud computing platforms, multiple tenants usually share physical resources. But if we continue to apply the traditional scheduling policies, we will find that certain tenants will consume more resources while other tenants are starved, resulting in poor fairness and utilization.

Objective

I The main aims:
1.Ensure long-term intended weighted fairness across multiple tenants with changing workloads.
2.Adapt dynamically to workload imbalance by adjusting scheduling weights.
3.Prevent starvation, even when some tenants temporarily generate heavy workloads.
4.Maintain good resource utilization while improving fairness metrics.

System architecture





Algorithm

Core Algorithm: Adaptive weighted adjustment

Goal: Adjust tenant weights every 50 time slices so that:

- Tenants with *low usage* get increased weight
- Tenants with *high usage* get decreased weight
- Long-term weights remain close to the originally intended distribution

Step 1 — Candidate Weight Based on Inverse Usage

For each tenant:

$$candidate_i = \frac{initialWeight_i}{usage_i + \epsilon}$$

Meaning:

- If a tenant executed fewer tasks → candidate becomes larger
- If a tenant executed zero tasks → gets a strong boost
- Guarantees **no starvation**

Step 2 — Normalize Candidate

$$candidate_i \leftarrow \frac{candidate_i}{\sum_j candidate_j}$$

Ensures the vector forms a valid weight distribution.

Step 3 — Smooth Fusion With Initial Weights

$$newWeight_i = (1-\alpha) \cdot initialWeight_i + \alpha \cdot candidate_i$$

Where α = 0.3.

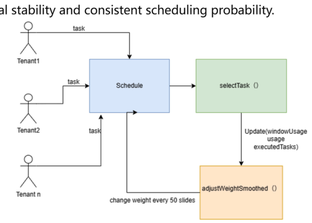
Meaning:

- Prevents violent oscillations
- Keeps long-term behavior aligned with initial intended weights
- Balances stability and responsiveness

Step 4 — Final Normalization

$$weight_i \leftarrow \frac{newWeight_i}{\sum_j newWeight_j}$$

Ensures numerical stability and consistent scheduling probability.



Weighted Random Scheduling

Goal: Select exactly one task per time slice, with the probability of choosing each tenant proportional to its weight.

Algorithm Steps

1. Collect all tenants whose queues are non-empty → `active` list.
2. If only one tenant is active → directly pop a task from that tenant.
3. Compute total weight:

$$W = \sum_{t \in active} weight_t$$

4. Generate a random number

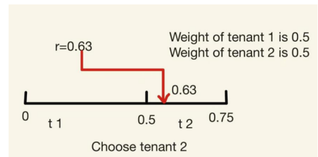
$$r \in [0, W]$$

5. Iterate through active tenants and accumulate weights:

$$acc+ = weight_i$$

6.When acc>=r; choose the tenant :

Example:



Results

The simulation of the adaptive multi-tenant scheduler over 1000 time slices shows that all five tenants received execution opportunities proportional to their intended weights, with final task counts of 181, 121, 362, 118, and 218, respectively. The scheduler maintained 100% CPU utilization, ensured no tenant starved, and dynamically adjusted weights toward stable values close to their original settings (0.208, 0.105, 0.363, 0.138, 0.186). Overall fairness reached a Jain Index of 0.8337, demonstrating that the adaptive weight-smoothing algorithm effectively balanced responsiveness and long-term weighted fairness across tenants.

```
Start simulating: n=5, timeSlices=1000, maxNewTasksPerSlice=3, adjustInterval=50, alpha=0.200, seed=1764144446750
t=50 adjusted weight: T1=0.223 T2=0.112 T3=0.344 T4=0.133 T5=0.189
t=100 adjusted weight: T1=0.190 T2=0.199 T3=0.345 T4=0.092 T5=0.174
t=150 adjusted weight: T1=0.235 T2=0.120 T3=0.344 T4=0.116 T5=0.185
t=200 adjusted weight: T1=0.188 T2=0.141 T3=0.348 T4=0.140 T5=0.184
t=250 adjusted weight: T1=0.198 T2=0.145 T3=0.363 T4=0.103 T5=0.191
t=300 adjusted weight: T1=0.201 T2=0.117 T3=0.367 T4=0.116 T5=0.199
t=350 adjusted weight: T1=0.181 T2=0.098 T3=0.338 T4=0.202 T5=0.181
t=400 adjusted weight: T1=0.197 T2=0.109 T3=0.374 T4=0.126 T5=0.194
t=450 adjusted weight: T1=0.201 T2=0.114 T3=0.352 T4=0.120 T5=0.212
t=500 adjusted weight: T1=0.233 T2=0.103 T3=0.378 T4=0.102 T5=0.185
t=550 adjusted weight: T1=0.180 T2=0.194 T3=0.343 T4=0.108 T5=0.175
t=600 adjusted weight: T1=0.197 T2=0.116 T3=0.379 T4=0.112 T5=0.196
t=650 adjusted weight: T1=0.209 T2=0.106 T3=0.362 T4=0.124 T5=0.199
t=700 adjusted weight: T1=0.210 T2=0.121 T3=0.353 T4=0.114 T5=0.202
t=750 adjusted weight: T1=0.196 T2=0.127 T3=0.342 T4=0.139 T5=0.195
t=800 adjusted weight: T1=0.195 T2=0.107 T3=0.362 T4=0.134 T5=0.202
t=850 adjusted weight: T1=0.208 T2=0.122 T3=0.359 T4=0.118 T5=0.193
t=900 adjusted weight: T1=0.200 T2=0.138 T3=0.366 T4=0.108 T5=0.188
t=950 adjusted weight: T1=0.180 T2=0.108 T3=0.343 T4=0.156 T5=0.213
t=1000 adjusted weight: T1=0.208 T2=0.105 T3=0.363 T4=0.138 T5=0.186
=== result ===
Tenant 1: executed=181, finalWeight=0.208
Tenant 2: executed=121, finalWeight=0.105
Tenant 3: executed=362, finalWeight=0.363
Tenant 4: executed=118, finalWeight=0.138
Tenant 5: executed=218, finalWeight=0.186
Time slices: 1000, Executed tasks: 1000, Idle slices: 0, CPU utilization: 1.000
Jain Fairness: 0.8337
```