# Lab 5 – Distributed Key-Value Storage System

## 目录

# 功能简述

- 由三个结点组成的 ZOOKEEPER 集群存储系统的元数据
- 系统由一个 primary 结点、至少大于等于 2 个的 worker 结点以及每个 worker 的 stand by 结点组成
- 实现了一个简单的 KV 内存数据库 RingoDB，Key 和 Value 的类型都是 String
- Client 能做 PUT, GET, DELETE 三个操作
- 在 client、primary 结点、worker 结点（包括 primary data node 和 standby data node）之间的通信都使用 SofaRPC
- 支持 Scale Out，实现了自己的一致性哈希算法，可以动态加入 worker 结点，并在 Scale Out 过程中只影响一部分数据的写操作，读操作不受影响
- 支持 multi-client 与 Concurrent Data Accessing
- primary data node 异步复制数据到 standby data node，当 primary data node FAIL，ZOOKEEPER 会选举一个 standby data node 成为 primary data node。

- 所有 data node 都会每隔一段时间将数据库的 snapshot 写入磁盘，worker FAIL 之后 recover 时会读取该 snapshot

# 环境配置

JAVA8 安装

```
sudo apt-get install openjdk-8-jdk
```

ZOOKEEPER 下载

```
wget https://mirrors.aliyun.com/apache/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz
tar -xzf zookeeper-3.4.14.tar.gz
```

ZOOKEEPER 集群模式

先写好启动的配置文件，一共三个 zookeeper server,写三个配置文件。

由于三个 server 在同一台机器上部署，clientPort 分别是 2181，2182，2183

配置文件 z2.cfg 如下

```
tickTime =2000
initLimit=10
syncLimit=5
dataDir=./data
clientPort=2182
server.1=0.0.0.0:2222:2223
server.2=0.0.0.0:3333:3334
server.3=0.0.0.0:4444:4445
```

ZOOKEEPER server 启动命令

```
../zookeeper-3.4.14/bin/zkServer.sh start ./z2cfg
```

下图显示 server.2 成为 zookeeper leader

```
 INFO  [QuorumPeer[myid=2]/0:0:0:0:0:0:0:0:2182:Leader@380] - LEADING - LEADER ELECTION TOOK - 258
 INFO  [LearnerHandler-/172.16.170.226:34538:LearnerHandler@346] - Follower sid: 1 : info : org.apa
uorumServer@6509777d
 INFO  [LearnerHandler-/172.16.170.226:34538:LearnerHandler@401] - Synchronizing with Follower sid:
0x0 peerLastZxid=0x0
 INFO  [LearnerHandler-/172.16.170.226:34538:LearnerHandler@410] - leader and follower are in sync,

 INFO  [LearnerHandler-/172.16.170.226:34538:LearnerHandler@475] - Sending DIFF
 INFO  [LearnerHandler-/172.16.170.226:34538:LearnerHandler@535] - Received NEWLEADER-ACK message f
 INFO  [QuorumPeer[myid=2]/0:0:0:0:0:0:0:0:2182:Leader@964] - Have quorum of supporters, sids: [ 1,
sed zxid: 0x100000000
```

下图显示 primary 结点与 zookeeper server3 进行连接

```
] - Session establishment complete on server 112.124.23.139/112.124.23.139:2183, se
```

JENKINS 根据 git push 自动部署 Worker

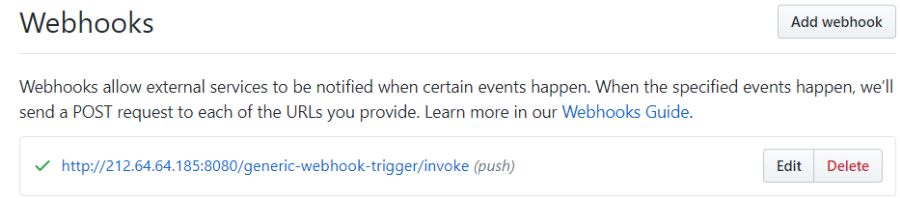
Jenkins 的工作目录

/var/lib/jenkins/workspace/kv-store

Jenkins 启动命令

```
sudo systemctl start jenkins
```

github 设置相应的 WEBHOOK 给 JENKINS 发送 BUILD 通知




杀死之前运行的进程 clear.sh

```
sudo netstat -tunlp| grep 12201 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #primary worker1的端口
sudo netstat -tunlp| grep 12202 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #primary worker2的端口

sudo netstat -tunlp| grep 12401 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #standby1 worker1的端口
sudo netstat -tunlp| grep 12402 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #standby2 worker1的端口

sudo netstat -tunlp| grep 12403 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9  #standby1  worker2的端口
sudo netstat -tunlp| grep 12404 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #standby2 worker2的端口
sudo netstat -tunlp| grep 12200 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #primary的端口
sudo netstat -tunlp| grep 12301 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #worker3-primary的端口
#sudo netstat -tunlp| grep 12302 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #worker4-primary的端口

sudo netstat -tunlp| grep 12501 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #standby1 worker3的端口
sudo netstat -tunlp| grep 12502 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #standby2 worker3的端口

#sudo netstat -tunlp| grep 12503 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #standby1 worker4的端口
#sudo netstat -tunlp| grep 12504 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #standby2 worker4的端口
```

部署 Worker 的脚本 deployworkers.sh

然后在不同的路径启动 worker 进程以及 standby data node 进程

```
sh  ./kv-store/target/clear.sh
sleep 15s
mkdir worker1-primary
mkdir worker2-primary
mkdir worker1-standby1
mkdir worker1-standby2
mkdir worker2-standby1
mkdir worker2-standby2

cd worker1-primary
sudo rm -rf log4j2.log
sudo  find ./ -name 'snapshot*-*' -exec rm {} \;
java -cp  ../kv-store/target/kv-store-1.0-SNAPSHOT.jar Worker  112.124.23.139:2181,112.124.23.139:2182,112.124.23.139:2183 \
 212.64.64.185:12201  212.64.64.185:12201 notrecover & #primary data node
cd ..

cd worker2-primary
sudo rm -rf log4j2.log
sudo  find ./ -name 'snapshot*-*' -exec rm {} \;
java -cp  ../kv-store/target/kv-store-1.0-SNAPSHOT.jar Worker  112.124.23.139:2181,112.124.23.139:2182,112.124.23.139:2183 \
 212.64.64.185:12202   212.64.64.185:12202   notrecover & #primary data node
cd ..

cd worker1-standby1
sudo rm -rf log4j2.log
sudo  find ./ -name 'snapshot*-*' -exec rm {} \;
java -cp  ../kv-store/target/kv-store-1.0-SNAPSHOT.jar Worker  112.124.23.139:2181,112.124.23.139:2182,112.124.23.139:2183 \
 212.64.64.185:12201  212.64.64.185:12401   notrecover & #standyby1 data node
cd ..
```

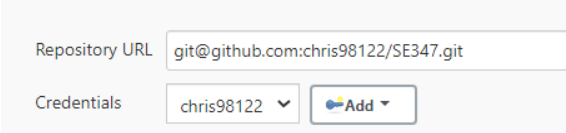
testWorkerFail.sh 先杀死 worker1-primary 进程

