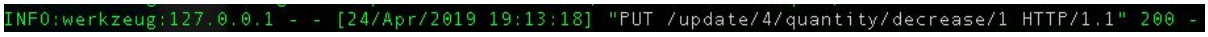
Catalog Server:

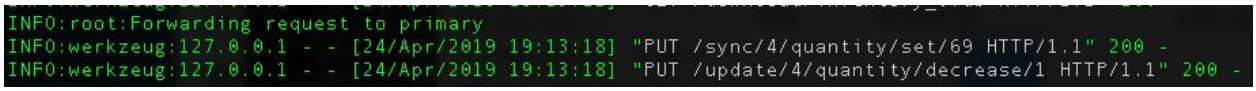
Catalog server uses primary remote-write protocol to ensure sequential consistency. Each server is assigned a integer ID. When a server first comes alive, it goes through the list of servers specified in the config file and takes the server with the highest ID as its primary whether or not it is in fact running. It then goes through the same list in attempt to sync up with any running server. When a server responds to its sync up request, it replaces its own primary server to the same as responding server. When a server detects that the primary server is down which will happen when a forward write request happens, it holds an election that uses bully algorithm to determine the election outcome.

This mechanism doesn't guarantee the server that has the highest ID in the network is the primary but it ensures only one primary exists at any given time and the whole network knows about the primary.



When a write request is received, the server first check if it's the primary. If it is, it simply fulfill the request and calls a sync_all function to update all other non primary servers. Therefore, a write request to a primary server will look like below figure.

The non primary server will get two PUT http requests. One if the original write request called update from the client and the other one is a write request called sync from the primary server.



They don't show up in consistent order because the async nature of the requests

Order server:
Order server also uses primary remote-write protocol to ensure sequential consistency. Any writes to the database from the non-primary replica must be forwarded to the primary, and writing on the primary replica database is locked until the replica is updated with the write.

While both replicas are running, replica 0 is assigned the primary. If a server crash occurs, the surviving replica is assigned the primary. As soon as the second replica comes back online, a broadcast to both replicas occurs where the primary is reassigned to replica 0 (since it must be online).

Load balancing is also implemented if a catalog server has crashed. It follows the same protocol as the frontend server, where it randomly picks a server from the config file to send the request. When the server does not respond, it goes through the same process again in hope that a different server is picked and it's running.

Frontend:

For load balancing, frontend randomly picks a server from the config file to send the request. When the server does not respond, it goes through the same process again in hope that a different server is picked and it's running.

When frontend gets a read request from a client, it first looks into the cache before it makes request to backend servers. Since the client can only issue read requests for topic and item number, which are all unique in the database. The cache is simply a dictionary that use those as the keys and Json response as values.

Improvements and possible extensions:
The order server could be improved by not using hardcoded primary replica protocol. Furthermore, although the random load balancing is effective in expectation, it could be improved by discarding crashed server from the random selection pool.