分布式系统作业

第4次作业

姓名：郝裕玮

班级：计科1班

学号：18329015

理论题：

1、Leader selection在分布式系统中具有重要的用途，主要用于容错，即当主节点失效后能够从备份节点中选择新的leader，但是新选择的leader需要得到其他节点的认同。主流的leader 选择算法有： Bully、 Ring算法，但不限于这些算法，调研以下软件，简述这些软件所采用的选举算法： Zookeeper、 Redis、 MongoDB、 Cassandra。

1. Zookeeper

Zookeeper的Leader选举存在两个阶段：一是服务器初始化启动时，二是运行过程中Leader服务器发生宕机。

参数如下：

服务器 ID（myid）：编号越大在选举算法中权重越大；

事务 ID（zxid）：值越大说明数据越新，权重越大；

逻辑时钟(epoch-logicalclock)：同一轮投票过程中的逻辑时钟值是相同的，每投完一次，值会增加。

各节点的状态有以下4种：

LOOKING：竞选状态

FOLLOWING：随从状态，同步Leader状态，参与投票

OBSERVING：观察状态，同步Leader状态，但不参与投票

LEADING：领导者状态

对于第一种情况，服务器初始化启动时：

（1）每个Server发出一个投票。投票包含所推举的服务器的myid和zxid，并各自将这个投票发给集群中其他机器。

（2）各服务器接受来自其他各个服务器的投票。集群的每个服务器收到投票后，需要判断该票是否有效。

（3）处理投票。选拔规则如下：

① 优先比较zxid。zxid较大的服务器优先作为Leader。

② 若zxid相同，则比较myid。myid较大的服务器优先作为Leader。

（4）统计投票。每次投票结束后后，服务器都会统计投票信息，判断是否已经有过半的机器接收到相同的投票信息。若已过半，则新的Leader被选出。

（5）修改各服务器状态。新的Leader确认后，每个服务器需要更新自己的状态（FOLLOWING，LEADING）。

对于第二种情况，运行过程中Leader服务器发生宕机时：

（1）非Leader服务器宕机或有新成员加入，仍然正常运行；

（2）若Leader服务器发生宕机，则整个集群需要暂停对外服务。各服务器需要更新自己的状态，其他非OBSERVER的服务器需要将自身状态更新为LOOKING。

（3）进入到新一轮的投票阶段，规则和流程仍与第一种情况相同。

2. Redis

Redis采用哨兵机制进行选举，其中每个集群有master和slave，他们的关系为上下级。

其过程如下：

（1）slave发现自己的master变为FAIL。

（2）slave将自己记录的集群currentEpoch（选举轮次标记）加1，并广播并FAILOVER\_AUTH\_REQUEST信息给集群中其他节点。

（3）其他节点收到该信息，只有master响应，判断请求者的合法性，并发送FAILOVER\_AUTH\_ACK，一个epoch只发送一次ack。

（4）尝试failover的slave收集master返回的FAILOVER\_AUTH\_ACK，当某个slave收到超过半数master的ack后变成新master。

（5）广播消息通知其他集群节点。

3. MongoDB

MongoDB节点之间维护心跳检查，主节点选举由心跳触发。

（1）心跳检查

MongoDB复制集成员会向自己之外的所有成员发送心跳并处理响应信息，因此每个节点都维护着从该节点POV看到的其他所有节点的状态信息。节点根据自己的集群状态信息判断是否需要更换新的Primary。

在实现的时候主要由两个异步的过程分别处理心跳响应和超时，抛开复杂的条件检查，核心逻辑主要包括：

① Secondary节点权重比Primary节点高时，发起替换选举；

② Secondary节点发现集群中没有Primary时，发起选举；

③ Primary节点不能访问到大部分（Majority）成员时主动降级；

降级操作会断开链接，终止用户请求等。

（2）选举发起

发起选举的节点需要首先做一些条件判断，比如节点位于备选节点列表中、POV包含复制集Majority等，真实情况的条件判断更加复杂。然后将自己标记为选举过程中，并发起投票请求。

（3）投票

投票发起者向集群成员发起Elect请求，成员在收到请求后经过会一系列检查，如果通过检查则为发起者投一票。一轮选举中每个成员最多投一票，在PV0中用30秒“选举锁”避免为其他发起者重复投票，这导致了如果新选举的Primary挂掉，可能30秒内不会有新的Primary选举产生；在PV1中通过为投票引入单调递增的Term解决重复投票的问题。