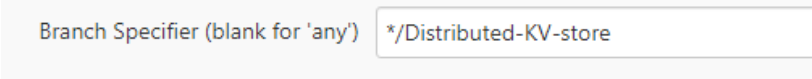
然后过 20 秒用 recover 模式启动 worker1-primary 进程

```
sleep 3m

sudo netstat -tunlp| grep 12201 | awk  '{print $7}' |cut -d"/" -f1 |sudo xargs kill -9 #primary worker1的端口

sleep 20s

#recover primary worker1
cd worker1-primary
java -cp  ../kv-store/target/kv-store-1.0-SNAPSHOT.jar Worker  112.124.23.139:2181,112.124.23.139:2182,112.124.23.139:2183 \
 212.64.64.185:12201  212.64.64.185:12201 isrecover & #primary data node
```

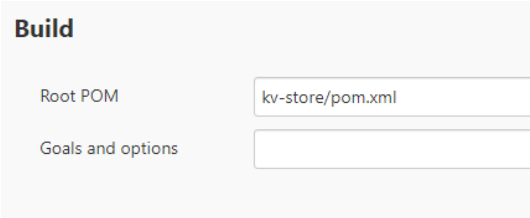
Jenkins 自动 BUILD

先配置好 SSH key，然后自动从 git repo pull 最新 commit



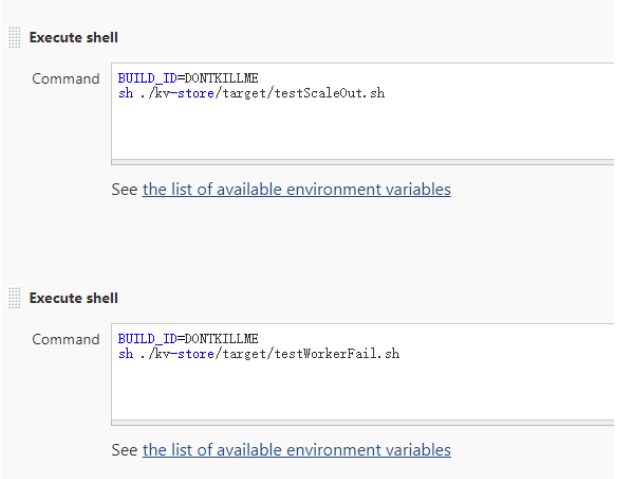




使用 maven 根据 pom.xml build



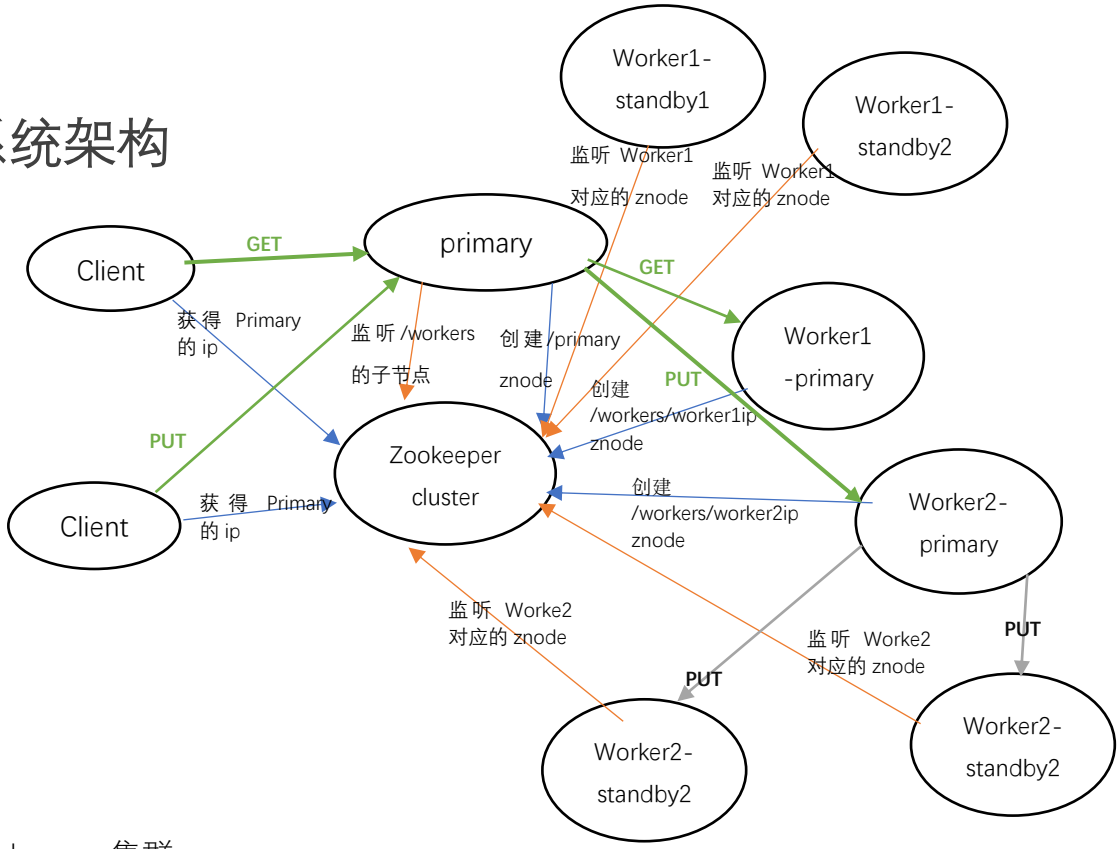

Pom 文件设定了用 maven-assembly-plugin 把工程打包成 jar-with-dependencies

Jenkins 自动部署并测试


BUILD_ID = DONTKILLME 保证了后台进程不会被 JENKINS 杀死

# 系统架构




Zookeeper 集群

出于 zookeeper 的可靠性考虑，本次 Lab 选用仲裁模式配置的 zookeeper 集群。仲裁模式指集群内部选举其中一个 zookeeper server 作为群首，剩下的服务器作为追随者，当具有符合法定仲裁（三分之二）的可用服务器时，便可向集群外部提供服务。本次 Lab 启用了三个 zookeeper server。
zookeeper 集群在本次 lab 用于注册 primary 结点的信息，监视 worker 结点状态，以及选举 worker 结点等功能。

Client 的设计

Client 支持命令行输入，如下图

```
please enter PUT or GET or DELETE or QUIT
PUT
input key:
ringo
input value:
apple
OK
please enter PUT or GET or DELETE
GET
input key:
ringo
apple
please enter PUT or GET or DELETE
```

Client 的代码逻辑

1. 从 zookeeper 集群获得/primary znode 的数据（即 primary 的 ip）
2. 通过 SOFARPC 的客户端对 primary 发送 PUT/GET/DELETE 请求

## Primary 的设计

### 功能介绍

- 通过在 ZooKeeper 中维持一个短暂（EPHEMERAL）类型的 znode，从而选举成为主节点，并将 ip 写入 znode。
- 实现一致性哈希，给 worker 节点分配 key 的范围。
- 将 Client 发来的请求，做并发控制，并分配给知一致性哈希环上相应的 worker 节点。
- 监视/workers znode 下的子节点，如果发生 worker 的领导权转移则更新相应的 worker ip 信息，如果 worker fail 则更新相应的 worker 状态为 FAIL，如果有新的 worker 加入则给新 worker 节点分配 key 的范围，并通知一致性哈希环上相应的 worker 转移数据和重新设置 key 的范围。

