☁️NaiveKV: A distributed, persistent key-value store

Yijun Chen, 517021910387, July 2020

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Contents

1 Architecture Overview

2 Deploying zookeeper

3 Persistence capabilities

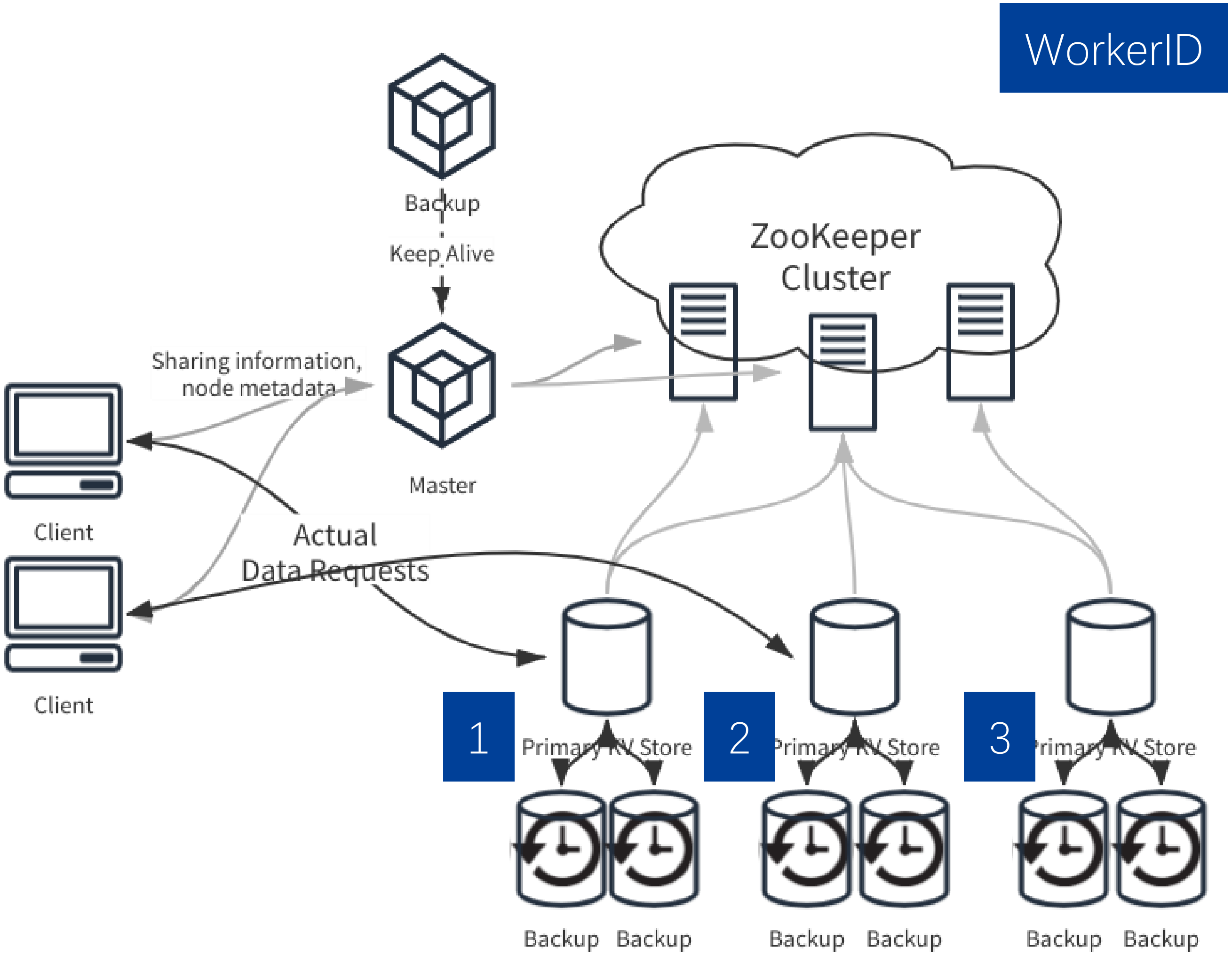
4 Load balancing

5 Migration, backup sync

6 Master node high-availability

7 Client

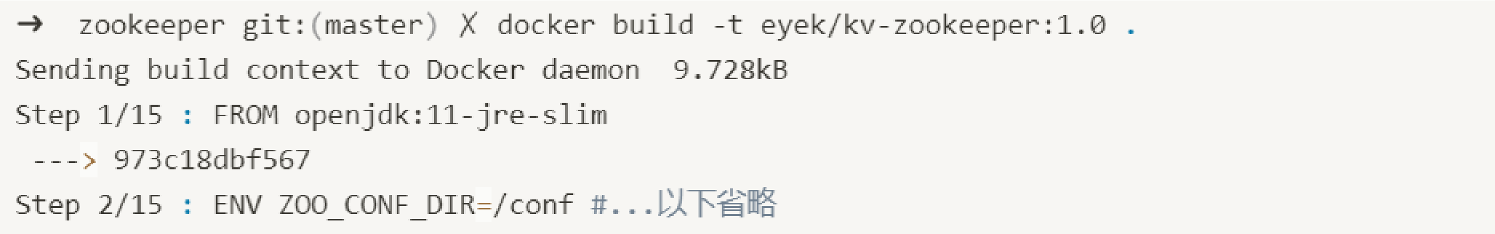
Architecture Overview



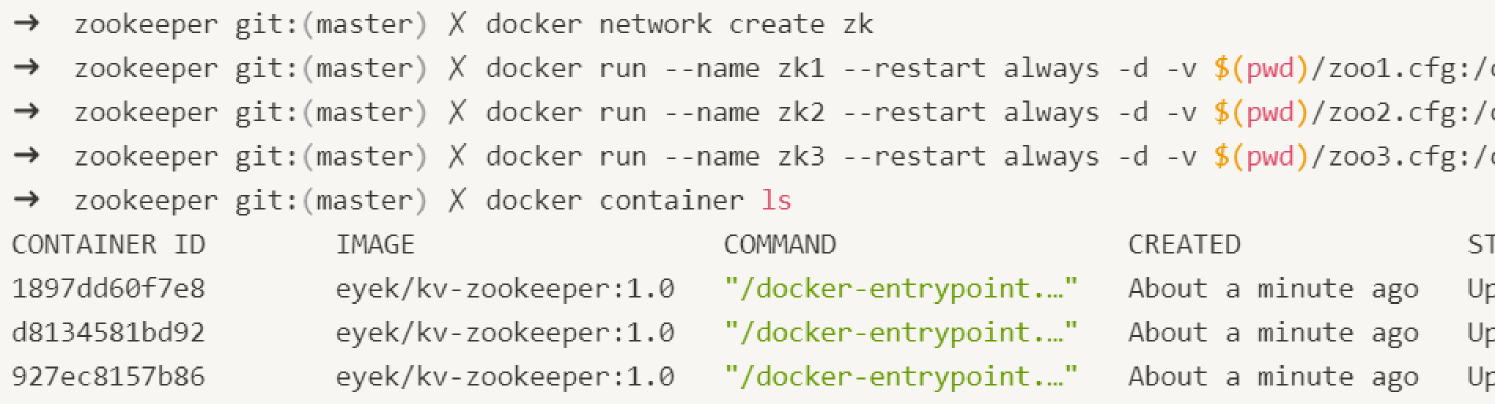
Architecture Overview

- The system is made up of master node(s), primary node(s), backup node(s) and the zookeeper cluster.
- Primary nodes and backup nodes are both “worker” nodes. Primary node and backup node that are in the same group shares the same Worker ID, which is the unit of load balancing and hash slot allocation for master node.
- Master node(s) and worker node(s) communicate only through modifying metadata stored in zookeeper.

