LiteMQ

Zhao Zhixiang 11711621
dept. CSE of SUStech
11711621@mail.sustech.edu.cn

***Abstract*—This project is to build a kafka-like MQ named LiteMQ and with a simple cluster management name Lite ZK.**

I. Introduction

A simpliied, Kafka-like distributed messaging system called LiteMQ is implemented in this project with following functionalities:

- Publish/Subscribe Pattern
- Consumer/Producer API
- A central configuration / coordination service
- Partitioning for topic
- Fault tolerance for nodes
- Leader election for nodes/partitions

II. Proposed System Design

I recognize the system as a distributed file system. And it is based on JAVA RMI.

The developing steps are:

1) Connection: First the setup the LiteZK, which is a file system too but of a small size. it will create several directories such as "/brokers", "/admin", "/brokers/ids". And it maintain a map contains the brokerId and the corresponding Registry that is a component in Java RMI. And then we can run several broker and it will try to connect the LiteZK. If successful, then the LiteZK will create a file with zero byte in it and "$brokerId$.zk". And and create a local Registry. The broker will get its broker ID and other infomation.
2) Heartbeat Daemon: This Daemon will continuously detect the path "/brokers/ids" and check if there is a new ID file created. If there is, it will start a timer and reveives the packet of the heartbeat from the broker. If it cannot recieve the heartbeat then it will delete the ID file that can be treated as a shared memory.
3) setting remote object of the broker: this kind of remote object is for creating a topic, corresponding partitions and replicas.
4) writing remote obeject of the broker: this kind of remote object is for the producer to send the record of data to the broker. The producer will first store the data into the buffer named byteBuffer, (nio.Buffer) if it reaches a defined position then the data from buffer will be transmitted to the leader producer. This fixed-size data can be wrapped into a segment stored in partition. After the leader finishes all the tasks of replicas, it will send a commit to the producer and set the returned future. It is "all Acks" mode.
5) reading remote object of the broker: the offset is the position in read mode after filping the ByteBuffer. And the consumer group is a bit more complicated design and I do not touch it.
6) leader election: I want to simplify it. When the leader was down, the LiteZK will send the info to all the brokers. The fastest packet recieved from a specific broker will be a new leader. i.e. First reply and first win.
7) LiteZK now is a single point of failure: If I choose to improve it, it is a little hard to expend it for acting like the real ZooKeeper since the Registries. It is difficult to design Zab and Znode such as normal mode and recover mode.
8) Exactly-once semantics: I do not know how it works. Does it works in producer side or consumer side? I think it cannot work at consumer if we do not know where the data will go and how they are consumed. That is impossible. I think it at most can achieve at least once or at most once. And if it really can, does it worth doing it? I doubt.

III. Future Work

Now my work is at the second part... I need to speed up and finish it. This is not easy. And I just waste a lot of time. I am such a fool, I am stupid.