13 | PBFT算法：有人作恶，如何达成共识？

2020-03-11 韩健

分布式协议与算法实战 进入课程 >

讲述：于航
时长 10:19　大小 9.46M

你好，我是韩健。

学完了01 讲的拜占庭将军问题之后，有同学在留言中表达了自己的思考和困惑：口信消息型拜占庭问题之解在实际项目中是如何落地的呢？先给这位同学点个赞，很棒！你能在学习的同时思考落地实战。

不过事实上，它很难在实际项目落地，因为口信消息型拜占庭问题之解是一个非常理论化的算法，没有和实际场景结合，也没有考虑如何在实际场景中落地和实现。

比如，它实现的是在拜占庭错误场景下，忠将们如何在叛徒干扰时，就一致行动达成共识。但是它并不关心结果是什么，这会出现一种情况：现在适合进攻，但将军们达成的最终共识

却是撤退。

很显然，这不是我们想要的结果。因为在实际场景中，我们需要就提议的一系列值（而不是单值），即使在拜占庭错误发生的时候也能被达成共识。那你要怎么做呢？答案就是掌握 PBFT 算法。

PBFT 算法非常实用，是一种能在实际场景中落地的拜占庭容错算法，它在区块链中应用广泛（比如 Hyperledger Sawtooth、Zilliqa）。为了帮助你更好地理解 PBFT 算法，在今天的内容中，我除了带你了解 PBFT 达成共识的原理之外，还会介绍口信消息型拜占庭问题之解的局限。相信学习完本讲内容后，你不仅能理解 PBFT 达成共识的基本原理，还能理解算法背后的演化和改进。

老规矩，在开始今天的学习之前，咱们先看一道思考题：

假设苏秦再一次带队抗秦，这一天，苏秦和 4 个国家的 4 位将军赵、魏、韩、楚商量军机要事，结果刚商量完没多久苏秦就接到了情报，情报上写道：联军中可能存在一个叛徒。这时，苏秦要如何下发作战指令，保证忠将们正确、一致地执行下发的作战指令，而不是被叛徒干扰呢？

带着这个问题，我们正式进入今天的学习。

首先，咱们先来研究一下，为什么口信消息型拜占庭问题之解很难在实际场景中落地，除了我在开篇提到的非常理论化，没有和实际的需求结合之外，还有其他的原因么？

其实，这些问题是后续众多拜占庭容错算法在努力改进和解决的，理解了这些问题，能帮助你更好地理解后来的拜占庭容错算法（包括 PBFT 算法）。

口信消息型拜占庭问题之解的局限

我想说的是，这个算法有个非常致命的缺陷。如果将军数为 n、叛将数为 f，那么算法需要递归协商 f+1 轮，消息复杂度为 O(n ^ (f + 1))，消息数量指数级暴增。你可以想象一下，如果叛将数为 64，消息数已经远远超过 **int64** 所能表示的了，这是无法想象的，肯定不行啊。

另外，尽管对于签名消息，不管叛将数（比如 f）是多少，经过 f + 1 轮的协商，忠将们都能达成一致的作战指令，但是这个算法同样存在“理论化”和“消息数指数级暴增”的痛点。

讲到这儿，你肯定明白为什么这个算法很难在实际场景中落地了。可技术是不断发展的，算法也是在解决实际场景问题中不断改进的。那么 PBFT 算法的原理是什么呢？为什么它能在实际场景中落地呢？