Deploying ZooKeeper

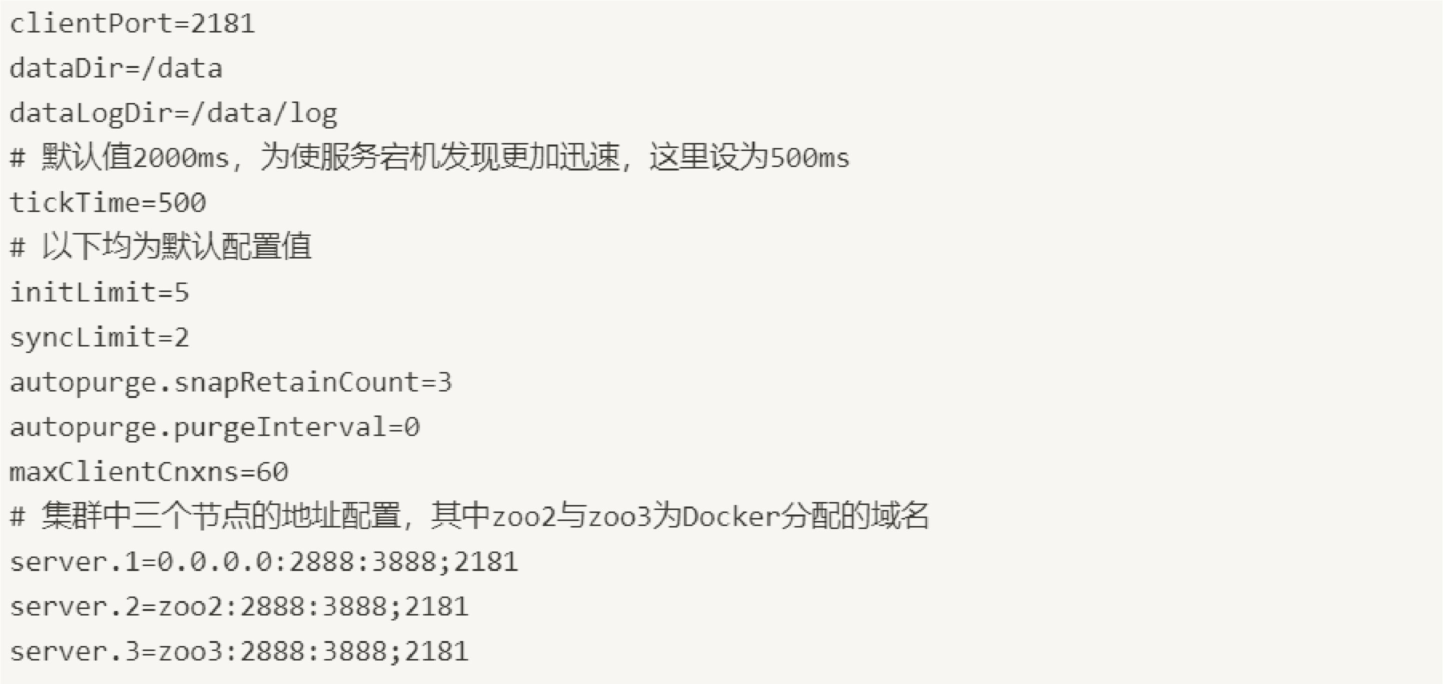


```
➜  zookeeper git:(master) ✗ docker build -t eyek/kv-zookeeper:1.0 .
Sending build context to Docker daemon  9.728kB
Step 1/15 : FROM openjdk:11-jre-slim
 ---> 973c18dbf567
Step 2/15 : ENV ZOO_CONF_DIR=/conf #...以下省略
```



```
➜  zookeeper git:(master) ✗ docker network create zk
➜  zookeeper git:(master) ✗ docker run --name zk1 --restart always -d -v $(pwd)/zoo1.cfg:/
➜  zookeeper git:(master) ✗ docker run --name zk2 --restart always -d -v $(pwd)/zoo2.cfg:/
➜  zookeeper git:(master) ✗ docker run --name zk3 --restart always -d -v $(pwd)/zoo3.cfg:/
➜  zookeeper git:(master) ✗ docker container ls
CONTAINER ID        IMAGE                    COMMAND                  CREATED              ST
1897dd60f7e8        eyek/kv-zookeeper:1.0    "/docker-entrypoint.…"   About a minute ago   Up
d8134581bd92        eyek/kv-zookeeper:1.0    "/docker-entrypoint.…"   About a minute ago   Up
927ec8157b86        eyek/kv-zookeeper:1.0    "/docker-entrypoint.…"   About a minute ago   Up
```

