Distributed Systems Project Proposal

Fusheng Yuan (fyuan)

Yi Li (yili1)

Overview

Kiku, a lightweight mobile application that helps friends to come up with a quick decision in an interesting way, consists of a simple user interface on iOS platform and a distributed backend system based on Paxos algorithm.

Has this ever happened to you in daily life? You and your friends ordered pizza when you are doing a group project late in the evening. The pizza comes, but everyone is so involved in coding that no one wants to leave his or her computer to pick up the pizza.

At that time, you could open Kiku on your phone and start passing the timed “bomb”. By dragging the “bomb” to your friend’s name displayed on your screen, you can pass the “bomb” to another person before it explodes. Be sure to do it quick because the “bomb” explodes in 10 seconds. To add fun and surprises, the countdown is not displayed on the screen and the “bomb” does not appear on your phone unless it is passed to you. If the “bomb” goes off on your phone, then you are the one to pick up the pizza.

To balance the load, different users are partitioned on different servers. Thus servers need to keep the replicated state consistent, i.e. information about which user currently has the bomb needs to be the same across all servers.

To achieve the above consensus and tolerate asynchronous failures, we use Paxos algorithm to implement the backend system using Go language.

System Architecture

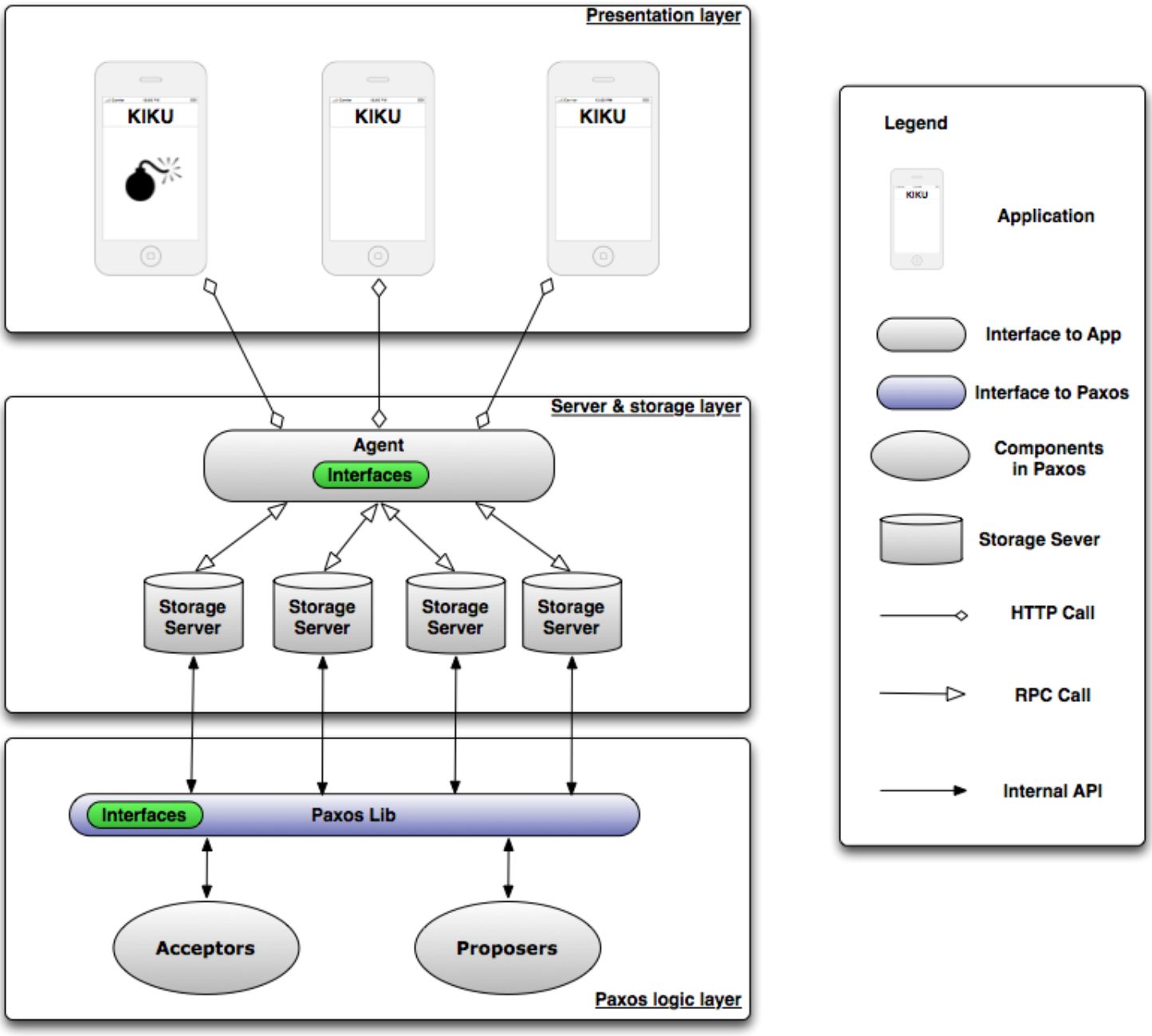

Figure 1. Architecture for Kiku system

We decide to divide the system into three layers: Presentation layer, server & storage layer and paxos logic layer.

Presentation Layer

In this layer, each Kiku application would send the requests as a client to the server & storage layer by dragging the “bomb” to his/her friends or getting the “bomb”.

Server & Storage Layer

This layer is composed of two major components: agent (not sure) and a set of storage servers. Agent is responsible for redirecting the requests from client to one or more storage servers. Storage server maintains sufficient replication to be fault tolerant. This set of cluster is responsible for:

- Collecting, storing user names and choosing one of them to put a bomb on his/her screen
- Updating the location of bomb when receiving the requests from client
- Providing the location of bomb when receiving the requests from client
- Using Paxos lib to complete all operations