如果投票发起者获得超过半数的投票，则选举通过成为Primary节点，否则重新发起投票。

（4）注意事项

① MongoDB选举需要获得大多数投票才能通过，在一轮选举中两个节点得票相同则重新选举，为避免陷入无限重复选举，MongoDB建议复制集的成员个数为奇数个，当Secondary节点个数为偶数时，可以增加一个Arbiter节点。

② PV0版本中，所有成员都可以投否决票，一个否决票会将得票数减少10000，所以一般可以认为只要有成员反对，则该节点不能成为Primary。PV1版本取消了否决票。

③ 选举过程中，复制集没有主节点，所有成员都是只读状态。

4. Cassandra

Cassandra主要使用了Gossip 算法。

Gossip 算法, 灵感来自办公室八卦, 只要一个人八卦一下, 在有限的时间内所有人都会知道该八卦的信息，这种方式也与病毒传播类似, 因为 Gossip 有众多的别名"闲话算法"、"疫情传播算法"、"病毒感染算法"、"谣言传播(Rumor-Mongering)算法"。

但 Gossip 并不是一个新东西, 之前的泛洪查找、路由算法都归属于这个范畴, 不同的是 Gossip 给这类算法提供了明确的语义、具体实施方法及收敛性证明。

特点:

Gossip 算法又被称为反熵（Anti-Entropy）, 熵是物理学上的一个概念, 代表杂乱无章, 而反熵就是在杂乱无章中寻求一致。

这充分说明了 Gossip 的特点：在一个有界网络中, 每个节点都随机地与其他节点通信, 经过一番杂乱无章的通信，最终所有节点的状态都会达成一致. 每个节点可能知道所有其他节点, 也可能仅知道几个邻居节点，只要这些节可以通过网络连通, 最终他们的状态都是一致的, 当然这也是疫情传播的特点。

要注意到的一点是, 即使有的节点因宕机而重启, 有新节点加入, 但经过一段时间后，这些节点的状态也会与其他节点达成一致, 也就是说, Gossip 天然具有分布式容错的优点。

实验题：

2、可靠多播在分布式系统中具有重要的用途，比如传播选举消息等，可靠多播的实现方式有多种，请从以下软件中选择一种，编译运行，观察是否可以实现可靠多播，并撰写报告。

我选择第4个链接：

<https://github.com/daeyun/reliable-multicast-chat>

具体操作流程如下：

（1）下载链接中的压缩包并解压；

（2）在C:\Users\93508\Desktop\reliable-multicast-chat-master\reliable\_multicast\_chat 目录下打开4个cmd窗口（相当于4个节点）

（3）运行方法为：

python main.py [process ID] [delay time] [drop rate]

所以在4个cmd窗口分别输入以下指令:

python main.py 0 0.1 0.1

python main.py 1 0.1 0.1

python main.py 2 0.1 0.1

python main.py 3 0.1 0.1

运行发现报错AttributeError: module 'socket' has no attribute 'SO\_REUSEPORT'

查询资料后可知不同操作系统设置socket，SO\_REUSEPORT选项不同，Windows只能识别SO\_REUSEADDR。

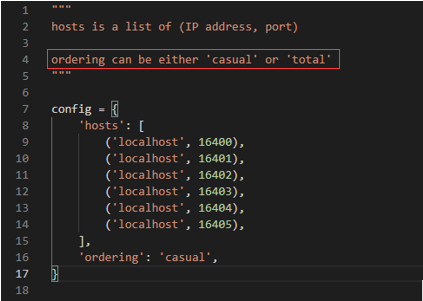
所以将chat\_process.py中的第39行从：

        self.sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEPORT, 1)