Deploying ZooKeeper: configuration



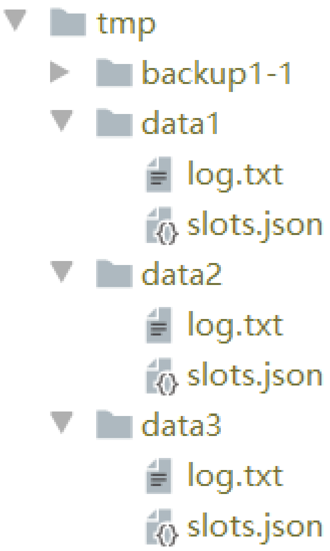
```
clientPort=2181
dataDir=/data
dataLogDir=/data/log
# 默认值2000ms，为使服务宕机发现更加迅速，这里设为500ms
tickTime=500
# 以下均为默认配置值
initLimit=5
syncLimit=2
autopurge.snapRetainCount=3
autopurge.purgeInterval=0
maxClientCnxns=60
# 集群中三个节点的地址配置，其中zoo2与zoo3为Docker分配的域名
server.1=0.0.0.0:2888:3888;2181
server.2=zoo2:2888:3888;2181
server.3=zoo3:2888:3888;2181
```

Metadata in zookeeper

- /kv
  - workers Registration service, weight information for all workers.
  - migrations each round’s migration plan
  - election playground for backups to re-elect a primary node
  - masters Registration service for all masters(and master leader election)
  - slots Saves current hash slot allocation table
  - version Saves current hash slot allocation table version
  - workerId Saves next available workerID

Persistence capabilities

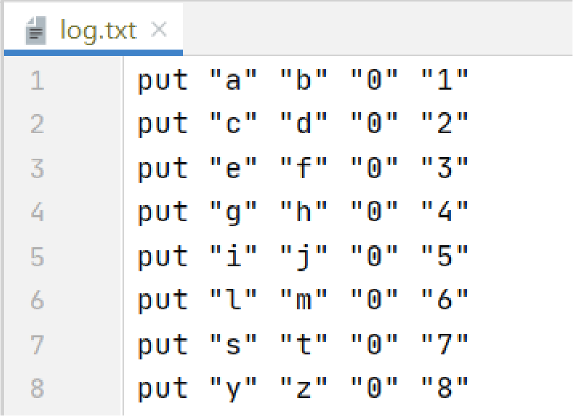
- To provide fault-tolerance for single node.
- A WAL-based data persistent layer, consists of a base file, a redo log and in-memory lookup cache, like SQLite.
- When committing a change: first change in memory, write log into OS cache, use `sync` to flush change to disk, return.



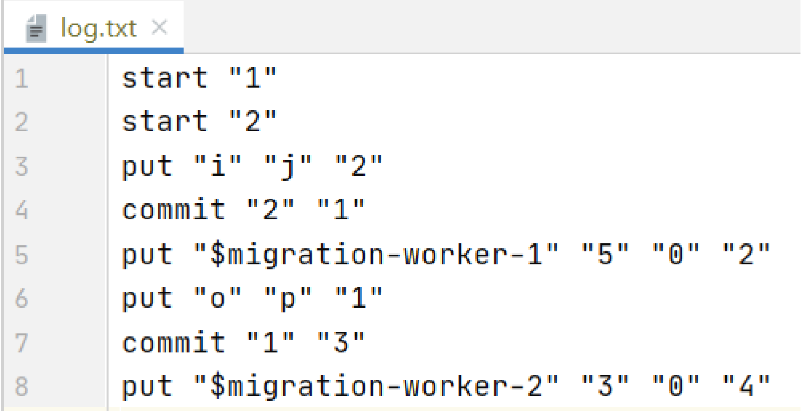
```
▼ tmp
  ▶ backup1-1
  ▼ data1
      log.txt
      slots.json
  ▼ data2
      log.txt
      slots.json
  ▼ data3
      log.txt
      slots.json
```

Persistence capabilities

- Log format is:
- <operation> <arg1> <arg2> <transactionNumber> <versionNumber>
- Transaction number is here to support concurrent transactions. Will be introduced later.