For the consistency issues when updating location, we are going to use Paxos by calling Paxos functions in Paxos logic layer. In this way, not only the consistency among all servers is assured, but also the failure of servers can be handled.

Paxos Logic Layer

In this layer we will implement a Paxos library to guarantee the consistency and failure tolerance for storage server. This library consists of acceptors and proposers like normal Paxos. This layer promises guarantee that proposals that may come from other proposers will not be erroneously accepted, and in particular they ensure that only the latest of the proposals sent by the proposer is accepted.

Test Plan

The tests consist of four parts: correctness tests, robustness tests, performance tests and integration tests.

Correctness Tests

- Single proposer in a single round of Paxos
- Single proposer in multiple rounds of Paxos
- Multiple proposers in multiple rounds of Paxos

Robustness Tests

- Network Delay
  - Single slow-response acceptor
  - Multiple slow-response acceptors
- Failure-stop
  - Single acceptor fails and stops
  - Multiple acceptors fail and stop
  - Proposer fails and stops
- Failure-Recover
  - Single acceptor fails and recovers
  - Multiple acceptors fail and recovers
  - Proposer fails and recovers

Performance Tests

- Sequential Multiple Requests
  - Through a single proposer to test clear log mechanism
  - Through multiple proposers to test time performance
- Concurrent Multiple Requests
  - Through multiple proposers to test race conditions

Integration Tests

- Combine Agent, Storage server and Paxos lib to test overall process
- Add UI and test the whole application on iOS platform

Tiers

Tier 1

- Implement basic functions of Paxos lib, Agent and Storage server
- Implement correctness tests and pass them

Tier 2

- Implement advanced functions of Paxos lib, Agent and Storage server, i.e. recover mechanism and clear log mechanism
- Implement robustness and performance tests and pass them

Tier 3

- Implement the front end GUI on iOS platform
- Implement integration tests and conduct user testing

Schedule

April 10 – April 14	Design application functions and system structure
April 15 – April 19	Implement basic functions of the backend and pass correctness tests
April 20 – April 23	Implementation advanced functions and pass robustness and performance tests
April 24 – April 26	Implement UI and pass all tests
April 27 – April 29	Refine all parts and prepare presentation