**📜 Code Explanation (Super Detailed)**

**1. Class: WeightedRoundRobinLoadBalancer**

**python**

**CopyEdit**

**class WeightedRoundRobinLoadBalancer:**

**"""Load Balancer that uses Weighted Round Robin strategy."""**

* **A class is created to model the behavior of a Load Balancer.**
* **This Load Balancer uses Weighted Round Robin (WRR) to distribute incoming client requests across multiple servers.**

**2. Initializer: \_\_init\_\_(self, servers)**

**python**

**CopyEdit**

**def \_\_init\_\_(self, servers):**

**self.servers = servers**

**self.lock = threading.Lock()**

**self.total\_requests = 0**

* **servers: List of Server objects passed while creating Load Balancer.**
* **self.lock: A thread Lock object to ensure that only one thread at a time can modify shared data like picking a server.  
  👉 *Needed because client requests are coming from multiple threads (parallel).***
* **self.total\_requests: Keeps count of total requests handled by the load balancer (incremented every time a request comes).**

**3. Weighted Server List Building**

**python**

**CopyEdit**

**self.weighted\_servers = []**

**for server in servers:**

**self.weighted\_servers.extend([server] \* server.weight)**

* **self.weighted\_servers: A special list built from servers based on their weight.**
* **For each server, we add multiple copies into the list according to their weight.**

**For example:**

* + **S1 (weight 2) → [S1, S1]**
  + **S2 (weight 1) → [S2]**
  + **S3 (weight 3) → [S3, S3, S3]**
* **So the full weighted list becomes:**

**python**

**CopyEdit**

**[S1, S1, S2, S3, S3, S3]**

**⚡ *This list is the core idea behind WRR in this code.***

**4. Method: distribute\_request(self, request)**

**python**

**CopyEdit**

**def distribute\_request(self, request):**

**"""Assign a request to a server based on Weighted Round Robin."""**

**with self.lock:**

**server = random.choice(self.weighted\_servers)**

* **Purpose: This method assigns one client request to one server.**
* **with self.lock: ensures that only one thread picks a server at a time — no conflict.**
* **Important:  
  random.choice(self.weighted\_servers) randomly picks one server from the weighted list.**
  + **S1 appears twice, so has 2/6 probability.**
  + **S2 appears once, so has 1/6 probability.**
  + **S3 appears three times, so has 3/6 probability.**

**💡 *Thus, servers with higher weight have higher chances of being selected.***

**Continuing in distribute\_request**

**python**

**CopyEdit**

**# Process request in a new thread**

**threading.Thread(target=server.handle\_request, args=(request,)).start()**

**self.total\_requests += 1**

* **A new thread is created to simulate the server handling the client request without blocking others.**
* **The server's handle\_request() method is called in a new thread.**
* **self.total\_requests is incremented to track total client requests handled so far.**

**📦 Next Part: simulate\_client\_requests**

**python**

**CopyEdit**

**def simulate\_client\_requests(load\_balancer, num\_requests):**

**"""Generates and distributes client requests."""**

**for i in range(num\_requests):**

**request = ClientRequest(i+1)**

**threading.Thread(target=load\_balancer.distribute\_request, args=(request,)).start()**

**time.sleep(random.uniform(0.5, 1.5)) # Random inter-arrival time**

* **This simulates multiple clients sending requests to the Load Balancer.**
* **For each request:**
  + **Create a new ClientRequest object with a unique ID.**
  + **Call load\_balancer.distribute\_request(request) in a new thread (again, real-world parallelism).**
* **After sending each request, wait randomly between 0.5s to 1.5s to simulate natural client traffic (not all at once).**

**🏗 Server Creation and Load Balancer Setup**

**python**

**CopyEdit**

**server\_list = [Server("S1", weight=2), Server("S2", weight=1), Server("S3", weight=3)]**

**load\_balancer = WeightedRoundRobinLoadBalancer(server\_list)**

* **Create three Server objects:**
  + **S1 → weight 2**
  + **S2 → weight 1**
  + **S3 → weight 3**
* **Pass them into WeightedRoundRobinLoadBalancer, which will build the weighted\_servers list.**

**🚀 Running the Simulation**

**python**

**CopyEdit**

**simulate\_client\_requests(load\_balancer, 10)**

**time.sleep(10)**

* **Simulate 10 client requests.**
* **Sleep for 10 seconds to give enough time for all threads (requests) to complete.**

**📊 Printing Load Balancing Summary**

**python**

**CopyEdit**

**print("\n📊 Load Balancing Summary:")**

**print(f"🔹 Total Requests Processed: {load\_balancer.total\_requests}")**

**for server in server\_list:**

**print(f"✅ {server.name}: {server.total\_requests\_handled} requests handled")**

* **After simulation ends:**
  + **Print total requests processed.**
  + **Print how many requests each server has handled.**

**📌 In Short:**

| **Step** | **Action** |
| --- | --- |
| **Initialize Servers** | **With weight** |
| **Build Weighted List** | **Multiple copies according to weight** |
| **Accept Requests** | **Distribute randomly from weighted list** |
| **Process in Threads** | **Servers handle them in parallel** |
| **Print Results** | **How many requests each server served** |

**⚡ Key Point: Why use random.choice?**

**You chose random.choice because it:**

* **Quickly models the probability-based weighted distribution.**
* **Easy and fast for simulations.**
* **However, it is not 100% accurate for small numbers (small sample randomness).**