prevote + fast election

尽可能保证最新数据节点当选

fast election

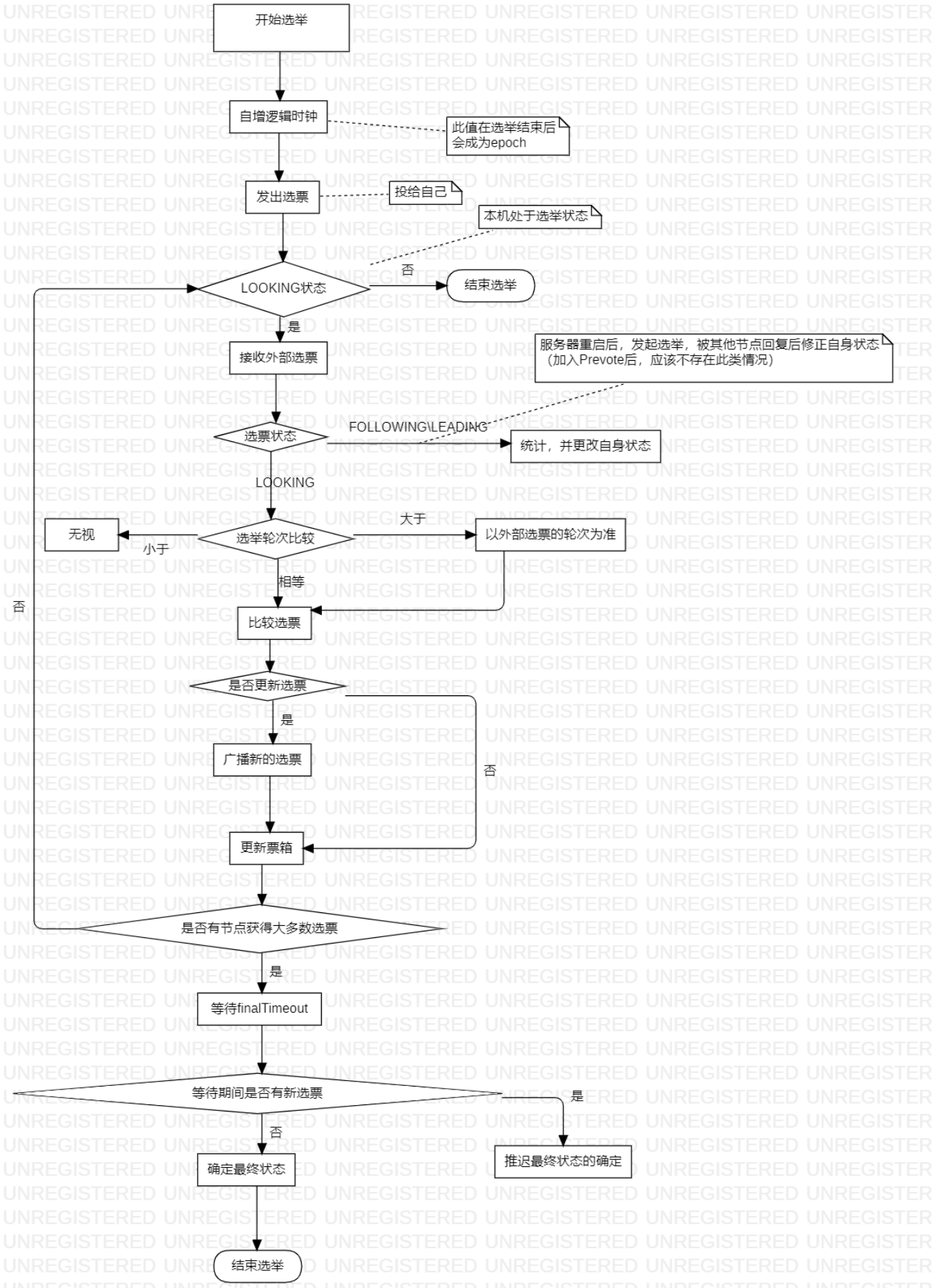
因网络延迟等原因，分布式集群中leader发生宕机时，leader、follower节点间可能存在部分的数据差异，而新leader当选后，各节点的数据均以leader的数据为基准，所以为保证已经提交（或达到大多数同意）的数据不被删除，希望数据最新的节点可以当选成为新的leader，但由于各节点间缺乏全局视角，所以通过网络报文，交换各自数据状态，从而使得节点拥有全局状态信息，最终选出最优节点成为leader。

由于最终目的为选举出最优的节点成为leader，所以将自身数据状态这一概念提炼为选票，即初始时，各节点天然的认为自身即符合最优条件的节点，发送投给自身的选票即等同于告知其他节点本机当前的数据状态，收到此选票后，接收节点通过比较选票中的数据版本信息来进一步决定是否认可其他节点为新leader，即是否改投选票为其他节点；只要出现选票的更改，本节点就需要将此改变广播告知所有的节点，此时如果数据版本信息是严格大于，不存在等于，那么通过各节点间多轮相互广播，最终可选出唯一一台节点作为半数以上认可的新leader。

但是因网络等现实因素的存在，当选节点不一定是最优节点，因为最优的节点可能才刚开始发出选票或者排到了接收方的网络接收队列的后方，还未被处理，这时其他节点的选举已经结束，导致最优节点被漏掉了。所以各节点尽可能同时发起选举，此外在各节点最终确定自身状态前，等待一段时间，此时段内若有新选票到来则推迟确定状态，否则直接确定状态，即尽可能等待所有节点的选票到达。