### 向 Client 提供的接口

```
String PUT(String key, String value);

String GET(String key);

String DELETE(String key);
```

### 向 Worker 提供的接口

```
String notifyTransferFinish(String WorkerSenderAddr, String newKeyEnd);
```

## Worker 的设计

### 功能介绍

Primary Data Node 的功能

- 通过在 ZooKeeper 中维持一个短暂（EPHEMERAL）类型的 /workers/workerip:workerport 节点，从而选举成为主节点，并将 ip 与端口写入 znode。
- 接收 Primary 发来的 SetKeyRange 请求
- 接收 standby 节点发来的注册信息
- 处理 Primary 发来的请求，并将写操作做并发控制异步复制给 stand by 节点。
- 隔一段时间对数据持久化，将 snapshot 写入磁盘
- 接收 Primary 发来的 ResetKeyEnd 请求

Standby Data Node 的功能

- 监视/workers/workerip:workerport 节点,如果节点状态转为 NoNode 就竞争该结点

- 向 Primary Data Node 注册自己为 standby 节点
- 接收 Primary Data Node 发来的 SetKeyRange 请求

向 Primary 提供的接口

```
String SetKeyRange(String keystart, String keyend, boolean dataTranfer);

String ResetKeyEnd(String oldKeyEnd, String NewKeyEnd, String WorkerReceiverADRR);

String PUT(String key, String value);

String GET(String key);

String DELETE(String key);
```

Standby Data Node 向 Primary Data Node 提供的接口（复用一部分接口）

```
String SetKeyRange(String keystart, String keyend, boolean dataTranfer);

String PUT(String key, String value);

String GET(String key);

String DELETE(String key);
```

Primary Data Node 向 Standby Data Node 提供的接口

```
String RegisterAsStandBy(String StandByAddr);
// Standby Data Node -> Primary Data Node
```

# RingoDB 的设计与实现

RingoDB 是基于本次 LAB 设计实现的一个带有持久化与恢复功能的 KV 内存数据库。

## ConcurrentHashMap 数据结构

ConcurrentHashMap 是 Java 并发包中提供的一个线程安全且高效的 HashMap 实现,且 ConcurrentHashMap 采用了"分段锁"策略, 来保证线程竞争激烈的并发场景中依旧有良好的性能。所以本次 lab 基于性能与线程安全的考虑使用 ConcurrentHashMap 作为内存数据库的存储结构。

向 Worker 提供的接口

Put/ Get/ Delete

检查 key 是否为空，不为空就对 ConcurrentHashMap 进行相应操作。

hasValueInRange/TrunkMap/ setMap

这三个接口都仅在 ScaleOut 过程中才会用到
hasValueInRange 用于检查是否有 key 在一致性哈希环上指定的范围内。
TrunkMap 用于仅仅保留在一致性哈希环上指定的范围内的键值对，而删除其他键值对。
setMap 用于赋值 ConcurrentHashMap

snapshot/delete_oldest_snapshot

snapshot 用于持久化当前数据，实现思路是将 ConcurrentHashMap 的拷贝序列化并写入文件。
delete_oldest_snapshot 用于删除目录下能找到的最早的 snapshot 文件。

recover

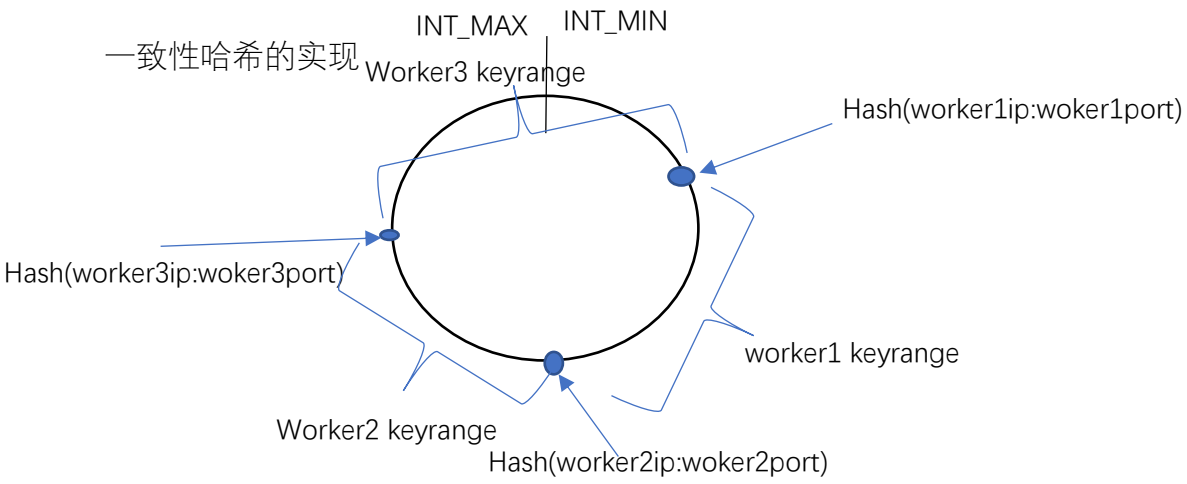
recover 读取能找到的时间最近的 snapshot，反序列化文件内容并恢复 ConcurrentHashMap

单例模式

单例模式确保某个类只有一个实例，而且自行实例化并向整个系统提供这个实例。
RingoDB 使用 enum 关键字来实现单例模式，这样做好处是代码简洁，且初始化 RingoDB 对象时是线程安全的。

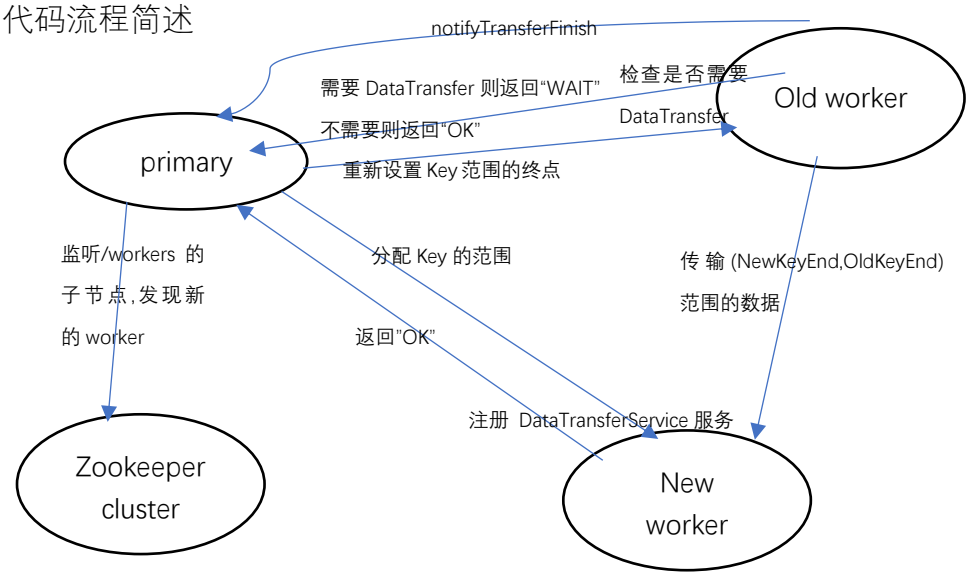
# Scalability 的设计与实现

一致性哈希的实现



如图所示，将 worker1ip:woker1port 字符串(e.g.“212.64.64.185:12201“)哈希后在环上的位置作为 worker 所被分配到的 KEY 的起点。KEY 的终点由在环上的下一个 worker 的起点决定。一致性哈希以及 PUT/GET/DELETE 传来的 key 的 Hash 函数都选择使用 MD5，因为 MD5 对于相似的字符串的哈希值差别很大，优点是相似的输入也能被分散到不同 worker。