PBFT 是如何达成共识的？

我们先来看看如何通过 PBFT 算法，解决苏秦面临的共识问题。先假设苏秦制定的作战指令是进攻，而楚是叛徒（为了演示方便）：

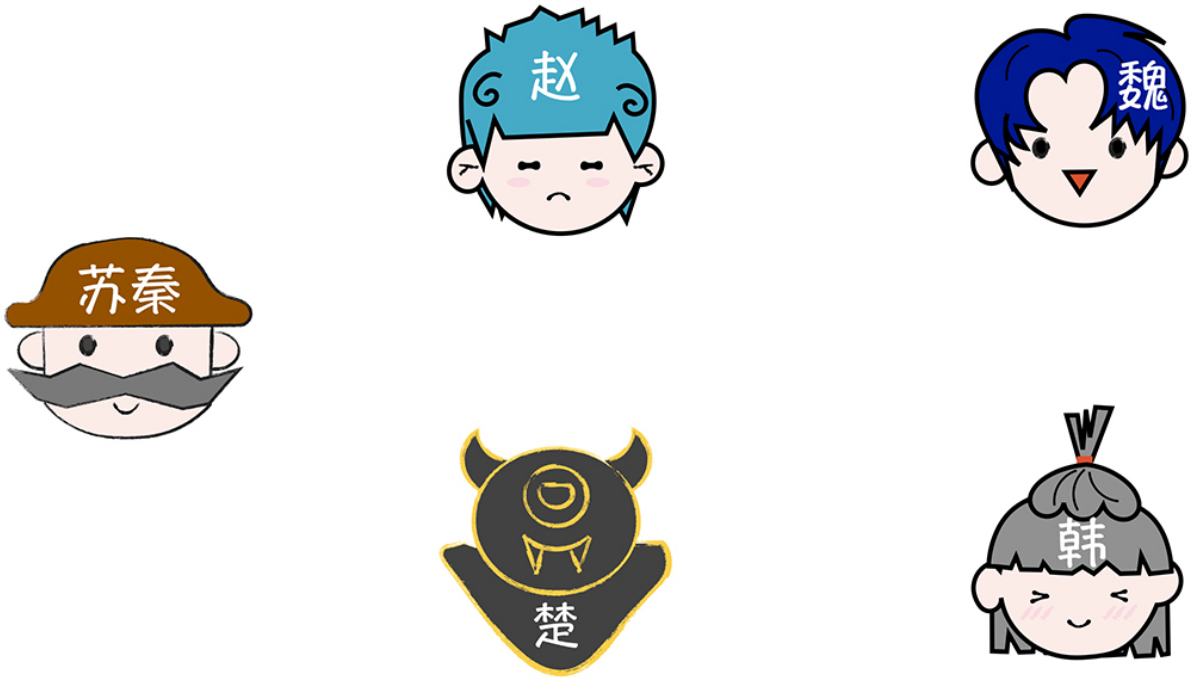

图1

需要你注意的是，所有的消息都是签名消息，也就是说，消息发送者的身份和消息内容都是无法伪造和篡改的（比如，楚无法伪造一个假装来自赵的消息）。

首先，苏秦联系赵，向赵发送包含作战指令“进攻”的请求（就像下图的样子）。

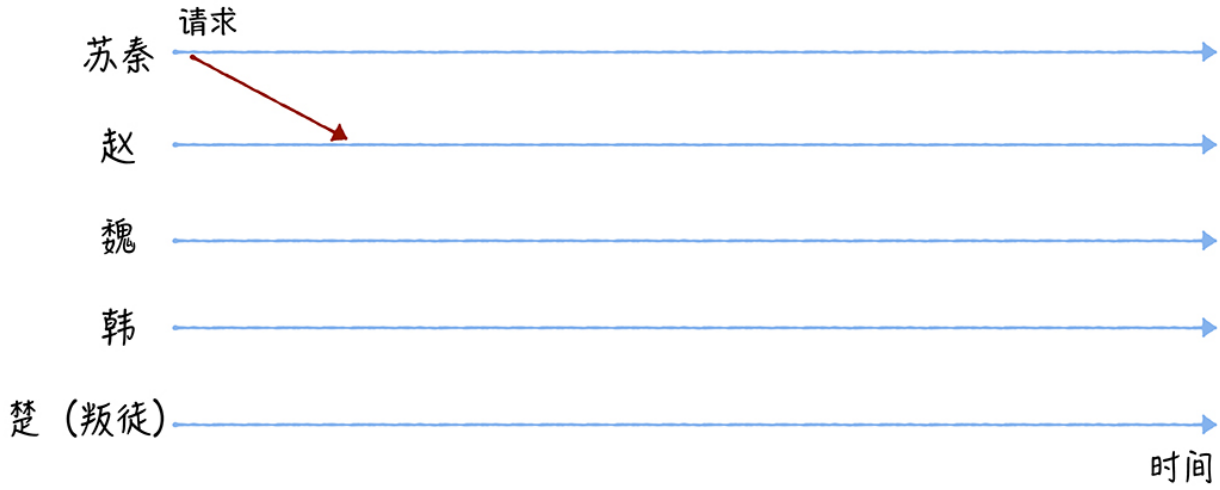

图2

当赵接收到苏秦的请求之后，会执行三阶段协议（Three-phase protocol）。

赵将进入预准备（Pre-prepare）阶段，构造包含作战指令的预准备消息，并广播给其他将军（魏、韩、楚）。

