Load Balancing Implementation Report

1. Basic Assumptions

- **Server Registration and Discovery:**
  - Assumption: Workers register with the etcd service upon startup, providing their ID and address. The LB server retrieves this information from etcd to maintain a list of available workers.
  - Implementation:
    - Workers use the `go-etcd/etcd` client to connect to the etcd service.
    - Upon startup, each worker creates a lease with a TTL (Time-To-Live) and registers its information (ID and address) in etcd with the lease attached.
    - The LB server also uses the `go-etcd/etcd` client to watch for changes to worker registrations under the `/services/workers/` prefix.
    - The LB server periodically retrieves the list of available workers and their information from etcd.
- **Server Load Reporting:**
  - Assumption: Workers periodically report their CPU utilization to the LB server via heartbeat messages.
  - Implementation:
    - Workers use the `gopsutil` library to obtain their current CPU utilization.
    - The LB server sends heartbeat requests to each registered worker periodically.
    - The `updateWorkers` function in the LB server retrieves worker information from etcd and sends Heartbeat requests to the worker.
- **Client-LB Interaction:**
  - Assumption: Clients always query the LB server via gRPC to determine which worker server to use for a task.
  - Implementation:
    - A `GetWorker` gRPC service is defined on the LB server, which clients call to request a worker.
    - The client provides a unique Client ID in the `WorkerRequest`.
    - The LB server responds with the assigned worker's ID and address.
- **Communication Protocol:**
  - Assumption: gRPC is used for all communication between clients, the LB server, and backend worker servers.
  - Implementation:
    - gRPC services and message types are defined using protobuf in the `proto` directory (`lb.proto`, `worker.proto`, and `heartbeat.proto`).
    - gRPC is used for:
      - Client requests to the LB server for worker assignment (`GetWorker`).
      - LB server heartbeat requests to workers (`SendHeartbeat`).
      - Worker responses to LB server heartbeat requests (`SendHeartbeatResponse`).
      - Client requests to worker servers for task execution (`DoWork`).

2. Load Balancing Policies Implementation

- **Pick First:**
  - Implementation:
    - If the scheduling type is set to "first", the LB server selects the first worker from the list of available workers.
    - The `getNextWorker` function in the LB server retrieves the first worker from the list.
- **Round Robin:**
  - Implementation:
    - If the scheduling type is set to "rr", the LB server maintains a `last_assigned_worker` index.
    - The `getNextWorker` function increments this index (modulo the number of available workers) and selects the worker at the resulting index.
- **Least Load:**
  - Implementation:
    - If the scheduling type is set to "ll", the LB server iterates through the list of available workers and compares their reported CPU utilization (load).
    - The `getNextWorker` function selects the worker with the lowest reported load.
    - If no load is reported by a worker, it is treated as the lowest load.

3. Testing and Results

- To test the load balancing policies, multiple worker servers and client requests were simulated.
- The number of workers was varied (e.g., 3 workers).
- Clients sent requests with varying task types ("sleep" and "sum-up") to simulate different workloads.
- The load balancing policy was switched using the `-type` flag when starting the LB server.
- Observations:
  - **Pick First:** In basic tests, "Pick First" sends all requests to the first available server. This can lead to uneven load distribution, especially when the first server is slow.
  - **Round Robin:** "Round Robin" distributes requests more evenly across available servers. This helps to prevent overloading any single server.
  - **Least Load:** "Least Load" attempts to distribute requests based on the reported CPU utilization of the servers. In tests, this policy generally directs traffic to less loaded servers, but its effectiveness depends on the accuracy and frequency of load reporting. If the load reporting mechanism isn't very precise or frequent, "Least Load" might not perform significantly better than "Round Robin."

  Overall, the round-robin policy is the best choice as the `gopsutil` library used for CPU utilization reporting has a significant delay in reporting the CPU utilization. The CPU utilization needs to ramp up slowly to get the correct value, but our implementation adds all the clients at once, meaning the CPU utilization is not accurate. This gives very similar though a little better performance than the pick first policy.
- Further testing with more sophisticated load simulation and metrics collection (response time, throughput) would provide a more in-depth analysis.

4. Sample Run Output

LL Total Requests: 150 Total Execution Time: 126.42 sec System Throughput: 1.19 requests/sec

RR Total Requests: 150 Total Execution Time: 125.09 sec System Throughput: 1.20 requests/sec

PF Total Requests: 150 Total Execution Time: 142.42 sec System Throughput: 1.05 requests/sec