第六次作业

学号：1300013022

姓名：武守北

一、阅读 Paxos 算法的材料并用自己的话简单叙述：

Paxos 算法是一种基于消息传递通信模型的分布式系统，使得各节点就某个值达成一致的问题的算法，其既可以工作在单机的多个进程上面，也可以工作在网络上面的多个主机上面。Paxos 协议假定各个节点之间的通信采用异步的方式，且基于非拜占庭模型，也就是允许消息的延迟、丢失或者重复，但是不会出现内容损坏、篡改的情况，在实践中通过添加额外的校验信息很容易保证收到的消息是完整的。

在 paxos 算法中，分为 4 种角色：

Proposer ：提议者

Acceptor：决策者

Client：产生议题者

Learner：最终决策学习者

Proposer 就像 Client 的使者，由 Proposer 使者拿着 Client 的议题去向 Acceptor 提议，让 Acceptor 来决策。

在 paxos 算法中所有的行为为：

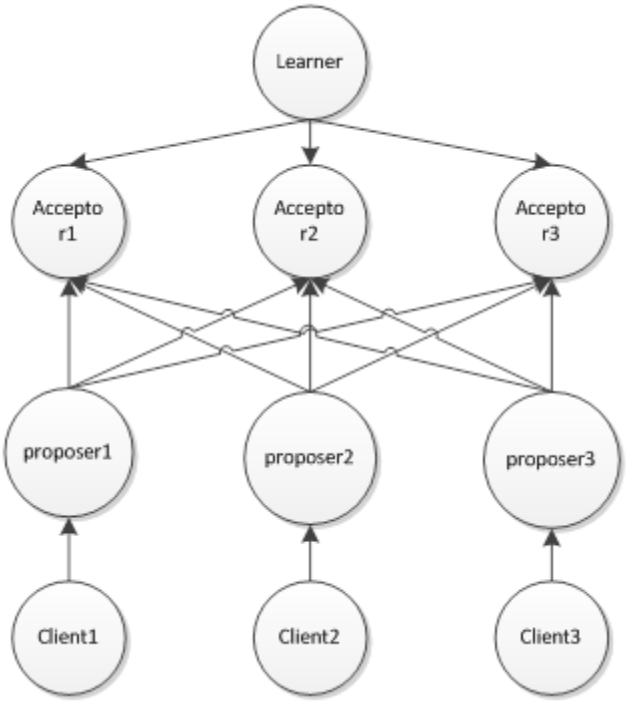
1、Proposer 提出议题；

2、Acceptor 初步接受或者 Acceptor 初步不接受；

3、如果上一步 Acceptor 初步接受则 Proposer 再次向 Acceptor 确认是否最终接受；

4、Acceptor 最终接受或者 Acceptor 最终不接受；

算法流程图如下：



现在以一个简单的例子来解释一下 Paxos 算法的流程（两阶段提交）：

有 2 个 Client（老板、老板之间是竞争关系）和 3 个 Acceptor（政府官员）：

现在需要对一项议题来进行 paxos 过程，议题是“A 项目我要中标！”，这里的“我”指每个带着他的秘书 Proposer 的 Client 老板，故事中的比特币是编号，议题是 value。

一、第一阶段：

Proposer 听老板的话，带着议题和现金去找 Acceptor 政府官员。

作为政府官员，当然想谁给的钱多就把项目给谁。

Proposer-1 小姐带着现金同时找到了 Acceptor-1~Acceptor-3 官员，1 与 2 号官员分别收取了 10 比特币，找到第 3 号官员时，没想到遭到

了 3 号官员的鄙视，3 号官员告诉她，Proposer-2 给了 11 比特币。不过没关系，Proposer-1 已经得到了 1,2 两个官员的认可，形成了多数派（如果没有形成多数派，Proposer-1 会去银行提款在来找官员们给每人 20 比特币，这个过程一直重复每次+10 比特币，直到多数派的形成），满意的找老板复命去了，但是此时 Proposer-2 保镖找到了 1,2 号官员，分别给了他们 11 比特币，1,2 号官员的态度立刻转变，都说 Proposer-2 的老板懂事，这下子 Proposer-2 放心了，搞定了 3 个官员，找老板复命去了，当然这个过程是第一阶段提交，只是官员们初步接受贿赂而已。

