Vimeo Programming Assignment

Language Used: Golang
Tools Used: Postman Client

Attachments: Binary of load balancer (load-balancer), source code and report.

Installation: Attaching a go binary (load-balancer) to execute the service.
Command:- **./load-balancer nodes=http://localhost:9000, http://localhost:9001,http://localhost:9002,http://localhost:9003,http://localhost:9004 -p 8000**

-node= is taking registering server in ',' separated form.
-p is the port number on which load balancer service is running

Test: localhost:8000 (GET)

Response:

- **Success: {val: id}**
- **Failure:**
    - **503, Something went wrong!**
    - **No server found after multiple retries**

Assumptions:

1. Waiting for up to 2 seconds (OR 20 Retries after every 100 ms) to get the response in case "selected server is down". If no live server found, then will return "503 Error".
2. Edge case: If there is a live server and any request comes, If in between these two operations, server goes down the it will repeat the process of step1.

Entities & Approach:

- instanceList: store the list of registered servers with the load balancer
- nodeStatus: Thread safe (mutex lock) Map to store only live servers.
- During service invocation, will call the member function of every node and keep pooling it in interval of 2 seconds.
    a. If server is up, then add it to the nodeStatus Map otherwise remove it from the map.
    b. Instantiate nodeStatus map only once using "sync" package of Go and this newly created object will act as the instance of singleton class nodeStatus.
- Every registered server is being polled in interval of 2 second in their own separate go routine.
- When no live server found, it will return "503 Error"

- If there is race condition where server is up but just before the request, it goes down. In this case with retry on some other server (maximum 20 retries)

Note:

- If number of registered servers are more, the system will be more reliable because the probability of going down all server at the same time is very low.

Performance/Bottleneck:

- This load-balancer is not handling the case of request drop offs where if workload is high then reverse proxy will push the requests in a queue for some time and regain the request and its response.
- No. of successful requests served is directly proportional to No. of Registered Servers.
- Not handling the rate limiter on multiple requests coming from same resource (generally load balancers have some rate limiter layer on top of it)
- One bottleneck could be load-balancer service itself, if it goes down then whole system goes down.
- One solution to above problem is to run it in clustered mode and add ELB in front of it.
- Ping generally takes very minimal resources but it's a waste of time and resources for the load-balancer. Ideally there should be some mobile client running on Darwin server itself and in case of any event (going live or down) it should send callback to load balancer (one mobile client in server mode is running on load balancer).
- Round robin algorithm is taking into consideration the resource and its consumption of registered servers. Ideally the bigger node should receive more requests as compared to smaller (lightweight) node.
- Logic implemented for request allocation in round robin here is not considering the data store, access control and types of services. Because some endpoints have separation of concern based on the request type and access token coming in the request (like in Access control list, we might have higher load for some specific users).