## 代码流程简述




当 Primary 监听/workers 的子节点的变化时，会运行 processWorkerChange 线程。processWorkerChange 线程发现有新的 worker 结点加入时，会运行 scaleOut 线程。scaleOut 进行两步操作。

1. 计算新的 worker 结点的 key range，向新的 worker 结点发送 SetKeyRange 请求
2. 如果第一步成功则向被分片的 worker 发送 resetKeyRange 请求，返回值为“WAIT”说明 Primary 结点需要等待传递数据完毕的通知，返回值为“OK”说明无需传递数据。

这里将数据复制的操作用异步的方式实现，在完成后再通知 Primary，主要是考虑到大量数据传输下耗时较久，如果同步等待数据传完很可能会使得 resetKeyRange 请求超时。

由于被分片的 worker 的数据需要复制到新加入的 worker 结点，所以在第二步的过程中，被分片的 worker 的状态会被设成只读。此时 Client 发来的 PUT 和 DELETE 操作的 KEY 如果落在被分片的 worker 的 key range 内，会收到"the service is not available right now."的回复，GET 操作不受影响。

## 运行情况

## 服务不受影响的情况举例

Primary 的 LOG
表明 ScaleOut 期间两个 PUT 操作都有正常完成。
第一个 PUT 落在被分片的 worker 的 key range 内，但是处于 scaleOut 两步操作的第一步所以不受影响。

第二个 PUT 没有落在被分片的 worker 的 key range 内，所以不受影响。

```
2020-06-24 07:33:25,120 [scaleOut1] INFO  [Primary] - ScaleOut triggered
2020-06-24 07:33:25,154 [scaleOut1] INFO  [Primary] - 822816550 >= -396654193 and < 1302869320
2020-06-24 07:33:25,154 [scaleOut1] INFO  [Primary] - workerReiverService.SetKeyRange()
2020-06-24 07:33:25,942 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - 101219426 >= -396654193 and < 1302869320
2020-06-24 07:33:25,942 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - ASSIGN PUT 1535417895:1535417895 TO 212.64.64.185:12202
2020-06-24 07:33:25,942 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  get lock of 1535417895
2020-06-24 07:33:25,967 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  keyRWLockMap.remove(key)
2020-06-24 07:33:26,900 [scaleOut1] INFO  [Primary] - workerReiverService.SetKeyRange()
2020-06-24 07:33:26,922 [scaleOut1] INFO  [Primary] - resetKeyRange 212.64.64.185:12202 TO 822816550
2020-06-24 07:33:26,937 [scaleOut1] INFO  [Primary] - resetKeyRange WAIT
2020-06-24 07:33:26,937 [scaleOut1] INFO  [Primary] - wait for 212.64.64.185:12202 to notify datatransfer finish
2020-06-24 07:33:26,975 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - -687383825 >= 1302869320 or < -396654193
2020-06-24 07:33:26,975 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - ASSIGN PUT 881190480:881190480 TO 212.64.64.185:12201
2020-06-24 07:33:26,975 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  get lock of 881190480
2020-06-24 07:33:26,982 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  keyRWLockMap.remove(key)
2020-06-24 07:33:27,067 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - GET notifyTransferFinish
2020-06-24 07:33:27,067 [scaleOut1] INFO  [Primary] - add 212.64.64.185:12301  workerkeymap and workerState
2020-06-24 07:33:27,068 [scaleOut1] INFO  [Primary] - ScaleOut 212.64.64.185:12301 Finished
```

被分片的 Worker2 的 LOG

```
2020-06-24 07:33:26,925 [SOFA-SEV-BOLT-BIZ-12202-3-T20] INFO  [Worker] - ready ResetKeyEnd to 822816550
2020-06-24 07:33:26,925 [SOFA-SEV-BOLT-BIZ-12202-3-T20] INFO  [Worker] - checkNeedDataTransfer: 822816550 1302869320
2020-06-24 07:33:26,926 [SOFA-SEV-BOLT-BIZ-12202-3-T20] INFO  [DB.RingoDB] - SplitMap
2020-06-24 07:33:26,939 [datatransfer] INFO  [Worker] - do datatransfer: {35=35, 17=17, 62=62, 30=30, 766146214=7661462
2020-06-24 07:33:26,939 [datatransfer] INFO  [Worker] - GetServiceByWorkerADDR 212.64.64.185:12301
2020-06-24 07:33:27,073 [datatransfer] INFO  [Worker] - notifyTransferFinish OK
2020-06-24 07:33:27,073 [datatransfer] INFO  [DB.RingoDB] - {48=48, 1535417895=1535417895, 795881323=795881323, 183885
2020-06-24 07:33:27,075 [datatransfer] INFO  [DB.RingoDB] - TrunkMap-396654193 822816550
2020-06-24 07:33:27,078 [datatransfer] INFO  [DB.RingoDB] - {66=66, 67=67, 69=69, 48=48, 27=27, 1535417895=1535417895,
```

新加入的 Worker3 的 LOG

```
2020-06-24 07:33:26,913 [SOFA-SEV-BOLT-BIZ-12301-3-T4] INFO  [Worker] - 212.64.64.185:12301:822816550 1302869320
2020-06-24 07:33:26,913 [SOFA-SEV-BOLT-BIZ-12301-3-T4] INFO  [Worker] - registerDataTransferService PORT212.64.64.185:12301
2020-06-24 07:33:26,921 [SOFA-SEV-BOLT-BIZ-12301-3-T4] INFO  [Worker] - 212.64.64.185:12301 resgistered data transfer Service
2020-06-24 07:33:26,994 [SOFA-SEV-BOLT-BIZ-12301-3-T5] INFO  [Worker] - DoTransfer{35=35, 17=17, 62=62, 30=30, 766146214=766146214, 32=32}
2020-06-24 07:33:26,998 [SOFA-SEV-BOLT-BIZ-12301-3-T5] INFO  [DB.RingoDB] - {35=35, 17=17, 62=62, 30=30, 766146214=766146214, 32=32}
```