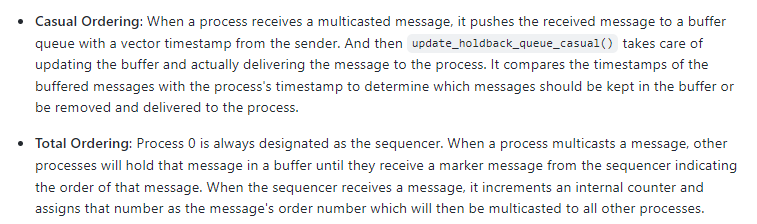
修改为：

        self.sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

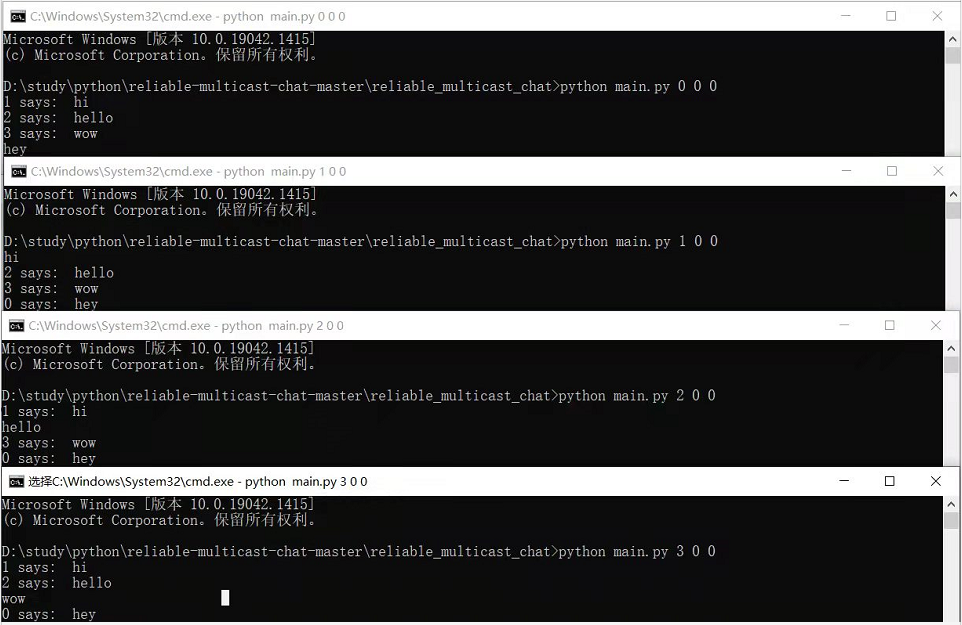
查看config.py可知：



ordering的方式为casual，由注释和github链接可知，还有一种排序方式为total。

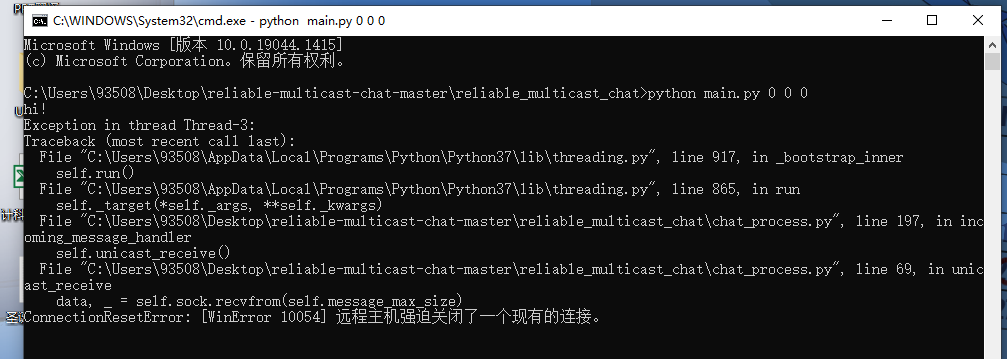


但是在Windows系统运行时结果不稳定，有时候能实现多播（见下页）：