这个过程保证了在某一时刻，某一个 proposer 的议题会形成一个多数派进行初步支持。

==========第一阶段结束==========

二、第二阶段：

现在进入第二阶段提交，现在 proposer-1 小姐使用分身术(多线程并发)分了 3 个自己分别去找 3 位官员，最先找到了 1 号官员签合同，遭到了 1 号官员的鄙视，1 号官员告诉他 proposer-2 先生给了他 11 比特币，因为上一条规则的性质 proposer-1 小姐知道 proposer-2 第一阶段在她之后又形成了多数派(至少有 2 位官员的赃款被更新了);此时她赶紧去提款准备重新贿赂这 3 个官员(重新进入第一阶段)，每人 20 比特币。刚给 1 号官员 20 比特币， 1 号官员很高兴初步接受了议

题。在他还没来得及见到 2,3 号官员的时候，proposer-2 先生也使用分身术分别找 3 位官员(注意这里是 proposer-2 的第二阶段)，被第 1 号官员拒绝了告诉他收到了 20 比特币，第 2,3 号官员顺利签了合同，这时 2，3 号官员记录 client-2 老板用了 11 比特币中标，因为形成了多数派，所以最终接受了 Client2 老板中标这个议题，对于 proposer-2 先生已经出色的完成了工作；

这时 proposer-1 小姐找到了 2 号官员,官员告诉她合同已经签了，将合同给她看，proposer-1 小姐是一个没有什么职业操守的聪明人，觉得跟 Client1 老板混没什么前途，所以将自己的议题修改为“Client2 老板中标”，并且给了 2 号官员 20 比特币，这样形成了一个多数派。顺利的再次进入第二阶段。由于此时没有人竞争了，顺利的找 3 位官员签合同，3 位官员看到议题与上次一次的合同是一致的，所以最终接受了，形成了多数派，proposer-1 小姐跳槽到 Client2 老板的公司去了。

Paxos 过程结束了，这样，一致性得到了保证，算法运行到最后所有的 proposer 都投“client2 中标”所有的 acceptor 都接受这个议题，也就是说在最初的第二阶段，议题是先入为主的，谁先占了先机，后面的 proposer 在第一阶段就会学习到这个议题而修改自己本身的议题，因为这样没职业操守，才能让一致性得到保证，这就是 paxos 算法的一个过程。