## 服务受到影响的情况举例

Primary 的 LOG 如下

ScaleOut 的第二阶段，出现 P UT 落在被分片的 worker 的 key range 内，所以该 PUT 操作不能正常完成。

```
2020-06-24 19:47:17,748 [processWorkerChange0] INFO  [Primary] - 1 new workers
2020-06-24 19:47:17,763 [scaleOut0] INFO  [Primary] - ScaleOut triggered
2020-06-24 19:47:17,797 [scaleOut0] INFO  [Primary] - 822816550 >= -396654193 and < 1302869320
2020-06-24 19:47:17,797 [scaleOut0] INFO  [Primary] - workerReiverService.SetKeyRange
2020-06-24 19:47:18,143 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - 101219426 >= -396654193 and < 1302869320
2020-06-24 19:47:18,143 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - ASSIGN PUT 1535417895:1535417895 TO 212.64.64.185:12202
2020-06-24 19:47:18,143 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] -  get lock of 1535417895
2020-06-24 19:47:18,149 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] -  keyRWLockMap.remove(key)
2020-06-24 19:47:19,154 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - -687383825 >= 1302869320 or < -396654193
2020-06-24 19:47:19,154 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - ASSIGN PUT 881190480:881190480 TO 212.64.64.185:12201
2020-06-24 19:47:19,154 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] -  get lock of 881190480
2020-06-24 19:47:19,158 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] -  keyRWLockMap.remove(key)
2020-06-24 19:47:19,568 [scaleOut0] INFO  [Primary] - workerReiverService.SetKeyRange
2020-06-24 19:47:20,092 [scaleOut0] INFO  [Primary] - workerReiverService.SetKeyRange
2020-06-24 19:47:20,101 [scaleOut0] INFO  [Primary] - resetKeyRange 212.64.64.185:12202 TO 822816550
2020-06-24 19:47:20,114 [scaleOut0] INFO  [Primary] - resetKeyRange WAIT
2020-06-24 19:47:20,114 [scaleOut0] INFO  [Primary] - wait for 212.64.64.185:12202 to notify datatransfer finish
2020-06-24 19:47:20,165 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - 539353067 >= -396654193 and < 1302869320
2020-06-24 19:47:20,165 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - ASSIGN PUT 616879486:616879486 TO 212.64.64.185:12202
2020-06-24 19:47:20,165 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - 212.64.64.185:12202can not put
2020-06-24 19:47:20,220 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - GET notifyTransferFinish
2020-06-24 19:47:20,221 [scaleOut0] INFO  [Primary] - add 212.64.64.185:12301  workerkeymap and workerState
2020-06-24 19:47:20,221 [scaleOut0] INFO  [Primary] - ScaleOut 212.64.64.185:12301 Finished
```

相同时间 Client 的输出为"the service is not available right now." 符合预期。

```
2020-06-24 19:47:20,165 [SOFA-SEV-BOLT-BIZ-
the service is not available right now.
```

### Consistency 与 Availability 的 trade off

Scale Out 时有两种策略。
一种是牺牲 Consistency 保证 Availability，也就是仅仅拷贝某个时间点的数据到新加入的 worker，期间依旧支持写操作，这样做的优点是服务高可用，缺点是新加入的 worker 的数据与原有的 worker 的数据不一致。
本次 lab 选择了牺牲写操作的 Availability 保证 Consistency 和读操作的 Availability。
优点是数据不会丢失，缺点是当 DataTransfer 很慢时，写操作的 Availability 大幅降低。

## Concurrency Control 的设计与实现

### Primary 的 Concurrency Control

Concurrency Control 的目的是让先到达 Primary 的写请求必然先写入 DB 里，而不会出现因为 Primary->Worker 的网络传输的先后使得先到达 Primary 的写请求反而后写入到 DB 里，这与 RingoDB 的存储结构是否线程安全无关。
Primary 的 Concurrency Control 使用 ReentrantReadWriteLock 来对向同一个 KEY 的读写操作分别加锁，好处是多个读锁不互斥，读锁与写锁互斥，写锁与写锁互斥。
将这个 KEY->LOCK 的映射用哈希表保存起来，这样读写同一个 KEY 的线程都会竞争同一把锁。
如果没有线程在等待这把锁那就将这个 KEY->LOCK 的映射从哈希表中删除，这样做的好处是用不到的锁可以被 JVM 垃圾回收。

### 运行情况

以下为 Client 并发写相同 KEY 时 Primary 的 LOG
表明有线程在等待这把锁

```
2020-06-24 07:34:50,768 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - 1330809503 >= 1302869320 or < -396654193
2020-06-24 07:34:50,768 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - ASSIGN PUT 0:0 TO 212.64.64.185:12201
2020-06-24 07:34:50,768 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - get lock of 0
2020-06-24 07:34:50,770 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - 1330809503 >= 1302869320 or < -396654193
2020-06-24 07:34:50,770 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - ASSIGN PUT 0:0 TO 212.64.64.185:12201
2020-06-24 07:34:50,775 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - hasQueuedThreads
2020-06-24 07:34:50,775 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - get lock of 0
2020-06-24 07:34:50,777 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - 1330809503 >= 1302869320 or < -396654193
2020-06-24 07:34:50,778 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - ASSIGN PUT 0:0 TO 212.64.64.185:12201
2020-06-24 07:34:50,785 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - hasQueuedThreads
2020-06-24 07:34:50,785 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - get lock of 0
2020-06-24 07:34:50,792 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - 1330809503 >= 1302869320 or < -396654193
2020-06-24 07:34:50,793 [SOFA-SEV-BOLT-BIZ-12200-8-T18] INFO  [Primary] - ASSIGN PUT 0:0 TO 212.64.64.185:12201
2020-06-24 07:34:50,801 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - hasQueuedThreads
```

以下为 Client 写不同 KEY 时 Primary 的 LOG
表明如果没有线程在等待这把锁那就将这个 KEY->LOCK 的映射从哈希表中删除

```
2020-06-24 07:34:32,821 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  get lock of 1576910733
2020-06-24 07:34:32,824 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  keyRWLockMap.remove(key)
2020-06-24 07:34:33,830 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - 2118356421 >= 1302869320 or < -396654193
2020-06-24 07:34:33,830 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - ASSIGN PUT 2097540862:2097540862 TO 212.64.64.185:12201
2020-06-24 07:34:33,830 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  get lock of 2097540862
2020-06-24 07:34:33,833 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  keyRWLockMap.remove(key)
2020-06-24 07:34:34,838 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - 134112156 >= -396654193 and < 822816550
2020-06-24 07:34:34,838 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] - ASSIGN PUT 150764808:150764808 TO 212.64.64.185:12202
2020-06-24 07:34:34,838 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  get lock of 150764808
2020-06-24 07:34:34,844 [SOFA-SEV-BOLT-BIZ-12200-8-T20] INFO  [Primary] -  keyRWLockMap.remove(key)
```

### Worker 与 standby data node 的 Concurrency Control

后来的写操作的异步复制线程有可能先被 JVM 调度线程调度，所以把 CopyToStandBy 线程对象都加进 BlockingQueue，再取出一一运行，这样保证了 CopyToStandBy 线程的顺序运行,从而保证了异步复制的数据与 primary data node 的数据的一致。