0发送的hey，1发送的hi，2发送的hello，3发送的wow，均被其他三个线程收到。

但有时候也会报错，且频率较高：



原因未知，经同学指导后得知：还是应该将原程序放到ubuntu上运行，而非Windows。

所以将chat\_process.py中的第39行重新从：

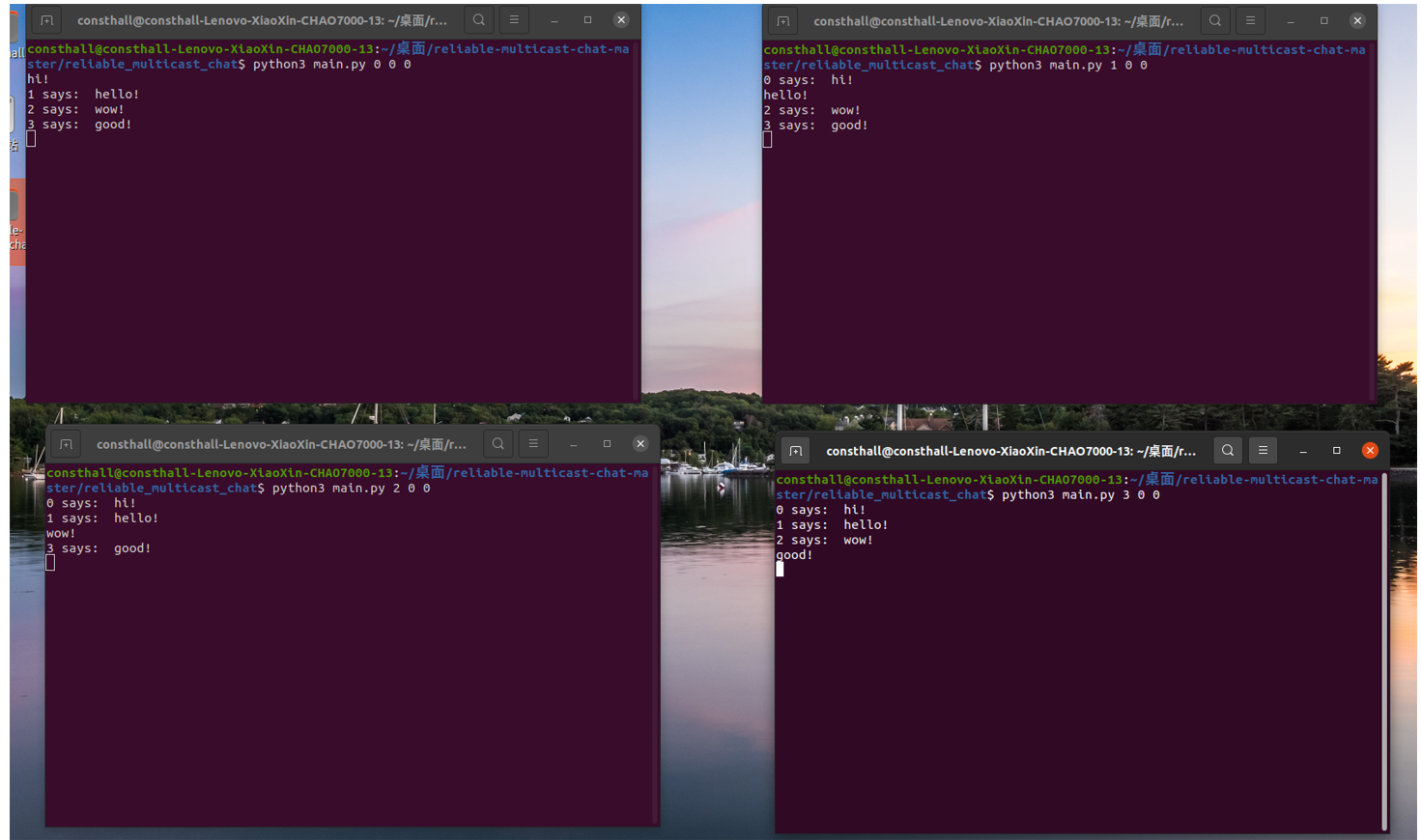
        self.sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

修改为：

        self.sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEPORT, 1)