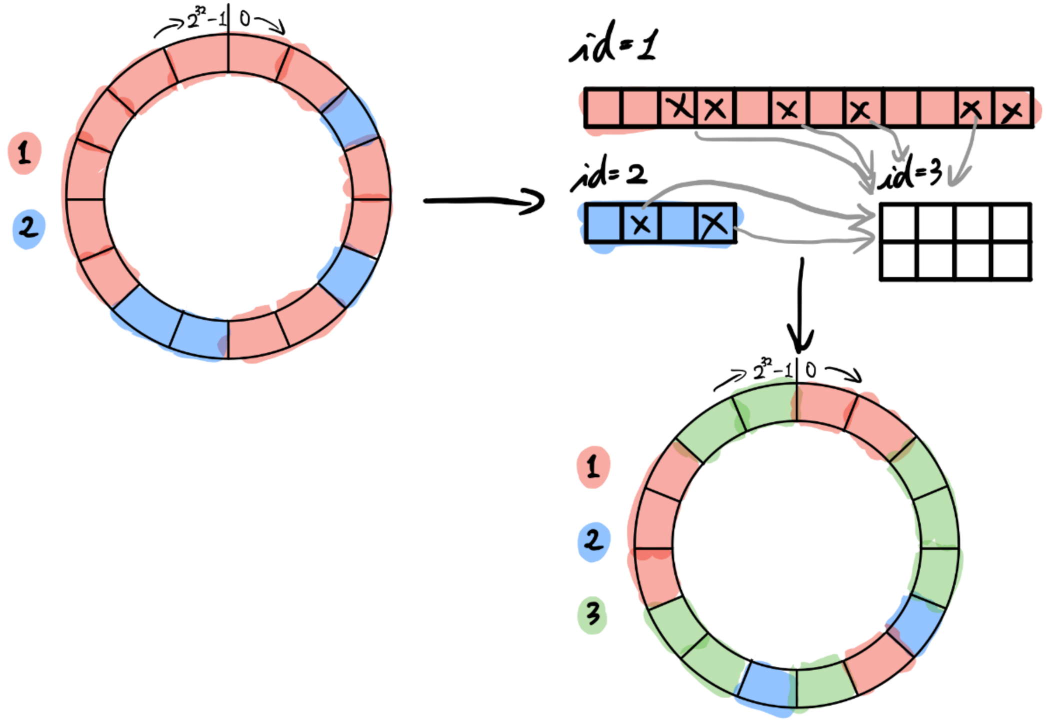
```
put "a" "b" "0" "1"
put "c" "d" "0" "2"
put "e" "f" "0" "3"
put "g" "h" "0" "4"
put "i" "j" "0" "5"
put "l" "m" "0" "6"
put "s" "t" "0" "7"
put "y" "z" "0" "8"
```



```
start "1"
start "2"
put "i" "j" "2"
commit "2" "1"
put "$migration-worker-1" "5" "0" "2"
put "o" "p" "1"
commit "1" "3"
put "$migration-worker-2" "3" "0" "4"
```

Load Balancing

- Use hash slot allocation. 1024 slots by default.
- Slots allocated according to each worker node’s weight.



