Optimizing Server CPU Allocation to Minimize Energy and Maximize Performance

Team: Cloud Calculus (Parikshith, Munjikesh) Members: BT2024249, BT2024247

1. Problem Statement and Motivation

Modern cloud computing environments face a core conflict: they must allocate CPU resources from multiple servers to a diverse set of **users**. Each **user** has their own performance requirements, while the servers have finite capacity and consume significant energy.

Allocating these CPU resources **randomly** or with simple **round-robin** strategies is highly inefficient. It leads to:

(1) **Energy Waste:** Serving **users** on non-optimal servers, increasing operational costs.
(2) **Poor User Experience:** Failing to meet the minimum performance requirement, D_i, for critical **users**, causing lag or failure.
(3) **Network Bottlenecks:** Over-utilizing some servers while others remain idle, leading to poor server utilization.

Our Motivation

This project will design, formulate, and solve an optimization model to find the provably optimal CPU resource allocation plan. The goal is to create a system that intelligently balances energy costs and performance (throughput) while respecting all physical server limits and **user-level** requirements.

2. Planned Methodology

We will investigate this problem in two phases: a fast Linear Program and a more realistic Non-Linear Program.

Phase 1: Linear Programming (LP) Model

First, we formulate a classic LP to find a guaranteed optimal solution for a simplified model.

2.1.1. LP Parameters and Decision Variables

- $\mathbf{i}, \mathbf{j}$**:** Indices for **user** (**1...N**) and server (**1...M**).
- $\mathbf{C_j}$**:** Max CPU capacity of server j.
- $\mathbf{D_i}$**:** Minimum required CPU for **user** i.
- $\mathbf{d_{ij}}$**:** A **cost** factor for **user** i on server j.
- $\mathbf{r_{ij}}$**:** The **processing rate** for **user** i on server j.
- α**:** A constant energy scaling factor.
- $\mathbf{w_1}, \mathbf{w_2}$**:** Weights for the energy and performance objectives.
- $\mathbf{x_{ij}}$**: (Decision Variable)** The CPU resources allocated from server j to **user** i.

2.1.2. LP Objective Function

The objective is to **minimize** the total **net cost** Z, defined as:

$$\min \quad Z = \underbrace{w_1 \sum_{i,j} \alpha d_{ij}^2 x_{ij}}_{\text{Total Energy Cost}} - \underbrace{w_2 \sum_{i,j} r_{ij} x_{ij}}_{\text{Total Performance Benefit}}$$

2.1.3. LP Constraints

(1) **Server Capacity:** $\sum_i x_{ij} \leq C_j \quad \text{(for each server } j\text{)}$
(2) **User Requirement:** $\sum_j x_{ij} \geq D_i \quad \text{(for each user } i\text{)}$
(3) **Non-Negativity:** $x_{ij} \geq 0 \quad \text{(for all } i, j\text{)}$

2.1.4. LP Solver

This classic LP model will be solved using the **PuLP** library in Python. PuLP translates our model and interfaces with an industrial-strength solver that uses the **Simplex Method**. Because our problem is fully linear (a convex problem), the Simplex algorithm is guaranteed to find the single, globally optimal x_{ij} plan that perfectly satisfies all interconnected constraints.

Phase 2: Non-Linear Programming (NLP) Model

To better reflect reality, we will upgrade the model with non-linear equations.

2.2.1. NLP Formulation

For this model, the **parameters, decision variables, and constraints are identical** to the LP model in Phase 1. The **only** change is that the objective function is upgraded to be non-linear to model real-world physics:

- **Non-Linear Energy:** We model energy as quadratic ($E \propto x_{ij}^2$) to penalize high loads.
- **Non-Linear Performance:** We model performance as logarithmic ($T \propto \log(1+x_{ij})$) to capture diminishing returns.

This gives the new non-linear objective $f(x)$:

$$\min \quad Z = f(x) = \sum_{i,j} \left(w_1 \alpha x_{ij}^2 \right) - \sum_{i,j} \left(w_2 \log(1 + r_{ij} x_{ij}) \right)$$

2.2.2. NLP Solver: KKT and Trust-Region

This problem is non-linear and non-convex and cannot be solved with the Simplex method. We will use a two-part methodology:

(1) **The "Target": KKT Conditions** We use the **Karush-Kuhn-Tucker (KKT) conditions** to mathematically define the properties of the optimal solution. This involves creating a **Lagrangian ($\mathcal{L}$)** function which combines the objective with its constraints using **price** variables called Lagrange Multipliers:

- λ_j: The **price** of server j's capacity.
- μ_i: The **price** of user i's requirement.
- ν_{ij}: The **price** of non-negativity.

The KKT conditions (Stationarity, Primal/Dual Feasibility, and Complementary Slackness) provide a system of non-linear equations that must be true at the optimal point. The **Stationarity** condition is our primary target:

$$\frac{\partial \mathcal{L}}{\partial x_{ij}} = \left[2 w_1 \alpha x_{ij} - \frac{w_2 r_{ij}}{1 + r_{ij} x_{ij}} \right] + \lambda_j - \mu_i - \nu_{ij} = 0$$

(2) **The "Engine": Trust-Region Method** We will implement a **Trust-Region algorithm** to numerically solve the KKT system. This is a highly stable iterative algorithm. At each step, it builds a simple quadratic **model** (m_k) of the objective within a small **trust region** (Δ_k), solves this subproblem to find a step, and then checks the step's accuracy on the real function. This **model-solve-check-update** loop continues until the KKT conditions are satisfied.