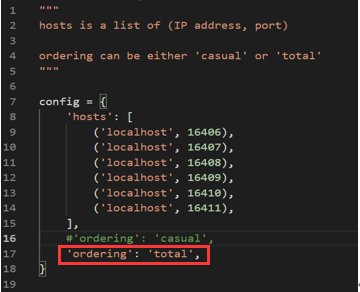
在ubuntu上运行结果为：

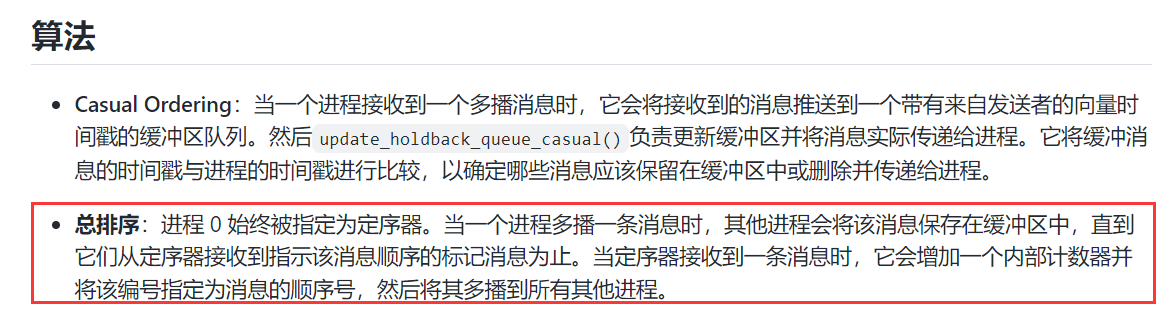
（1）ordering：casual



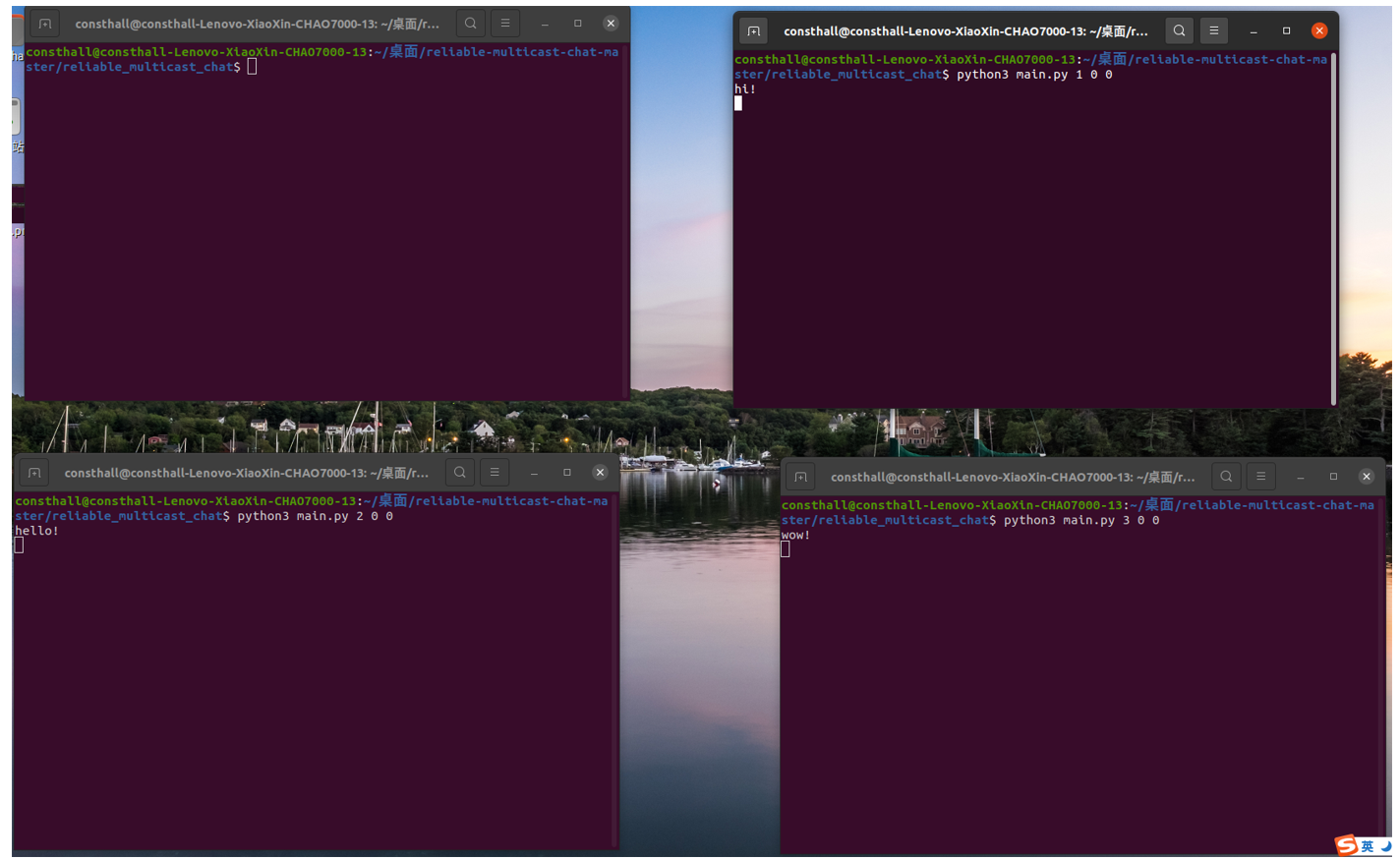
0发送的“hi！”，1发送的“hello！”，2发送的“wow！”，3发送的“good！”，均被其他三个线程收到，所以可靠多播成功。

（2）ordering：total（图见下页）





由上图可知，若为创建0号进程，则其余进程发送的消息都会保存在缓冲区中无法发送给其他进程。



创建0号进程后，发现缓冲区中的消息均已发送出去，可靠多播成功（图见下页）。