二、模拟 Raft 协议工作的一个场景并叙述处理过程：

场景：Leader 选举：

Raft 算法是 Paxos 的一个替代品。 场景：Leader Election

1、一开始所有的节点都在等待监听 Leader；

NODE	A	B	C
TERM	0	0	0

2、第一个计时结束的节点，C，要求别的节点给它投票；

NODE	A	B	C
TERM	0	0	1
VOTE COUNT			1

3、其它节点给 C 投票；

NODE	A	B	C
TERM	1	1	1
VOTE	FOR C	FOR C	COUNT: 1

4、C 获得了大多数选票，成为 leader，定期与其它节点联络 62；

NODE	A	B	C
TERM	1	1	1
LEADER	C	C	

5、其它节点接收到联络时，刷新自己的计时器，并向 C 发送心跳信息；

NODE	A	B	C
TERM	1	1	1
LEADER	C	C	

6、如果 C 突然宕机，其它节点会失去联络，定时器不会被重置

NODE	A	B
TERM	1	1
VOTE COUNT	C	C
NODE	A	B
TERM	1	1
VOTE COUNT	C	C

7、B 的定时器先到，成为 candidate，要求其它节点为他投票

NODE	A	B
TERM	1	2
	LEADER C	VOTR COUNT: 1

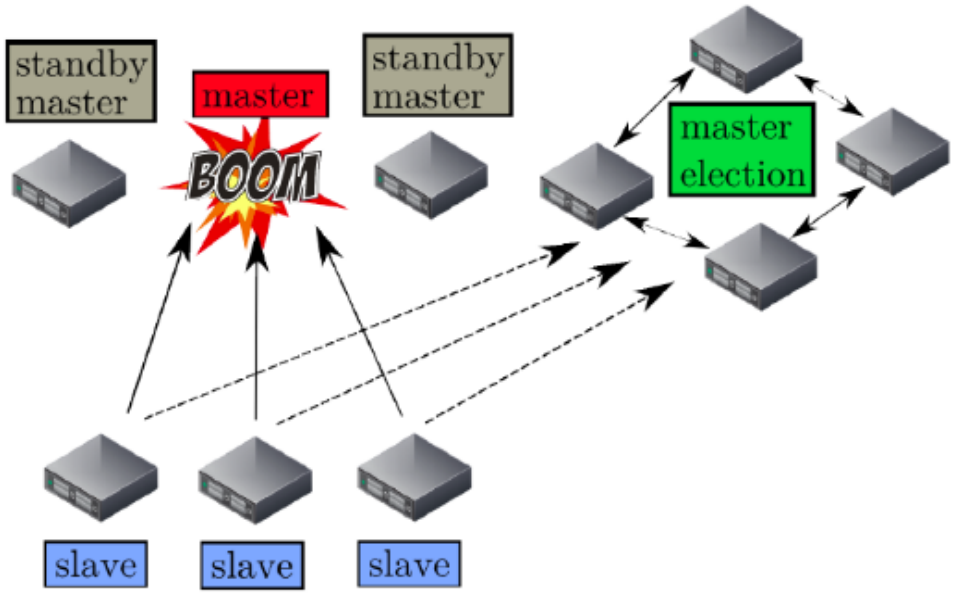
8、B 成功成为了 leader，并向其它节点发送联络

NODE	A	B
TERM	2	2

LEADER	B	

9、如果同时有多个节点成为 candidate，那么可能不会选举出一个 leader，因为没有一个节点获得大多数选票。直到有一个节点成为 leader，心跳联络重新开始

三、简述 Mesos 的容错机制并验证：



mesos 的容错机制体现在四个方面：

master 出错、slave 出错、executor 出错、framework 崩溃

1、master 出错 ： Mesos 使用热备份（hot-standby）设计来实现 Master 节点集合。一个 Master 节点与多个备用（standby）节点运行在同一集群中，并由开源软件 Zookeeper 来监控。Zookeeper 会监控 Master 集群中所有的节点，并在 Master 节点发生故障时管理新 Master 的选举。Mesos 的状态信息实际上驻留在 Framework 调度器和 Slave 节点集合之中。当一个新的 Master 当选后，Zookeeper 会通知

Framework 和选举后的 Slave 节点集合，以便使其在新的 Master 上注册。新的 Master 可以根据 Framework 和 Slave 节点集合发送过来的信息，重建内部状态。

2、slave Mesos 实现了 Slave 的恢复功能，当 Slave 和 master 失去连接时，可以让执行器/任务继续运行。当任务执行时，Slave 会将任务的监测点元数据存入本地磁盘。当 Master 重新连接 Slave，启动 slaver 进程后，因为此时没有可以响应的消息，所以重新启动的 Slave 进程会使用检查点数据来恢复状态，并重新与执行器/任务连接。当 slave 多次无响应，重连接失败，master 会删除这个 slave 节点。

3、executor 出错 当 Slave 节点上的进程失败时，mesos 会通知 framework，让 framework 决定下一步的处理。

4、framework 崩溃 Framework 调度器的容错是通过 Framework 将调度器注册 2 份或者更多份到 Master 来实现。当一个调度器发生故障时，Master 会通知另一个调度来接管。这个需要调度器自己实现。

