Shanghai Jiaotong University

Big Data Processing

Distributed Lock Design

Author:
WANG Lei
118260910044
WU Tiance
117260910057
GU Yuanzhe
118260910034
LIU Qingmin
118260910037

Supervisor:
WU Chentao

November 30, 2019

1 Problem Description

- Design a simple consensus system, which satisfy the following requirements
  - Contain one leader server and multiple follower server
  - Each follower server has a replicated map, the map is consisted with the leader server
  - The key of map is the name of distributed lock, and the value is the Client ID who owns the distributed lock
- Support multiple clients to preempt/release a distributed lock, and check the owner of a distributed lock
  - For preempting a distributed lock
    - If the lock doesn't exist, preempt success
    - Otherwise, preempt fail
  - For releasing a distributed lock
    - If the client owns the lock, release success
    - Otherwise, release fail
  - For checking a distributed lock
    - Any client can check the owner of a distributed lock
- To ensure the data consistency of the system, the follower servers send all preempt/release requests to the leader server
- To check the owner of a distributed lock, the follower server accesses its map directly and sends the results to the clients
- In this system, all clients provide preempt/release/check distributed lock interface
- When the leader server handling preempt/release requests
  - If needed, modify its map and sends a request propose to all follower servers
  - When a follower server receives a request propose

  - modify its local map
  - check the request is pending or not
  - if the request is pending, send an answer to the client

2 Structure and Method

In our project, we design the distributed lock in Python. There are three main classes in our project, which are **LeaderServer**, **FollowerServer** and **Client**. LeaderServer is responsible for the collection and distribution of lock information. FollowerServer is charge of saving and linking the leader server and client. Client own one lock, he can lock/unlock his own lock and check the status of other locks.

- LeaderServer.py
  In this class, we define multiple functions:
  - *def __init__* : define the port of leader server, and create empty lists to note client information, connection information, follower server information, lock map and lock name[1].
  - *def __new_client__* : define a client id for new client and add its information in the client list. **Return a client id.**
  - *def __new_follower__* : define a follower id for each follower server and its information in the follower server list. **Return a follower id.**
  - *def __lock_map__* : check whether it exists a lock named as the message.
  - *def __lock__* : Lock function is to detect whether the client can have a lock as named by him and whether the client has already have a lock. **Return a state.**
  - *def __release__* Release function is to detect whether there is a lock named as the message and whether the client has already have a lock. **Return a state.**
  - *def __response__* : define solutions when receiving different messages.

[1]In my design, each client can name their own lock.

  - *def run* : bind a socket and keep listening. The leader server create a new thread to process the communication with a follower server or client.

- FollowerServer.py
  In this class, we define multiple functions:

  - *def __init__* : define the port of leader server, the port of itself, and create empty lists to note client information, lock map and follower id[2].
  - *def __new_client__* : define a client id for new client and add its information in the client list. **Return a client id.**
  - *def __connect_with_leaderserver__* : connect with the leader server and send data. After receiving message from leader server, it can obtain its follower id. **Return follower server id.**
  - *def __connect_with_client__* : transmit information between the leader server and client.
  - *def run* : bind two socket ports, one is used to connect with the leader server and the other is used to keep listening for clients.

- Client.py
  In this class, we define multiple functions:

  - *def __init__* : define the server port, server socket and client id.
  - *def __connect_server__* : send message to its server and receive its client id. **Return server socket and client id.**
  - *def __status__* send "Status", the name of lock to server and receive the information of lock status.
  - *def __lock__* send "Lock", the name of lock to server and receive the information of lock status.
  - *def __release__* send "Release", the name of lock to server and receive the information of lock status.
  - *def run* : connect with server and send the request to the server.

[2]Follower id is defined.

3 Usage

There are 8 files in this project.

- LeaderServer.py : Define the class of leaderserver and run the leader server. The first file have to be run in the terminal.
- FollowerServer.py : Define the class of followerserver.
- Client.py : Define the class of client.
- Client_leader.py : Define a client connecting with the leader server. After leaderserver.py running, you can run this file.
- follower_1.py : Define a follower server. The second file have to be run in the terminal.
- follower_2.py : Define a follower server. The second file have to be run in the terminal.
- client_follower_1.py : Define a client connecting with the follower server 1. After follower_1.py running, you can run this file.
- client_follower_2.py : Define a client connecting with the follower server 2. After follower_2.py running, you can run this file.

