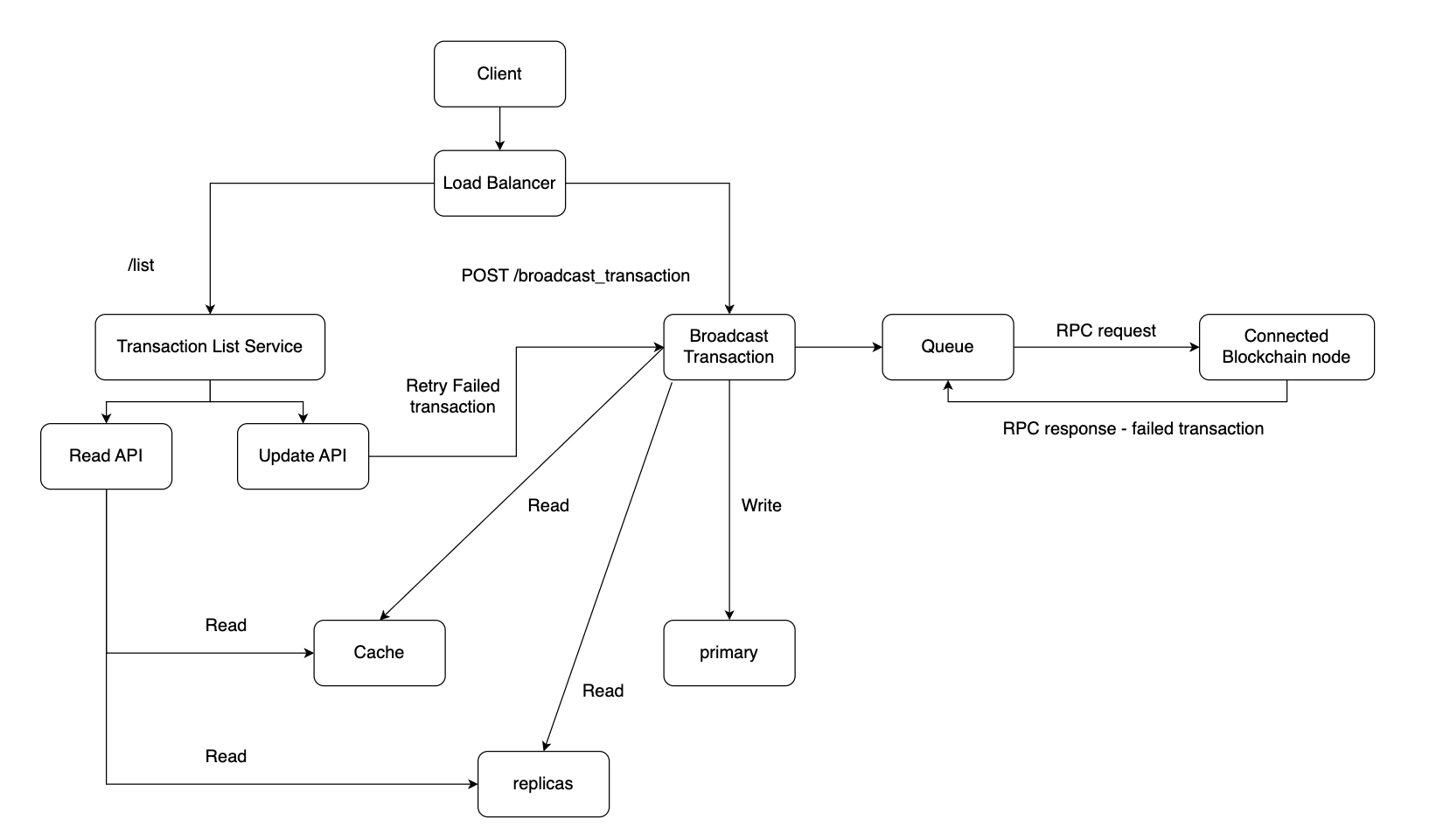
Flow Chart



Functional requirements - API calls

* Broadcast transactions to nodes of blockchain network
  + POST /broadcast\_transaction

|  |  |
| --- | --- |
| Input Field | Example |
| message\_type | add\_weight(address \_addr, uint256 \_weight) |
| data | 0xd71363280000000000000000000000005eb715d601c2f27f83cb554b6b36e047822fb70a00000000000000000000000000000000000000000000000000000000000000fa |

* Display a list of transactions
  + GET /list\_transaction
* An admin can, at any point in time, retry a failed transaction.
  + In viewing the list of transactions, an admin can invoke the post request to retry the failed transaction with the existing information

Performance requirements

* If the broadcaster service restarts unexpectedly, it should still fulfill them.
  + The service will persist data in database with replicas
  + Once the service restarts, it will retrieve the transactions that have HTTP 200 status but haven't been broadcasted.
* A broadcasted transaction might fail and if it fails, it should be retried automatically.
  + Since the transaction data has been stored in cache and database, retry will get the data from cache, which is quick.
  + The service will manage a queue to temporarily order transactions that need to be broadcasted.
    - Whenever a failed transaction needs to be retried automatically or an admin wants to retry a failed one or a new transaction is initiated, they will all be pushed into this queue.
* RPC request
  + Wait for a response for at most 30 seconds until receive a transaction; If there is still no response within 30 seconds, resend the transaction. This will happen only 5% of the time, which is acceptable.

Data Schema

|  |  |
| --- | --- |
| Transaction |  |
| **Transaction\_id** | String |
| Message\_type | String |
| Data | String |
| Broadcast\_status | Integer( 0 – failure, 1 – success ) |
| Created\_time | timestamp |

Data Layer Design

* Every time we store transaction information in cache first and then in the primary data node. Using cache will boost the efficiency to retrieve info for an instant retry when a transaction fail.
* Even if the read operation from the data cluster is not often, there is still need for replicas since these transaction information is important so we would like more backup in case of primary node failure.
* There will be replica sync delay/inconsistency. But this is not a big problem. Since the data is written to the cache first, the data is also read from cache first. The data replica is only for safety and persistent use.