Fail-Silent Replicated Token Manager with Atomic Semantics

You extend your client-server token manager from the last project to support replication and atomic semantics.

Each token is to be implemented as a (1,N) register with atomic semantics (see slides 38-45 in Distributed Systems Abstractions) in a fail-silent system model.

Your system should consist of a set of nodes that extend the server API from project 2. Your solution should be developed in Go and utilize the **gRPC** and Google **Protocol Buffer** frameworks.

Extend each token to include the access points (IP address and port number) of its single writer and multiple reader nodes; these nodes constitute the replication scheme of the token.

For simplicity, you may assume that the replication schemes of all tokens are static and known apriori to all the server and client nodes in your system. Server nodes create tokens when they start. Your implementation should use a simple YAML file with the replication scheme of all the tokens, i.e. an array of

token: <id>
writer: <access-point>
readers: array of <access-point>s

where <access-point> is of the form <ip-address>:<port>, whereas a writer may also be a reader.

A client sends all its write requests for a token to the token's writer node, while it sends its read requests to any one of the token's reader nodes (read requests for a token by the same client may be sent to different reader nodes, eg. chosen at random). The single server node contacted is the one responding to a client's request, and any client can read/write any token.

Extend the server API with additional RPC calls so that your system supports **atomic semantics**:

for any given token, whenever a client read returns a value v and a subsequent (by any) client read returns a value w, then the write of w does not precede the write of v

To this end, you may need to maintain additional state information about each token, and implement the **read-impose-write-all** (quorum) protocol.

Emulate **fail-silent** behavior for the server nodes as follows. A server node may "crash-stop" for operations on a particular token X (for which it is a reader or writer) at some time t by indefinitely postponing a response to all operations on X it receives after time t. A server may still be responding as usual to requests on other tokens for which it is a reader or writer.

What to submit:
Submit a .tar.gz archive with your

- complete Go code (and any supporting files)
- a bash script demonstrating running your server and sequence of client executions.
- README file (with relevant documentation and usage guide)

References

- Quickstart for the gRPC Framework
- Tutorial on Google Protocol Buffers
- gRPC with ProtoBuf Basics Tutorial
- Goroutines and gRPC Serve() method of gRPC Server
- sync package: Mutexes in Go or Mutex Tour or RWMutex or Map
- Command Line Flags in Go
- YAML configurations for Go
- *Introduction to Reliable and Secure Distributed Programming* by C. Cachin, L.Rodrigues, and R.Guerraoui (2011), sections 4.1.-4.3, pages 137-159.