# Paxos Made Simple 推理复述

作者首先提出了 consensus 这个问题：有一系列 processes 可以 propose values，设计一个算法保证在所有 proposed values 中，只有一个最终会被 chosen。

作者然后给出了一个合格的 consensus 算法应该满足的两个条件：

- safety: 只有 proposed values 中的其中一个最终会被 chosen.
- liveness: 在有限时间内，最终一定会有一个 value 被 chosen.

safety 说明了算法的正确性，liveness 说明了算法的有限性。

作者的推理过程是这样的：先提出一些简单直接的算法，论证它们是否满足 safety 和 liveness，包括在 unreliable network 和 fail-stop failure 的情况下。如果不满足，则提出更完善的算法，尝试满足 safety 和 liveness。最终得到一个合格的 consensus 算法，即 paxos。

作者首先提出了第一个算法：

> 算法 1: proposers 只向一个 acceptor propose values，这个唯一的 acceptor 会 accept 它所收到的第一个 proposal。这个唯一被 accept 的 proposal 自然成为被 chosen 的 proposal。

如果没有 failure，显然这个算法满足 safety 和 liveness。但是由于这个唯一的 acceptor 可能会 fail，即 single node failure，则这个算法在有 failure 的情况下并不满足 safety 和 liveness。

作者观察到算法 1 之所以不合格，是因为 single node failure，那么考虑设置多个 acceptors。作者于是提出了第二个算法：

> 算法 2: proposers 向多个 acceptors propose values，每个 acceptor 仅会 accept 它所收到的第一个 proposal。如果一个 proposal 被多数 acceptors 所 accept，则该 proposal 被 chosen。

这个算法既不满足 safety，也不满足 liveness。这是因为每个 acceptor 仅会 accept 它所收到的第一个 proposal，那么可能发生 split votes，则没有多数派的形成，则不会有 value 被 chosen。

safety 更重要，因此作者首先尝试修改算法使其满足 safety。作者发现如果一个 acceptor 仅能 accept 一个 proposal，那么多数派很难形成，故作者允许 acceptors accept 多个 proposals。作者于是提出了第三个算法：

> 算法 3: proposers 向多个 acceptors propose values，每个 acceptor 必须 accept 它所收到的第一个 proposal，且可以 accept 多个 proposals。如果一个 proposal 被多数 acceptors 所 accept，则该 proposal 被 chosen。

很遗憾，safety 依然没有满足。因为 acceptors 可以 accept 多个 proposals，那么可能有多个 proposals 被多数派 accept，则会有多个 values 被 chosen，显然违反了 safety。

那么如何完善算法以满足 safety 呢？作者发现既然一定会有多个 proposals 被 chosen，那么唯一可行的方法就是保证这个不变量：

> 不变量：所有被 chosen 的 proposals 都是同一个 proposal。

好，现在我们知道要实现一个合格的 consensus 算法，唯一的方法就是保证上面这个不变量。但是如何去保证呢？于是，作者之后开始想办法把这个不变量转化成其它等价或更严格的描述，以使其可以写成一个算法。

作者发现一个 proposal 要被 chosen，必须经过 accept 这一步骤。在一个 proposal 被 chosen 之后，如果能保证之后所有 proposal 被 accept 时都是同一个 proposal，那么不就可以保证不变量了吗。作者于是提出了不变量的等价描述：

> 不变量 a：所有被 chosen 的 proposals 在 accept 时都是同一个 proposal。

不变量 a 是无法保证的。回忆算法 3，每个 acceptor 必须 accept 它所收到的第一个 proposal。因为这里的“必须”，显然不变量 a 无法保证。

作者又发现一个 proposal 要被 chosen，不仅要经过 accept，在这之前还必须经过 propose 这一步骤。在一个 proposal 被 chosen 之后，如果能保证所有 proposal 在 propose 时都是同一个 proposal，那么不就可以保证不变量了吗。作者于是提出了不变量 a 的更严格描述（说更严格是因为有 accept 必定有 propose，而有 propose 不一定有 accept）：