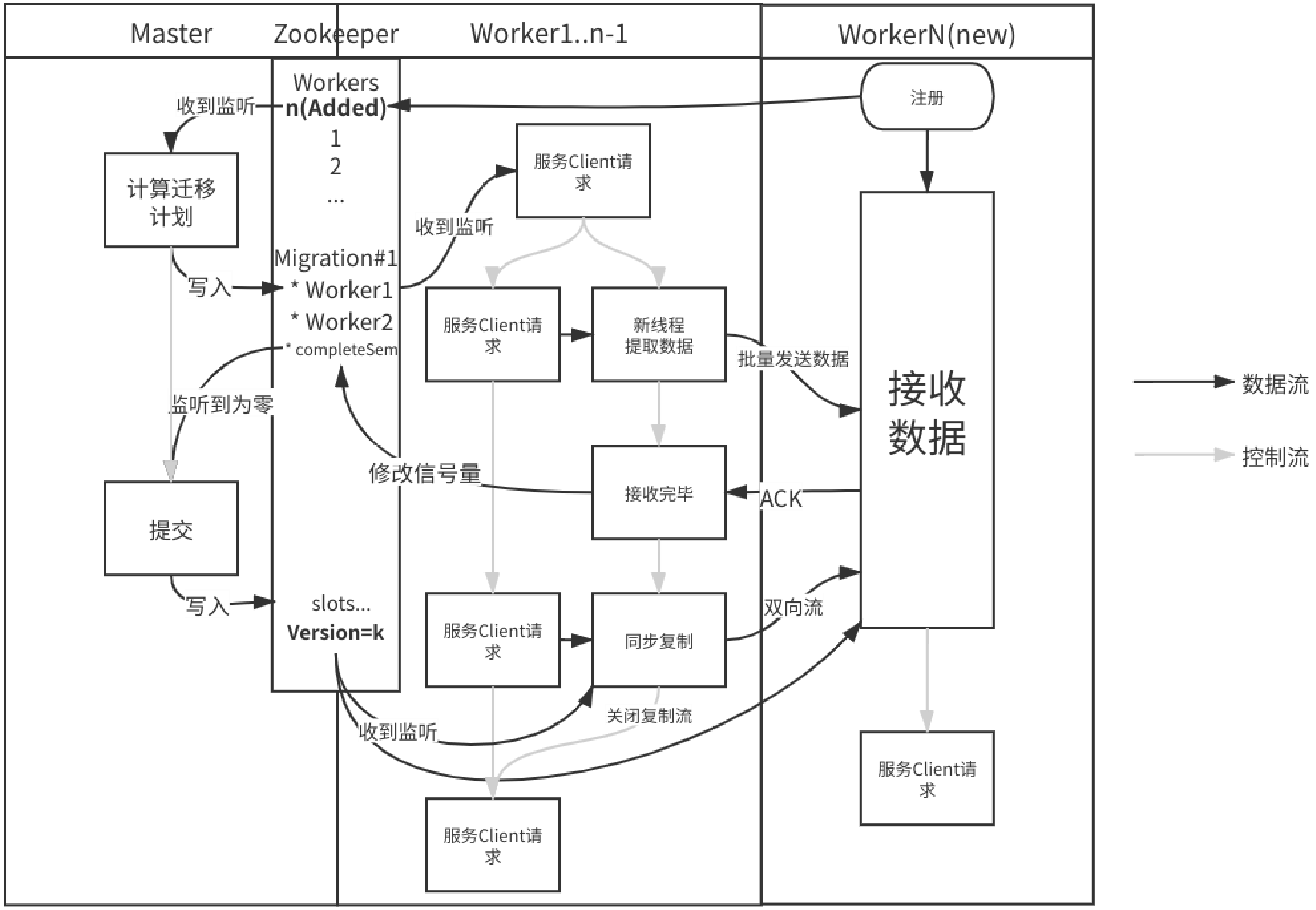
Migrating between primary nodes

- Migration happens when new primary node is added into the cluster.
- As described before, every primary node would participate in the migration process.
- Problems:
    - Crash consistency. We need some level of atomicity to ensure no data corruption, and that the migration process can be resumed after either side crashes.
    - Concurrency: It’s better not to lock both sides when doing migration, otherwise the whole cluster will be unavailable during migration.

Two steps of migration

- Full-size replication: Use transaction to maintain all-or-nothing atomicity.
- Incremental replication: Use synchronized replication. Data is first committed to destination worker node before being committed in source worker node.
- Both implemented with gRPC’s streaming RPC(long connection).