验证：

1、每台电脑下载一个 zookeeper：

```
wget http://www-eu.apache.org/dist/zookeeper/zookeeper-3.4.9/zookeeper-3.4.9.tar.gz
tar -xvf zookeeper-3.4.9.tar.gz
mv zookeeper-3.4.9.tar.gz zookeepe
```

2、分别配置 zookeeper、创建工作目录并启动：

```
cd zookeeper
cp conf/zoo_sample.cfg conf/zoo.cfg
vim conf/zoo.cfg
dataDir=/var/lib/zookeeper
```

```
server.1=172.16.6.103:2888:3888
server.2=172.16.6.225:2888:3888
server.3=172.16.6.95:2888:3888
mkdir /var/lib/zookeeper
echo "1" > /var/lib/zookeeper/myid
mkdir /var/lib/zookeeper
echo "2" > /var/lib/zookeeper/myid
mkdir /var/lib/zookeeper
echo "3" > /var/lib/zookeeper/myid
bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /home/pkusei/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

3、两台是 follower，一台是 leader：

```
bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /home/pkusei/zookeeper/bin/../conf/zoo.cfg
Mode: follower
bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /home/pkusei/zookeeper/bin/../conf/zoo.cfg
Mode: leader
bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /home/pkusei/zookeeper/bin/../conf/zoo.cfg
Mode: follower
```

4、分别启动 mesos：

```
mesos master --
zk=zk://172.16.6.249:2181,172.16.6.153:2181,172.16.6.2:2181/mesos --
quorum=2 --ip=172.16.6.2  --hostname=mas1 --work_dir=/var/lib/mesos --
log_dir=/var/log/mesos
mesos master --
zk=zk://172.16.6.249:2181,172.16.6.153:2181,172.16.6.2:2181/mesos --
quorum=2 --ip=172.16.6.153 --hostname=mas2 --work_dir=/var/lib/mesos --
log_dir=/var/log/mesos
mesos master --
zk=zk://172.16.6.249:2181,172.16.6.153:2181,172.16.6.2:2181/mesos --
quorum=2 --ip=172.16.6.249 --hostname=mas3 --work_dir=/var/lib/mesos --
log_dir=/var/log/mesos
```

5、Kill 掉 Leader，log 一下：

```
I0529 04:20:10.016351 60160 network.hpp:432] ZooKeeper group memberships
changed
I0529 04:20:10.021173 60160 network.hpp:480] ZooKeeper group PIDs: {  }
I0529 04:20:11.354900 60159 detector.cpp:152] Detected a new leader:
(id='2')
I0529 04:20:11.355798 60159 group.cpp:697] Trying to get
'/mesos/json.info_0000000002' in ZooKeeper
I0529 04:20:11.358307 60159 zookeeper.cpp:259] A new leading master
(UPID=master@172.16.6.153:5050) is detected
I0529 04:20:11.358961 60159 master.cpp:2017] Elected as the leading master!
I0529 04:20:11.359378 60159 master.cpp:1560] Recovering from registrar
I0529 04:20:11.362989 60163 log.cpp:553] Attempting to start the writer
```

可以看到，zookeeper 从备选的 master 中重新选举了一个 leader。

验证完毕。

四、综合作业：

编写一个 mesos framework，使用 calico 容器网络自动搭建一个 docker 容器集群（docker 容器数量不少于三个），并组成 etcd 集群，在 etcd 选举出的 master 节点上部署 jupyter notebook，使得可以从外部访问该集群。同时满足以下条件：

这些 docker 容器共用一套分布式存储

这些 docker 容器可以互相免密码 ssh 登录，且在 host 表中的名字从 XXX-0 一直到 XXX-n（XXX 是自己取的名字，n 是容器数量-1），其中 XXX-0 是 etcd 的 master 节点

当其中一个容器被杀死时，集群仍满足上一个条件

当 etcd master 节点被杀死时，jupyter notebook 会转移到新的 master 节点提供服务，集群仍满足上一个条件。