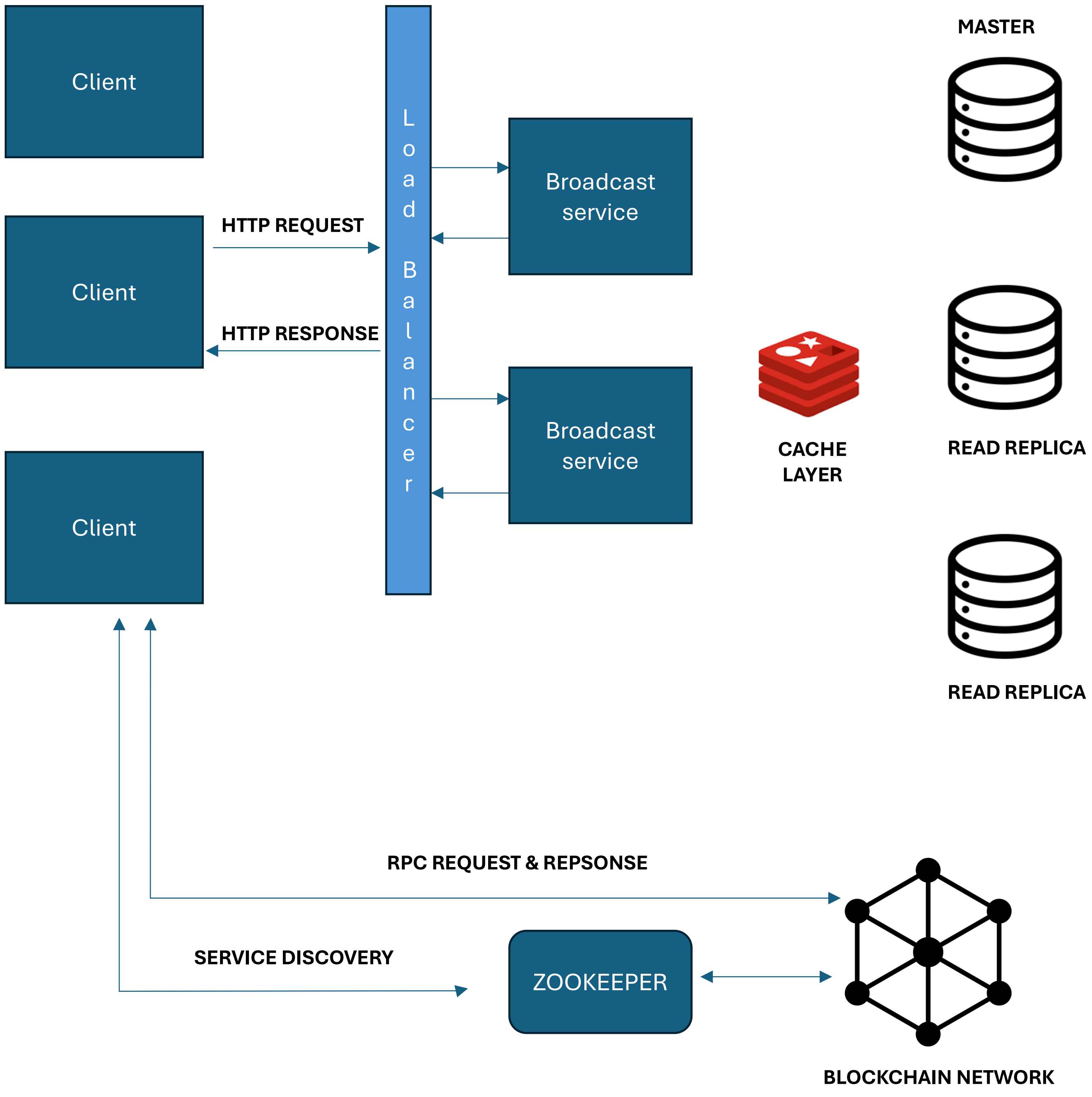
## 1. System architecture diagram



## 2. System components design

### i. Client

- The client component can be a web/mobile UI. Through the client both the admin and normal users can access the service.

- The client can access the broadcast service using the API (transaction/broadcast) endpoint provided.

- The client can broadcast the signed transaction to a node through a zookeeper instances that maintains information about the nodes in the blockchain.

- A back-off mechanism is implemented within the client to retry in case of failure. Back-off mechanisms like exponential backoff can be used to reduce unnecessary load.

### ii. Broadcast service

- This service returns the signed data

- Multiple instances of the services is running to ensure there is no single point of failure.

### iii. Load balancer

- A load balancer redistributes the load to the multiple broadcast services that are available.

- Load balancing techniques like round-robin and hashing can be used to distributed the incoming request.

### iv. Service discovery

- To contact a node, the IP address of the node should be known. However, in a blockchain with a network of nodes, given its dynamic nature, a dedicated service is necessary to obtain the IP address of the node,

- Therefore, services like zookeeper is used. Using the zookeeper we can maintain and retrieve the IP address of a node in the network

v. **<u>Blockchain node</u>**

- The client contacts the blockchain node using Remote procedural call (RPC).

- A response code will be returned to indicate the result of the signed broadcast throughout the network.

vi. **<u>Database</u>**

- The database is used to store information such as the status of each transaction, user information, and other necessary data required for the application's operation.

- To prevent a single point of failure, reduce load and scale, the database is sharded.

- Utilizing separate databases for read and write operations is a common strategy to optimize performance and maintain consistency. Read databases (replicas) can be distributed geographically to reduce latency for read-heavy workloads.

- Adjusting the ratio of read and write replicas depends on the nature of the application's workload. If the application is read-heavy, more read replicas can be created to handle the load, improving read performance. If the application is write-heavy, write replicas might be scaled or optimized accordingly.

vii. **<u>Cache</u>**

- To reduce database reads, caching layer can be used.

- Depending on the consistency requirements, caching update techniques like write-through, write-around, write-back can be used.