Two steps of migration

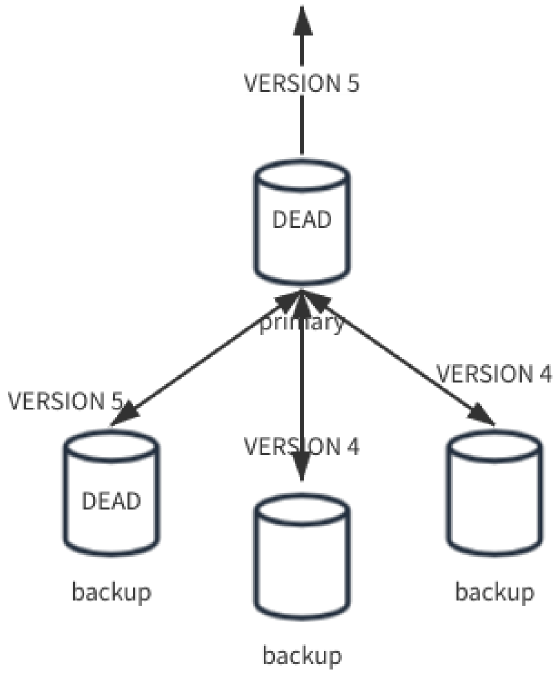


Primary-backup synchronization

- Semi-synchronous replication, similar with MySQL.
- If have multiple backups, the primary only have to receive ACK **from one of them** before committing and return to client.
- When primary crash:
  - Backup nodes detect that primary node is down, begin election process.
  - Elect out the one with **highest version number**. Those who have same version number, compare who is first to register in zookeeper.
  - The one got elected becomes primary temporarily, and receives data requests.
  - When the primary is back online, the temporary primary node **sync data with the primary and step down**.

Primary-backup synchronization

- What if primary and backup crash at the same time?
- No way to guarantee that data is consistent in remaining backup nodes.
- Service downgrade to read only, to avoid undoing.