### 运行情况

如下为 Client 并发写相同 KEY 时 Worker1 的 LOG

```
2020-06-24 12:40:11,984 [SOFA-SEV-BOLT-BIZ-12201-3-T19] INFO  [Worker] - [DB EXECUTION]put0:0
2020-06-24 12:40:11,985 [CopyToStandBy put 0-119] INFO  [Worker] - ready to send 212.64.64.185:12402
2020-06-24 12:40:11,999 [SOFA-SEV-BOLT-BIZ-12201-3-T19] INFO  [Worker] - [DB EXECUTION]put0:0
2020-06-24 12:40:12,003 [SOFA-SEV-BOLT-BIZ-12201-3-T18] INFO  [Worker] - [DB EXECUTION]put0:0
2020-06-24 12:40:12,006 [CopyToStandBy put 0-119] INFO  [Worker] - 212.64.64.185:12402 OK
2020-06-24 12:40:12,006 [CopyToStandBy put 0-119] INFO  [Worker] - ready to send 212.64.64.185:12401
2020-06-24 12:40:12,019 [CopyToStandBy put 0-119] INFO  [Worker] - 212.64.64.185:12401 OK
2020-06-24 12:40:12,019 [RunCopyToStandBy] INFO  [Worker] - [RunCopyToStandBy]CopyToStandBy put 0-120
2020-06-24 12:40:12,028 [CopyToStandBy put 0-120] INFO  [Worker] - ready to send 212.64.64.185:12402
2020-06-24 12:40:12,037 [CopyToStandBy put 0-120] INFO  [Worker] - 212.64.64.185:12402 OK
2020-06-24 12:40:12,037 [CopyToStandBy put 0-120] INFO  [Worker] - ready to send 212.64.64.185:12401
2020-06-24 12:40:12,053 [CopyToStandBy put 0-120] INFO  [Worker] - 212.64.64.185:12401 OK
2020-06-24 12:40:12,056 [RunCopyToStandBy] INFO  [Worker] - [RunCopyToStandBy]CopyToStandBy put 0-121
2020-06-24 12:40:12,060 [CopyToStandBy put 0-121] INFO  [Worker] - ready to send 212.64.64.185:12402
2020-06-24 12:40:12,072 [CopyToStandBy put 0-121] INFO  [Worker] - 212.64.64.185:12402 OK
2020-06-24 12:40:12,073 [CopyToStandBy put 0-121] INFO  [Worker] - ready to send 212.64.64.185:12401
2020-06-24 12:40:12,081 [CopyToStandBy put 0-121] INFO  [Worker] - 212.64.64.185:12401 OK
2020-06-24 12:40:12,081 [RunCopyToStandBy] INFO  [Worker] - [RunCopyToStandBy]CopyToStandBy put 0-122
2020-06-24 12:40:12,084 [CopyToStandBy put 0-122] INFO  [Worker] - ready to send 212.64.64.185:12402
2020-06-24 12:40:12,089 [CopyToStandBy put 0-122] INFO  [Worker] - 212.64.64.185:12402 OK
2020-06-24 12:40:12,089 [CopyToStandBy put 0-122] INFO  [Worker] - ready to send 212.64.64.185:12401
2020-06-24 12:40:12,096 [CopyToStandBy put 0-122] INFO  [Worker] - 212.64.64.185:12401 OK
2020-06-24 12:40:12,100 [RunCopyToStandBy] INFO  [Worker] - [RunCopyToStandBy]CopyToStandBy put 0-123
2020-06-24 12:40:12,104 [CopyToStandBy put 0-123] INFO  [Worker] - ready to send 212.64.64.185:12402
2020-06-24 12:40:12,114 [CopyToStandBy put 0-123] INFO  [Worker] - 212.64.64.185:12402 OK
2020-06-24 12:40:12,114 [CopyToStandBy put 0-123] INFO  [Worker] - ready to send 212.64.64.185:12401
2020-06-24 12:40:12,121 [CopyToStandBy put 0-123] INFO  [Worker] - 212.64.64.185:12401 OK
2020-06-24 12:40:12,121 [RunCopyToStandBy] INFO  [Worker] - [RunCopyToStandBy]CopyToStandBy put 0-124
```

# High Availability 的设计与实现

### 主备数据节点的设计

主备数据节点保证了 Availability，某个 worker 节点宕机的话，因为数据在其他结点上有备份，所以 standby datanode 选举成为 primary data node 后就可以继续提供服务，保证了 Availability。

在整个系统初始化时，为了保证服务的高可用与数据的备份，强制要求每个 worker 必须有大于等于 2 个的 standby data node 方可向 Primary 提供服务，在代码上表现为：Primary worker 的 StandBy node 都注册好了之后再等待 SetKeyRange。

standby data node 向 primary data node 注册完毕后，会收到 primary data node 发来的

SetKeyRange，这样保证了 worker 主备节点的 metadata 一致，也保证了 standby data node 获得领导权成为 primary data node 时，依旧支持 Scale Out 等操作。
standby data node 一边监听结点状态，一边接收 primary data node 发来的数据复制 PUT/DELETE 请求。

判断 Worker 宕机的条件

Primary 对/workers 的子节点注册 Watcher，如果原本 workerState 保存的 worker 不在子节点里，说明 Worker 宕机。
Standby data node 对/workers/workerip:workerport 注册 Watcher。 如果原本 worker 的 znode 消失，说明 Primary data node 宕机。

Primary 对 Worker Fail 的响应

当 Primary 监听/workers 的子节点的变化时，会运行 processWorkerChange 线程。
processWorkerChange 线程发现有旧的 worker znode 消失时，会把对应的 worker 的状态设成 FAIL。
如果 worker znode 被重新创建，又会触发 Watcher，然后 processWorkerChange 线程会比较每个 worker znode 中保存的 workerip:workerport 数据是否与 Primary 现有的数据一致，不一致说明 Worker 的领导权发生了转移，那么更新 RPC 发送的地址（保存在 workerConsumerConfigHashMap 内），并且更新 worker 的状态为可读可写。

在 worker 的领导权还没发生转移的短暂时间，如果 Client 发来了 PUT/GET/DELETE 落在 FAIL 的 worker 的 key range 内,只能向 Client 返回"the service is not available right now."

Standby data node 对 Woker Fail 的响应

当 Standby data node 监听到 worker1 znode 变化，尝试选举成为 primary data node，成功则做一切和原先的 primary data node 一样的操作。失败则向新的 primary data node 注册自己为 standby data node。

非公平选举

本次 lab 在 Worker 宕机后使用 Zookeeper 实现选举，且是非公平模式选举，也就是说最后会选到哪一个机器，是完全随机的（看谁抢的快），与某个节点是不是先来的，是不是等了很久无关。

运行情况

Primary 共监听到/workers 的子节点的变化两次。
第一次设置 worker1 为 FAIL
第二次设置 worker1 的地址为 Worker1-standby1 的地址

```
2020-06-24 19:50:39,388 [primary-EventThread] INFO  [Primary] - workersChangeWatcher triggered
2020-06-24 19:50:39,388 [primary-EventThread] INFO  [Primary] - register workersChangeWatcher
2020-06-24 19:50:39,399 [primary-EventThread] INFO  [Primary] - Successfully got a list of workers:2workers
2020-06-24 19:50:39,399 [primary-EventThread] INFO  [Primary] - [212.64.64.185:12202, 212.64.64.185:12301]
2020-06-24 19:50:39,411 [processWorkerChange1] INFO  [Primary] - [212.64.64.185:12202, 212.64.64.185:12301]
2020-06-24 19:50:39,419 [primary-EventThread] INFO  [Primary] - workersChangeWatcher triggered
2020-06-24 19:50:39,420 [primary-EventThread] INFO  [Primary] - register workersChangeWatcher
2020-06-24 19:50:39,432 [processWorkerChange1] INFO  [Primary] - 212.64.64.185:12202
2020-06-24 19:50:39,432 [processWorkerChange1] INFO  [Primary] - 212.64.64.185:12202
2020-06-24 19:50:39,456 [processWorkerChange1] INFO  [Primary] - 212.64.64.185:12301
2020-06-24 19:50:39,456 [processWorkerChange1] INFO  [Primary] - 212.64.64.185:12301
2020-06-24 19:50:39,456 [processWorkerChange1] WARN  [Primary] - 1worker failed
2020-06-24 19:50:39,461 [primary-EventThread] INFO  [Primary] - Successfully got a list of workers:3workers
2020-06-24 19:50:39,461 [primary-EventThread] INFO  [Primary] - [212.64.64.185:12202, 212.64.64.185:12301, 212.64.64.185:12201]
2020-06-24 19:50:39,474 [processWorkerChange2] INFO  [Primary] - [212.64.64.185:12202, 212.64.64.185:12301, 212.64.64.185:12201]
2020-06-24 19:50:39,496 [processWorkerChange2] INFO  [Primary] - 212.64.64.185:12202
2020-06-24 19:50:39,496 [processWorkerChange2] INFO  [Primary] - 212.64.64.185:12202
2020-06-24 19:50:39,520 [processWorkerChange2] INFO  [Primary] - 212.64.64.185:12301
2020-06-24 19:50:39,520 [processWorkerChange2] INFO  [Primary] - 212.64.64.185:12301
2020-06-24 19:50:39,544 [processWorkerChange2] INFO  [Primary] - 212.64.64.185:12201
2020-06-24 19:50:39,544 [processWorkerChange2] INFO  [Primary] - 212.64.64.185:12402
2020-06-24 19:50:39,545 [processWorkerChange2] INFO  [Primary] - set ConsumerConfig  212.64.64.185:12201->212.64.64.185:12402
2020-06-24 19:50:40,273 [SOFA-SEV-BOLT-BIZ-12200-8-T15] INFO  [Primary] - -1650556404 >= 1302869320 or < -396654193
```