4 Experiment and Result

As shown in Figure 1, when a client is initialized, generate the client id information based on the user information.

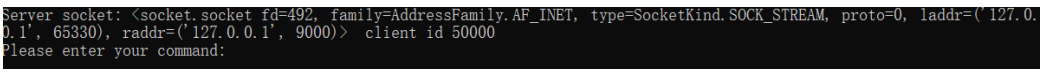

Figure 1: Client Initialization

As shown in Figure 2, for different clients, they have different client ids. For each follower, it has 50,000 positions for client, thus the ID spans 50,000, when we add a client in different server.
As shown in Figure 3 and 4, leader server keeps listening, and follower server connects between the leader server and client.

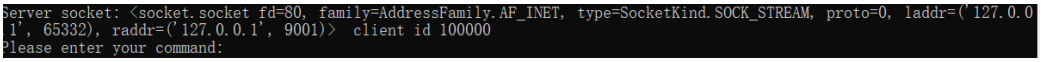

Figure 2: Different client

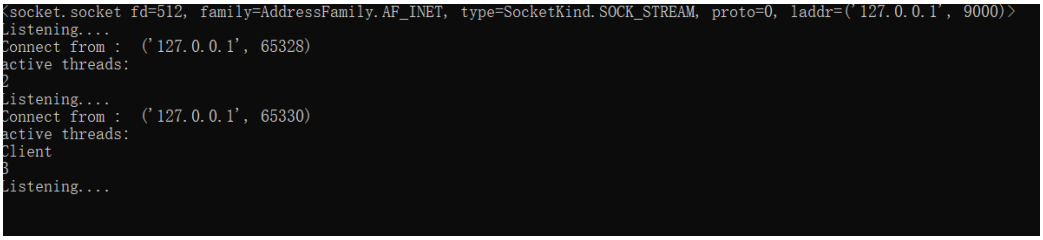

Figure 3: Leader Server

If a client wants to preempt a distributed lock,

- He didn't have a lock before, as shown in Figure 5.
- The name of lock didn't exist, as shown in Figure 6.

If a client wants to release a distributed lock,

- The owner of lock was client, as shown in Figure 7.
- It has already been locked, as shown in Figure 8.

As shown in figure 7, when a client checks the state of lock, server will return the information of its owner, which contains the socket, port, address, etc.

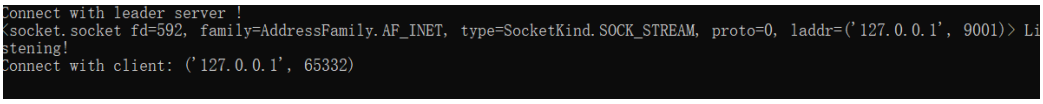

Figure 4: Follower Server

Figure 5: Have only one lock

Figure 6: The name of lock didn't exist

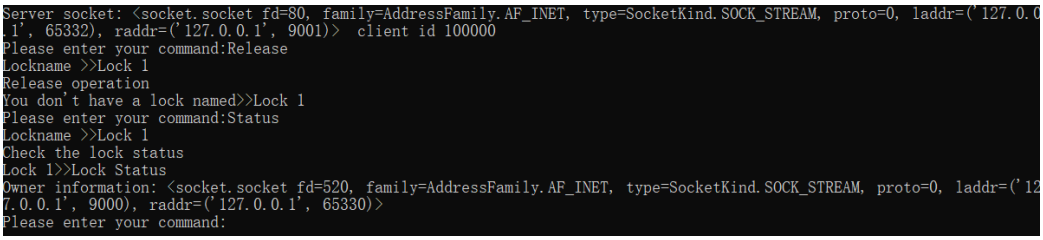

Figure 7: The owner of lock was client

Figure 8: Have been locked

5 Future and Development

In our project, we design a distributed lock. Each client can preempt/release/check the lock. In our next step, we can combine it with the actual scene, rather than implement it in isolation. Besides, we design the distributed lock in python, and we can have a try in others language, especially in Java.