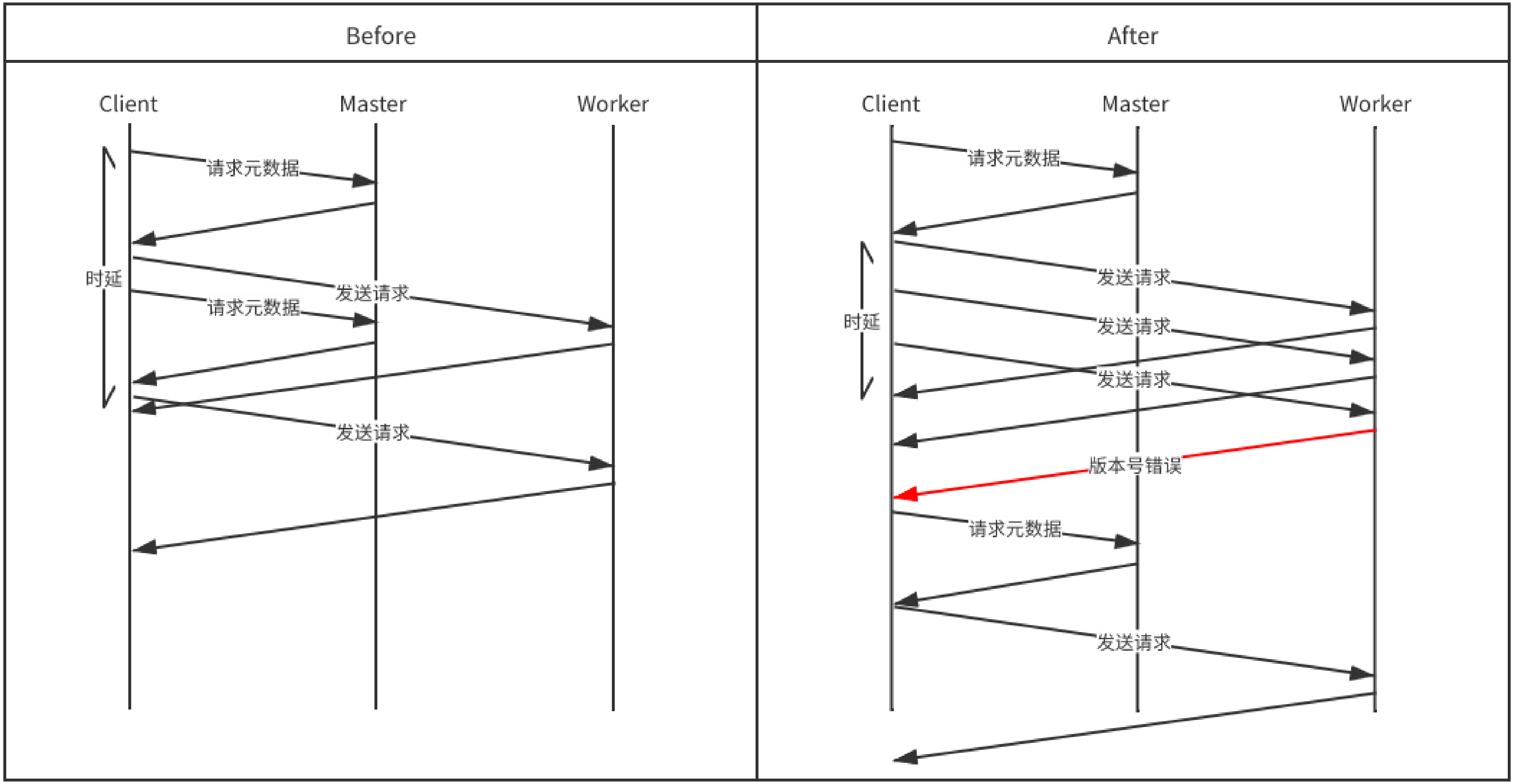
Master node high-availability

- Master node provides the client with worker metadata. This part is stateless(zookeeper proxy), easy to replicate.
- Master node also orchestrates the migration process. This require that only one master can operate at the same time.
- Implement a distributed lock in zookeeper’s /kv/masters directory, with sequential ephemeral zNodes.

Client, and an optimization

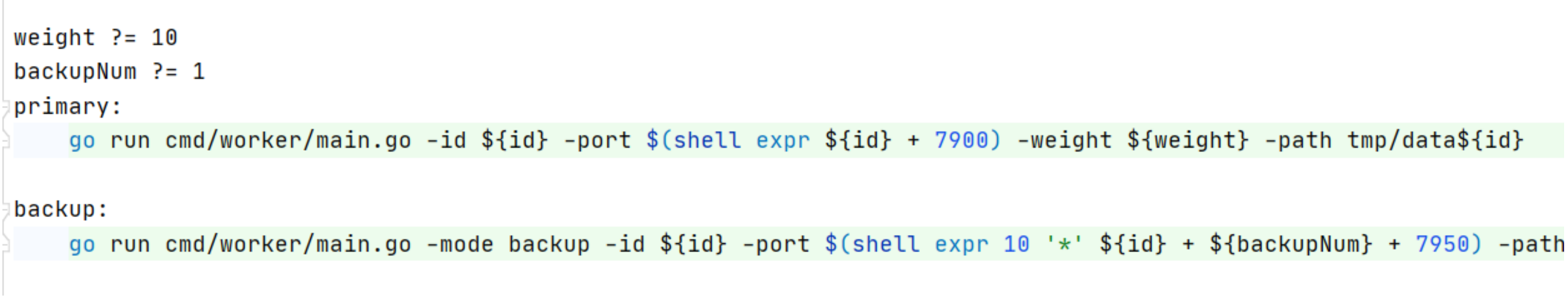
- Client is an REPL which receives user's request from keyboard and send requests to master node and worker node.
- Before: Requesting worker metadata from master before every single requests.
- After: Request all worker metadata from master in a single request. **Include a slot allocation version number in every request sent to worker.** Worker checks the version number, and refuses to serve if version number is wrong. Client refresh its metadata.
- This optimization is implemented after DDL.

Client, and an optimization



Implementation and deploying

- Implemented in Go with gRPC, roughly 3k~3.5k lines of code.
- Directly run with **go run** ···
- Hostname and port allocation logic is in Makefile. Supports hostname and port customization.



```
weight ?= 10
backupNum ?= 1
primary:
    go run cmd/worker/main.go -id ${id} -port $(shell expr ${id} + 7900) -weight ${weight} -path tmp/data${id}

backup:
    go run cmd/worker/main.go -mode backup -id ${id} -port $(shell expr 10 '*' ${id} + ${backupNum} + 7950) -path
```

谢谢！