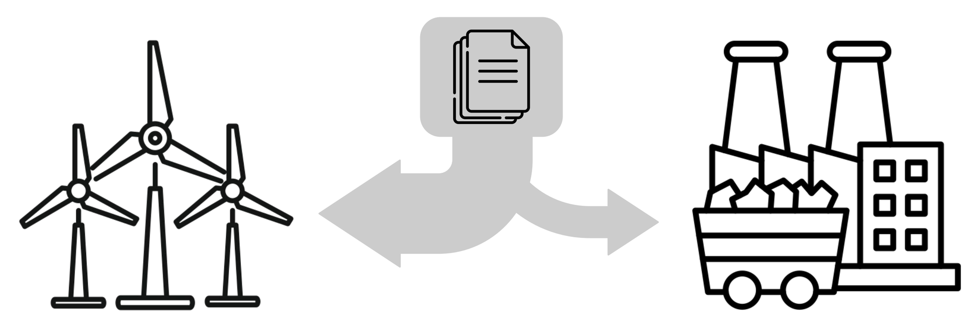
GreenBalance: Carbon-Aware Load Balancing

Himnish Chhabra & Nishil Adina | Mentor: Thanathorn (Tammy) Sukprasert

Introduction

- **Data centers use lots of electricity**, and the carbon footprint depends on the local power grid.
- **Our goal:** Cut down emissions by sending web traffic to regions running on cleaner energy.
- **Solution:** A **carbon-aware load balancer** that automatically shifts requests toward lower-carbon servers.
- **Key idea:** If Region A's grid is cleaner than Region B's right now, A should receive more requests — *all else equal*.



Key Tools & Data

Load Balancer

- We use **HAProxy**, a popular industry tool.
- It allows us to adjust **server weights on the fly** (no restart needed).
- It also has a **built-in dashboard** to watch traffic distribution.

Dataplane API

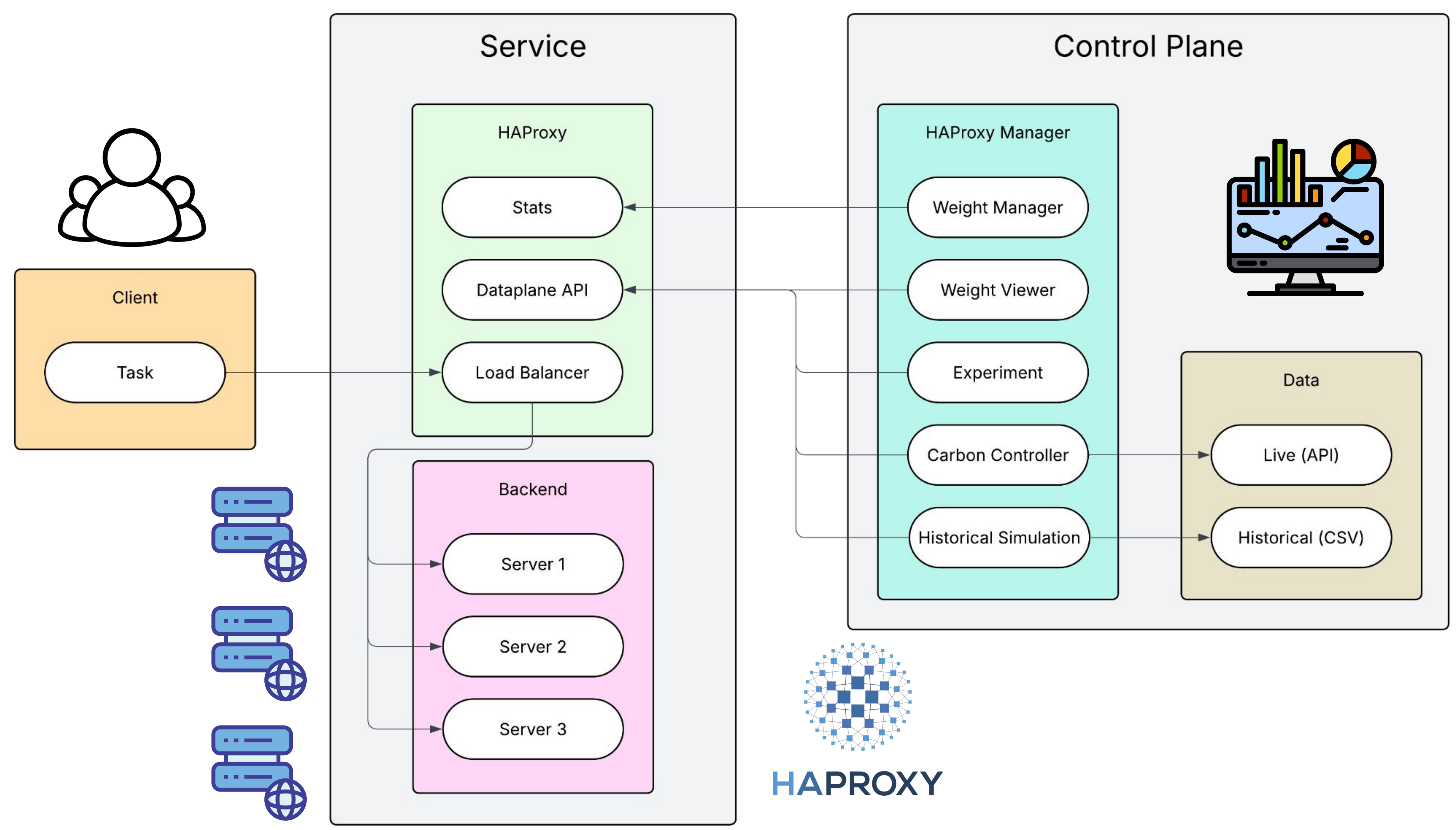
- We look at the **carbon intensity** (how clean the energy is) in each region.
- Cleaner regions get **higher weights** (more traffic), dirtier ones get **lower weights**.
- We update these weights via Dataplane API **every ~5 minutes**.