Worker1-standby1 监听到 worker1-primary znode 变化，尝试选举成为 primary data node 并失败，于是向 Worker1-standby2 注册自己为 standby data node

```
2020-06-24 19:50:39,389 [main-EventThread] INFO  [Worker] - workerExistsWatcher/workers/212.64.64.185:12201
2020-06-24 19:50:39,407 [main-EventThread] INFO  [Worker] - 212.64.64.185:12401 is running for PrimaryDataNode
2020-06-24 19:50:39,425 [main-EventThread] INFO  [Worker] - NodeExistsException,check if it's self
2020-06-24 19:50:39,473 [main-EventThread] INFO  [Worker] - RegisterAsStandBy OK
2020-06-24 19:50:39,473 [main-EventThread] INFO  [Worker] - workerExistsWatcher/workers/212.64.64.185:12201
2020-06-24 19:50:40,327 [SOFA-SEV-BOLT-BIZ-12401-3-T20] INFO  [Worker] - [DB EXECUTION]put477202365:477202365
2020-06-24 19:50:41,314 [SOFA-SEV-BOLT-BIZ-12401-3-T20] INFO  [Worker] - [DB EXECUTION]put1173764785:1173764785
2020-06-24 19:50:45,360 [SOFA-SEV-BOLT-BIZ-12401-3-T20] INFO  [Worker] - [DB EXECUTION]put805054408:805054408
```

Worker1-standby2 监听到 worker1-primary znode 变化，尝试选举成为 primary data node 并成功，接收 Worker1-standby1 的注册请求，并把写操作异步复制给 Worker1-standby1。

```
2020-06-24 19:50:39,389 [main-EventThread] INFO  [Worker] - workerExistsWatcher/workers/212.64.64.185:12201
2020-06-24 19:50:39,403 [main-EventThread] INFO  [Worker] - 212.64.64.185:12402 is running for PrimaryDataNode
2020-06-24 19:50:39,419 [main-EventThread] INFO  [Worker] - 212.64.64.185:12402 is PrimaryDataNode
2020-06-24 19:50:39,419 [main-EventThread] INFO  [Worker] - workerExistsWatcher/workers/212.64.64.185:12201
2020-06-24 19:50:39,466 [SOFA-SEV-BOLT-BIZ-12402-3-T20] INFO  [Worker] - StandBySet.add 212.64.64.185:12401
2020-06-24 19:50:40,290 [SOFA-SEV-BOLT-BIZ-12402-3-T20] INFO  [Worker] - [DB EXECUTION]put477202365:477202365
2020-06-24 19:50:40,292 [RunCopyToStandBy] INFO  [Worker] - [RunCopyToStandBy]CopyToStandBy put 477202365-0
2020-06-24 19:50:40,304 [CopyToStandBy put 477202365-0] INFO  [Worker] - ready to send 212.64.64.185:12401
2020-06-24 19:50:40,335 [CopyToStandBy put 477202365-0] INFO  [Worker] - 212.64.64.185:12401 OK
2020-06-24 19:50:41,304 [SOFA-SEV-BOLT-BIZ-12402-3-T20] INFO  [Worker] - [DB EXECUTION]put1173764785:1173764785
```

Worker1-standby2 接收恢复后的原 Worker1-primary 的注册请求

```
2020-06-24 19:50:45,788 [SOFA-SEV-BOLT-BIZ-12402-3-T20] INFO  [Worker] - StandBySet.add 212.64.64.185:12201
```

恢复后的原 Worker1-primary 向 Worker1-standby2 注册自己为 standby data node，并接收新的 primary data node 发来的写复制。

```
2020-06-24 19:50:45,121 [main] INFO  [Worker] - [DoRecover]register worker watcher
2020-06-24 19:50:45,136 [main] INFO  [Worker] - [DoRecover]checkPrimaryDataNode
2020-06-24 19:50:45,839 [main] INFO  [Worker] - RegisterAsStandBy OK
2020-06-24 19:50:53,474 [SOFA-SEV-BOLT-BIZ-12201-3-T1] INFO  [Worker] - [DB EXECUTION]put1650715247:1650715247
```

## 数据备份与恢复

### 异步复制（asynchronous replication）

本次 lab 的系统追求最终一致性，读写都只请求 primary data node，当一条写请求在对应的 primary data node 写成功后，会立刻返回给 Primary 成功，然后 primary data node 通过异步的方式将新的数据同步到对应的 standby data node，这样的方式减少了写入 standby data node 多个节点写成功等待的时间，不过在某些情况下会造成写丢失：

当 primary data node 接受一条写请求，写入并返回给 Primary 成功后不幸宕掉，此时刚才的写还未同步给其对应的 standby data node，而 standby data node 在选举成为新的 primary data node 之后，新的 primary data node 则永久丢失了之前老的 primary data node 向 Primary 确认的写。

### 持久化

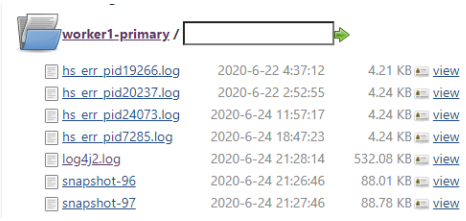
RingoDB 的 snapshot 接口用于持久化当前数据，实现思路是将 ConcurrentHashMap 的拷贝序列化并写入文件。
类似于 Redis 的 RDB 持久化的异步 bgsave 模式。
全量、异步的持久化的设计的好处是不会阻塞数据库服务，缺点是不能保证数据一致性，数据量大时持久化所需时间也较长。

### 从持久化数据恢复

RingoDB 的 recover 接口用于恢复持久化了的数据，实现思路是找到的时间最近的 snapshot 文件，反序列化文件内容并恢复 ConcurrentHashMap。
本次 lab 恢复数据选用了通过 snapshot 全量恢复的机制，好处是恢复数据的速度较快，缺点是不能保证数据一致性。