> 不变量 b：所有被 chosen 的 proposals 在 propose 时都是同一个 proposal。

这个不变量还是不够具体，作者于是提出了不变量 c：

> 不变量 c：如果一个 proposer 可以 propose，那么这两个条件之一必定满足：（1）之前没有 proposal 被 chosen；（2）这个 proposal 与最近被 chosen 的那个 proposal 为同一个 proposal。

满足了不变量 c 能否满足不变量 b 呢？我们可以使用不变量 c 的条件（2）进行递推。假设最早被 chosen 的 proposal 是第 m 个 proposal。对于第 m + 1 个 proposal，既然它能被 propose 出来，那么其必定与第 m 个 proposal 相同。如此递推，有：得到一个被 chosen 的 proposal 之后，其后的每个 proposal 都与其相同。即所有被 chosen 的 proposals 在 propose 时都是同一个 proposal，即不变量 b。这个推理告诉我们：如果能满足不变量 c，则必定满足不变量 b。

好，现在我们得到了不变量 c，但它还不足以称为算法，因为它并没有告诉我们怎么让一个 proposer 去确认自己满足了两个条件之一。作者提出的算法如下：

> 算法 4：每个 proposer 在 propose 之前去 contact acceptors，当且仅当该 proposal 满足不变量的两个条件之一，才允许其 propose。

费了这么大的劲，我们终于得到了一个看似可行的 consensus 算法，它就是 算法 3 + 算法 4。但是思考一番，发现以上算法还有三个问题需要解决：

1. 在分布式环境中，一个 proposer 无法直接知道哪个 proposal 被 chosen 了，它只能通过多数派的决议来判断。
2. 算法中出现了很多与时序有关的概念，如何表示时序？
3. 如何满足 liveness？

对于问题 1，考虑到一个 proposal 被 chosen，必定被某个多数派所 accept，因此对不变量 c 等价改写，得到不变量 d：

> 不变量 d：如果一个 proposer 可以 propose 一个 value 为 v 的 proposal，那么必须有：存在一个多数派，它们要么（1）在这个

> proposal 之前没有 accept 任何 proposal；（2）在这个 proposal 之前所 accept 的最近的那个 proposal 的 value 也为 v。

对于问题 2，作者提出设计一个 proposal number。这个 proposal number 可以由两部分构成：与 agent 相关的一部分，与时间相关的一部分。这个时间可以是物理时间，例如各个 agent 的 wall time，也可以是逻辑时间，例如 ticker。每个 proposal 不仅包含 value，还包含一个 unique 的 proposal number。所有的时序关系则根据 proposal number 来描述。

这又引出了一个问题。不变量 d 中说“在这个 proposal 之前“是根据 proposal number 来确定，那么在 unreliable network 的情况下，考虑这样一种情况： proposer A 想要 propose 一个 proposal number 为 n 的 value v，它向 proposer B 发送 msg 查询 accepted proposals。此时所有 proposal number < n 的 proposals 分为两组。一组是已经被 proposer B 所 accept 的，另一组 proposals 由于网络延迟等原因本应到达、却尚未到达 propoper B，因此尚未、但可能将要被 accept。在这种情况下，proposer A 可能根据反馈的信息认为自己满足了不变量 d 的两个条件之一，但是实际上不满足，因为反馈的信息仅仅考虑了第一组的 proposals，没有考虑第二组。

为了使得不变量 d 在这种情况下依然满足，作者认为 acceptor 应该保证不再 accept 第二组 proposals。作者于是提出了如下算法：

> 算法 5: 一旦某个 acceptor accept 了 proposal number 为 n 的 proposal，那么它将拒绝 accept 任何 proposal number < n 的 proposals。

对于问题 3，作者提出选出一个 leader，详见论文。

至此，作者推导出了一个合格的 consensus 算法，即 算法3 + 算法4 + 算法5 + 保证 liveness 的算法。

参考：

http://nil.csail.mit.edu/6.824/2015/papers/paxos-simple.pdf

https://www.cs.princeton.edu/courses/archive/fall16/cos418/docs/P3-consensus.pdf