Distributed Key Value Store using RAFT
Specification document

A. Abdulwahed, M. Kaoula, M. Laruina, R. Mancini

February 1, 2021

1 Introduction

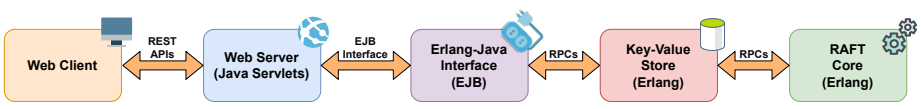
The goal of the project is to implement a distributed and strongly consistent in-memory key-value store using the RAFT algorithm for consensus. The architecture is highly modular, leading to better flexibility and re-usability. With this choice, synchronization and coordination issues are offloaded to the communication between the modules.

The possible operations on the distributed KV store will be the following:

- `V get(K key)`: returns the value associated with the key
- `Map<K,V> getAll()`: returns all key-value pairs
- `void set(K key, V value)`: sets the given value to the provided key
- `void delete(K key)`: deletes the item with the given key
- `void deleteAll()`: deletes all the keys

The core of the RAFT algorithm will be developed using Erlang and will be generic (section 2.1). Using this generic RAFT implementation, another Erlang RPC server will be developed which implements the Key-Value store (section 2.2) and provides an interface to access the datastore. Another module will make it possible to access the key-value store from Java, using the *Jinterface* package (section 2.3). This exported interface will be used by a web-server which will provide REST APIs for the key-value store and an administration GUI (section 2.4).

2 Architecture



2.1 RAFT core

This Erlang module will implement the RAFT finite-state machine (FSM) and will take care of all the communication between nodes. Communication between nodes needs to be reliable and will be taken care of by the Erlang middleware.

This module will be implemented using the *gen_statem* and *gen_server* Erlang *behaviours*. This module will run on all nodes.

2.2 KV store

This Erlang module will implement the Key-Value store. It will need to handle incoming commits from the RAFT core and reply to get and set requests, querying the current master node[1]. All communication with the Erlang core and the *Erlang-Java interface* will be done using RPCs.

This module will be implemented using the *gen_server* Erlang *behaviour*. This module will run on all nodes.

2.3 Erlang-Java interface

This Java module will expose the Erlang KV store to a JEE environment and will be implemented as an EJB. This EJB will provide an interface to view and edit the key-value store and will communicate with the Erlang environment through RPCs using the *Jinterface* package. Communication with other EJB will be handled by the JEE runtime.

2.4 Web Server

This Java module will implement a Web server to server REST APIs and an administrative GUI. It will be implemented using Java Servlets.

[1] *get* requests can be handled directly by a secondary node if a strong consistency is not required