Question 1 Report

Kshitij Nigam
2023202031

a) Broad Overview of the System

The system designed is a distributed client-server architecture using gRPC for remote procedure calls (RPC), implementing load balancing across multiple backend servers. The central component of the system is a Load Balancer (LB) server, which is responsible for distributing incoming client requests to the most suitable backend servers. The LB uses a Look-Aside load balancing approach, meaning that clients interact with the LB server to get the address of the backend server where they should send their requests.

The system implements three load balancing strategies:

1. **Pick First** – Selects the first available server from the list.
2. **Round Robin** – Distributes requests cyclically among available servers.
3. **Least Load** – Routes requests to the server with the least load.

Each backend server can handle computational tasks such as simple arithmetic operations (addition, multiplication) and CPU-intensive tasks (e.g., solving the Tower of Hanoi problem). The system also simulates load on the backend servers, allowing for dynamic performance measurement. To achieve distributed server management, the system uses **etcd** for server discovery, registration, and load reporting.

The client queries the LB for the most suitable backend server based on the selected load balancing policy, sends computational tasks to the backend, and collects results. The system has been tested to simulate a high request load, evaluating the effectiveness of each load balancing policy in terms of response time, server utilization, and overall throughput.

b) Design Choices for Each Requirement

- **gRPC Framework**: The decision to use gRPC as the communication protocol allows for efficient, strongly-typed, and scalable communication between clients, the load balancer, and backend servers. gRPC's use of HTTP/2 ensures multiplexing, header compression, and other optimizations for efficient communication in distributed systems. Furthermore, gRPC’s native support for Protocol Buffers allows for precise and compact message encoding.
- **Look-Aside Load Balancing**: The choice of Look-Aside load balancing was made because it consolidates the load balancing logic in the LB server, enabling easier scalability and flexibility. Clients query the LB server, which can maintain complex load balancing algorithms. This architecture decouples the client and server and provides a clear separation of concerns.
- **Load Balancing Policies**:

    - **Pick First**: This strategy is ideal when a client needs to select the first available server, providing simplicity and low overhead. It is effective in environments where backend servers do not change their availability frequently.
    - **Round Robin**: This policy is suited for situations where backend servers have similar computational capacity and should receive an equal share of requests. It distributes the load evenly, ensuring no server is overwhelmed.
    - **Least Load**: This policy dynamically routes requests to the backend server with the least load. It ensures that the servers with more available resources handle additional tasks, optimizing resource utilization and improving system performance.
- **Backend Servers**: Backend servers are responsible for processing computational requests from the client. They periodically report their status (load and availability) to the LB server. This information is crucial for the load balancing decisions and ensures that clients are directed to the most suitable backend.
- **etcd for Service Discovery**: The use of etcd for service discovery ensures that the LB server is always aware of the registered backend servers, their availability, and current load. etcd also supports leases and provides mechanisms for server registration, keeping the system resilient to failures and providing the ability to dynamically add or remove backend servers.
- **Concurrency and Scalability**: The system supports multiple concurrent clients through goroutines, allowing for the simulation of large-scale loads and providing insights into the system's scalability under high demand. The scale testing includes measuring response time, throughput, and evaluating the load balancing policies' efficiency.

c) Assumptions and Future Work

- **Assumptions**:
    - The backend servers can be assumed to have sufficient computational power to handle the requests as simulated in the system. In practice, actual server capabilities may vary.
    - The network latency between the client, LB server, and backend servers is minimal and does not significantly impact the overall performance of the system. In real-world deployments, network latency could be a factor to consider.
    - The backend servers register and report their load periodically to the LB, and the LB server is always aware of the current status of all backend servers.
    - For simplicity, the system uses a fixed TTL (Time-To-Live) for backend server leases in etcd (10 seconds). In a real system, lease times and the frequency of load reporting could be more flexible.
- **Future Work**:
    - **Extended Load Balancing Policies**: While the current system implements basic load balancing policies, future work could explore more sophisticated

algorithms like weighted least connections or dynamic load balancing based on server metrics beyond just load (e.g., response time, CPU usage, etc.).

- **Fault Tolerance**: The system assumes that backend servers are always responsive when queried. Future improvements could include handling server failures more gracefully, such as automatically retrying requests to alternative servers or implementing circuit breakers.
- **Security**: The system currently does not incorporate security features such as authentication or encryption. Future work could implement these features to ensure secure communication between clients, the load balancer, and backend servers.
- **Optimization and Profiling**: The system could benefit from optimizations and profiling tools to better understand performance bottlenecks, especially when handling large numbers of concurrent requests.

d) Other Considerations

- **Testing and Validation**: The system was subjected to extensive scale testing using the scale_test.sh script. The testing validated the correctness of the load balancing strategies, measured system performance under various loads, and ensured that the LB server could distribute requests effectively.
- **Performance Analysis**: During scale testing, the system demonstrated that different load balancing strategies impacted overall performance in distinct ways. For instance, while Round Robin provided an even distribution of requests across servers, the Least Load strategy resulted in better overall system responsiveness by directing requests to less busy servers. Pick First, although simple, had slightly higher latency under heavy load, as it did not dynamically adapt to server load.