因leader宕机很可能发生多次，每次宕机后均会发起选举，各次宕机选举之间应该为互不干扰，所以节点需要维护当前的term或epoch，标明当前所在的任期（即哪台节点为leader，而且即使相邻两次选举，选出同一台节点（原leader马上恢复）为leader，这也属于两个term），只有节点处于同一term才能有效的参加选举；另外为保证数据的比较是严格大于的，选票比较时，除比较数据版本信息外，若版本相同则比较选票的服务器id，由此可保证数据必然存在大于。

整体选举流程如下：


开始选举
自增逻辑时钟
此值在选举结束后会成为epoch
发出选票
投给自己
本机处于选举状态
LOOKING状态
否
结束选举
是
接收外部选票
服务器重启后，发起选举，被其他节点回复后修正自身状态（加入Prevote后，应该不存在此类情况）
选票状态
FOLLOWING\LEADING
统计，并更改自身状态
LOOKING
选举轮次比较
大于
以外部选票的轮次为准
无视
小于
相等
否
比较选票
是否更新选票
是
广播新的选票
否
更新票箱
是否有节点获得大多数选票
是
等待finalTimeout
等待期间是否有新选票
是
否
确定最终状态
推迟最终状态的确定
结束选举

prevote

集群冷启动时，由于无法同时启动所有服务器，所以选择在配置文件中人为选定某主机作为leader，根据当前的"小ip连接大ip"的网络特性，建议小ip主机作为初始leader；启动后，leader会发出心跳报文，保持集群连通性，同时维持当前leader的任期，如果出现leader服务器宕机，剩余服务器会检测心跳报文超时次数，超时出现5次以上，即判定与leader断开连接（即election timeout = 5 * 心跳包超时时间）。但由于集群中各服务器缺乏全局视角，各节点仅能判断本机与当前leader断开连接，贸然发起选举，可能出现集群"脑裂"现象；

所以引入类似raft中prevote机制，即在选举之前，各节点维护一张各节点与当前leader的连通性表，即主观下线表（subjectiveFailMap_），同时交换各自与leader之间的连通性。节点收到来自其他节点的主观下线报文后，判断自身与leader连通性，并更新主观下线表，如果表格中半数以上节点与当前leader断开连接，则客观上判定leader已掉线，可以发起新一轮的选举fast election。最先判定leader客观下线的节点将广播告知所有现存活节点：leader已客观下线，可以发起选举，同时自身也主动发起选举。此模式下各节点基本可保证几乎同时发起选举，因最早发起选举的节点最多早RTTs发起选举，此时各节点也已开始选举，杜绝了选票被无视等情况，亦保证了fast election算法中各节点尽可能同时发起选举的前提条件。

但此时也引入了新的问题，由于初始leader由配置文件指定，初始leader宕机后，选出新的leader，此时初始leader重启，会仍然认为自身为leader，从而发出心跳包，所以当前集群中节点会收到错误的心跳信息，错误的重置自身的选举定时器，集群出现两个leader。

为防止此现象发生，节点收到心跳包后，会先行检查心跳包的发出方是否为当前的leader，若来自其他节点（如宕机后重启的初始leader）会发送纠错报文，告知其当前实际leader，错误心跳的发出方会统计此类纠错报文，达到半数以上即选择信任此纠错报文，更新自身状态，成为follower加入集群；不过此机制下，同样会导致错误，如经过一段时间运行，初始leader已转变为follower，此时另一follower宕机重启，在此follower的配置文件中leader应该为初始leader，所以错误的向当前leader发出了纠错报文，此时需要当前leader直接纠正此状态错误，防止follower无法加入现有集群。（但是此报文不可能累计半数以上，因为半数以上发出错误的纠错，就代表着半数以上服务器重启，即集群已经整体宕机，已无法对外提供服务）

纠错报文的机制类似于各节点间发起了仅维持一轮的fast election，且不会多次广播。

新增与修改

1）修改启动时的步骤，不再使用静态的配置leader

取消了所谓的"纠错"机制，启动时各节点会首先读取acceptor中持久化的`paxos id`数据作为后续选票中的报文信息（`LoadPaxosState`），然后发送`check`报文，广播其状态信息（初始为LOOKING），其他节点会相应回复自身状态，当某节点收到的check报文中状态为LOOKING的报文达到半数以上，那么此节点启动选举。此机制也从另一个方向强制要求发起选举前需半数以上节点存在

2）此外，当LOOKING状态的主机直接收到心跳数据时，会直接加入当前集群

以上修改，保证了冷启动下的选举，而prevote保证了leader或含leader在内的少数节点宕机时选举的正确性。

但是仍然存在问题：

问题1：假定场景为集群中半数以上主机宕机重启，若数据较新的节点A最后重启恢复，而其他数据较为陈旧的节点B、C，启动并达成半数以上LOOKING状态，从而发起选举。此时，如果节点A启动很慢，导致错过了B、C节点最终确定状态前的等待时间，那么节点A完全可能错过此次选举，使得B、C中其中一台当选，A节点因为直接收到了心跳，从而加入该leader领导下的集群。

问题2：当前修改下，如果各个节点启动间隔较大，那么满足半数以上要求的那台主机（如5台主机中的第3台）启动时将会发起选举，无法确保数据最新，存在较大隐患；考虑将check报文的发送修改为定时任务或循环任务，当发起选举后停止，但是即使改为循环发送，此时也会遭遇到问题1，所以有没有修改必要呢？

所以，在选出leader，并进入到paxos的正常流程后，仍需验证follower节点发出的paxos id信息，当发现follower中的数据更新，paxos id更大时，重新进入选举。

或者，leader的心跳（或续租）报文中包含paxos id信息，当follower数据更新，重新选举；

但是此时会不会已经晚了，客户端针对当前的paxos id直接返回错误？引导后续重试？