### 运行情况

如下图为 Worker1-primary 路径下的 snapshot

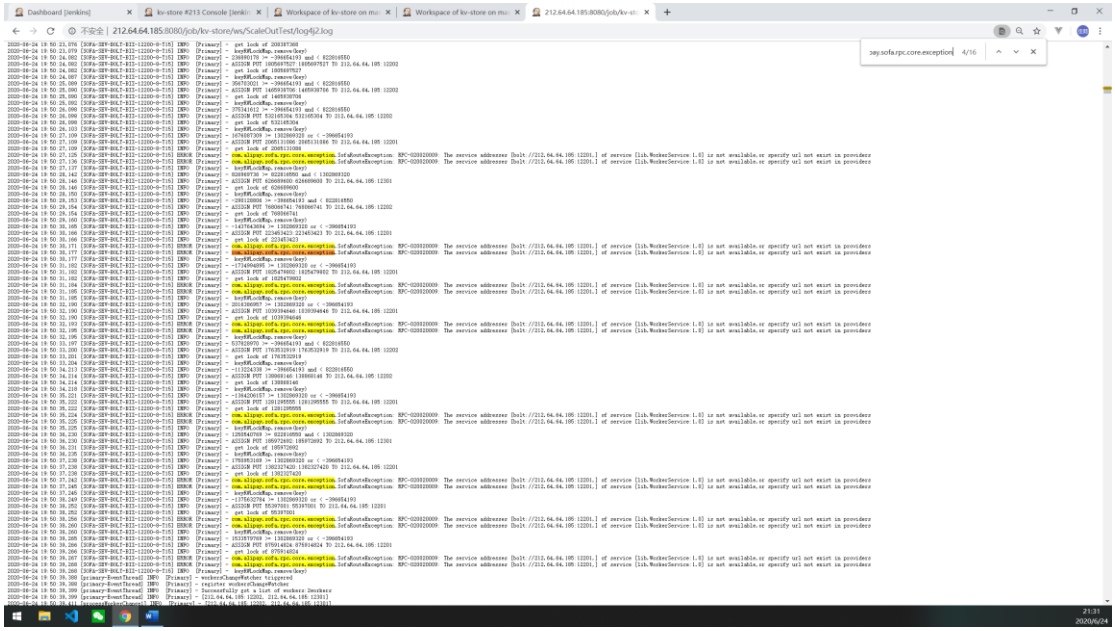



如下图，Worker1 在 2020-06-24 19:50:21,068 写了宕机前最后的 LOG,然后在 2020-06-24 19:50:44,135 恢复。
这与 testWorkerFail.sh 当中 sleep 20s 的命令是相对应的

```
2020-06-24 19:50:21,068 [CopyToStandBy put 250275288-228] INFO  [Worker] - 212.64.64.185:12401 OK
2020-06-24 19:50:44,135 [main] INFO  [Worker] - WORKER METADATA: [primaryNodeAddr] 212.64.64.185:12201 [realAddress] 212.64.64.185:12
2020-06-24 19:50:44,136 [main] INFO  [Worker] - is gonna recover
2020-06-24 19:50:44,151 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:zookeeper.version=3.4.6-1569965, built on
2020-06-24 19:50:44,151 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:host.name=localhost.localdomain
2020-06-24 19:50:44,151 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:java.version=1.8.0_252
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:java.vendor=Private Build
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:java.home=/usr/lib/jvm/java-8-openjdk-amd6
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:java.class.path=../kv-store/target/kv-stor
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:java.library.path=/usr/java/packages/lib/a
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:java.io.tmpdir=/tmp
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:java.compiler=<NA>
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:os.name=Linux
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:os.arch=amd64
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:os.version=4.15.0-88-generic
2020-06-24 19:50:44,152 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:user.name=jenkins
2020-06-24 19:50:44,153 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:user.home=/var/lib/jenkins
2020-06-24 19:50:44,153 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Client environment:user.dir=/var/lib/jenkins/workspace/kv-sto
2020-06-24 19:50:44,154 [main] INFO  [org.apache.zookeeper.ZooKeeper] - Initiating client connection, connectString=112.124.23.139:21
2020-06-24 19:50:44,177 [main] INFO  [Worker] - registerRPCServices PORT212.64.64.185:12201
2020-06-24 19:50:44,191 [main-SendThread(112.124.23.139:2183)] INFO  [org.apache.zookeeper.ClientCnxn] - Opening socket connection to
2020-06-24 19:50:44,425 [main-SendThread(112.124.23.139:2183)] INFO  [org.apache.zookeeper.ClientCnxn] - Socket connection establishe
2020-06-24 19:50:44,452 [main-SendThread(112.124.23.139:2183)] INFO  [org.apache.zookeeper.ClientCnxn] - Session establishment comple
2020-06-24 19:50:44,454 [main-EventThread] INFO  [Worker] - WatchedEvent state:SyncConnected type:None path:null,112.124.23.139:2181,
2020-06-24 19:50:45,091 [main] INFO  [Worker] - [DoRecover] START
2020-06-24 19:50:45,091 [main] INFO  [Worker] - [DoRecover]RecoverFromSnapshot
2020-06-24 19:50:45,092 [main] INFO  [DB.RingoDB] -  get_newest_snapshot_name()
2020-06-24 19:50:45,092 [main] INFO  [DB.RingoDB] - /var/lib/jenkins/workspace/kv-store/worker1-primary
2020-06-24 19:50:45,093 [main] INFO  [DB.RingoDB] - [RingoDB]1
2020-06-24 19:50:45,093 [main] INFO  [DB.RingoDB] - [RingoDB]2
2020-06-24 19:50:45,093 [main] INFO  [DB.RingoDB] - snapshot-2
2020-06-24 19:50:45,121 [main] INFO  [DB.RingoDB] - {88=88, 89=89, 487655583=487655583, 978079902=978079902, 555582039=555582039, 190
2020-06-24 19:50:45,121 [main] INFO  [Worker] - [DoRecover]register worker watcher
2020-06-24 19:50:45,136 [main] INFO  [Worker] - [DoRecover]checkPrimaryDataNode
2020-06-24 19:50:45,839 [main] INFO  [Worker] - RegisterAsStandBy OK
2020-06-24 19:50:53,474 [SOFA-SEV-BOLT-BIZ-12201-3-T1] INFO  [Worker] - [DB EXECUTION]put1650715247:1650715247
```

通过长时间运行证明系统的高可用

从 *2020-06-24 19:46:59,078* 开始提供服务，直到 *2020-06-24 21:29:15* 依旧在运行，期间仅在 worker1-primary 宕机到 worker1-standby 接管领导权的期间出现了小部分服务不可用，以及在 ScaleOut worker3 的过程中出现了小部分服务不可用，其余时间都能正常提供服务。下图为 Primary 的 LOG



# CAP 的 trade off

提高分区容忍性的办法就是一个数据项复制到多个节点上，那么出现 Primary 和 Primary data node 的分区（此时为了简化分析，暂时默认 Zookeeper 集群也和 Primary data node 发生了分区）之后，Primary 可以向选举成功的 standby data node 读写操作。容忍性就提高

了。

然而，要把数据复制到多个节点，就会带来一致性的问题，就是 Primary data node 与 standby data node 的数据、standby data node 与 standby data node 的数据可能是不一致的。要保证一致，每次写操作就都要等待全部节点写成功，而这等待又会带来可用性的问题。

总的来说就是，数据存在的节点越多，分区容忍性越高，但要复制更新的数据就越多，一致性就越难保证。为了保证一致性，更新所有节点数据所需要的时间就越长，可用性就会降低。

那么本次 lab 通过主备数据节点的异步复制、全量异步持久化与恢复两种方法，使得这一分布式内存数据库在牺牲一部分数据的一致性的情况下，具有高可用性。