WattTime

Data Sources

- **Real-time data:** From **WattTime Data API** (e.g., California).
- **Historical data:** 26k+ hourly records (2020–2022) for California, Texas and the Mid-Atlantic (CSV Files)
- **Simulation:** For regions without live data, we model realistic patterns.

Architecture



Evaluation

Key:

N : total requests, S : servers, r_i : requests to server i

c_i : carbon intensity ($\mathrm{gCO_2/kWh}$), w_i : carbon-aware weight

cost_i : cost/request, l_i : latency (ms)

Formulae:

Round-robin: $r_i = \lfloor N/S \rfloor + (N \bmod S)$

Carbon-aware weight: $w_i = \dfrac{1/c_i}{\sum_j 1/c_j}$

Requests: $r_i \approx \mathrm{round}(N \cdot w_i)$, $r_i \le \mathrm{capacity}_i$

Total carbon: $\sum_i r_i c_i$, Total cost: $\sum_i r_i \mathrm{cost}_i$

Avg latency: $(\sum_i r_i l_i)/N$, Savings: (Baseline − Aware)/Baseline

Example Input

- **Servers:**
  - S1: Capacity=50, Carbon=100, Cost=2, Latency=1
  - S2: Capacity=49, Carbon=20, Cost=2, Latency=1
  - S3: Capacity=50, Carbon=20, Cost=2, Latency=1
- **Total Requests (N):** 10

Example Output

- **Round-robin** ($r_i = \lfloor N/S \rfloor + (N \bmod S)$): Requests=[4,3,3], Carbon=520, Cost=20, Latency=1
- **Carbon-aware** ($w_i = \frac{1/c_i}{\sum_j 1/c_j}$, $r_i \approx \mathrm{round}(N \cdot w_i)$): Requests=[1,5,4], Carbon=300, Cost=20, Latency=1
- **Savings:** Carbon ≈ 42%, Cost/Latency unchanged

"Cumulative CO_2 savings steadily increase over time, showing the carbon-aware policy reduces emissions compared to round-robin scheduling." (Graph from CSV data in software)

Software & Next Steps

- **More servers:** Add more regions and multiple servers per region.
- **Smarter scheduling:** Go beyond weighted round-robin.
  - Predictive routing (use weather forecasts).
  - Balance cost *and* carbon.
  - Keep latency/reliability in mind.
- **Integrations:** With Kubernetes, CDNs, and edge devices.
- **Carbon reports:** Show savings in numbers for sustainability tracking.

Software

1. Download codebase
2. Add a .env file with WaTTime credentials
3. Run command "docker compose up"
4. Available on localhost:5000

Conclusion

The Green Load Balancer shows that small changes in how we route web traffic can make a big difference for the planet. In our evaluation, carbon-aware routing cut emissions by **42%** compared to round-robin, without increasing cost or latency. This approach is practical today and opens the door to smarter, sustainable cloud services tomorrow.