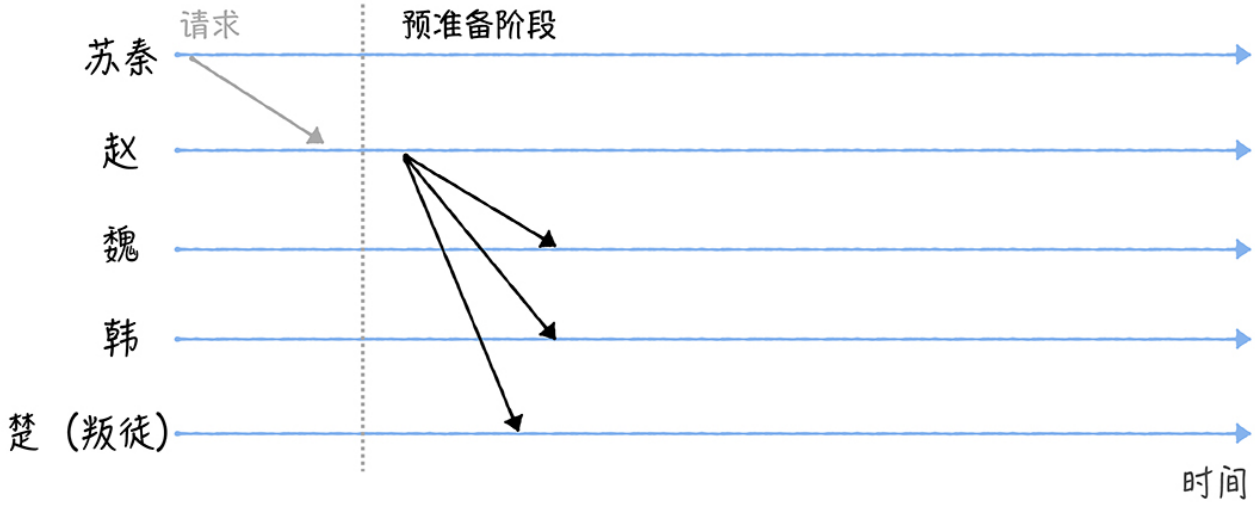

图3

那么在这里，我想问你一个问题：魏、韩、楚，收到消息后，能直接执行指令吗？

答案是不能，因为他们不能确认自己接收到指令和其他人接收到的指令是相同的。比如，赵可能是叛徒，赵收到了 2 个指令，分别是“进攻”和“准备 30 天的粮草”，然后他给魏发送的是“进攻”，给韩、楚发送的是“准备 30 天粮草”，这样就会出现无法一致行动的情况。那么他们具体怎么办呢？我接着说一下。

接收到预准备消息之后，魏、韩、楚将进入准备（Prepare）阶段，并分别广播包含作战指令的准备消息给其他将军。比如，魏广播准备消息给赵、韩、楚（如图所示）。为了方便演示，我们假设叛徒楚想通过不发送消息，来干扰共识协商（你能看到，图中的楚是没有发送消息的）。

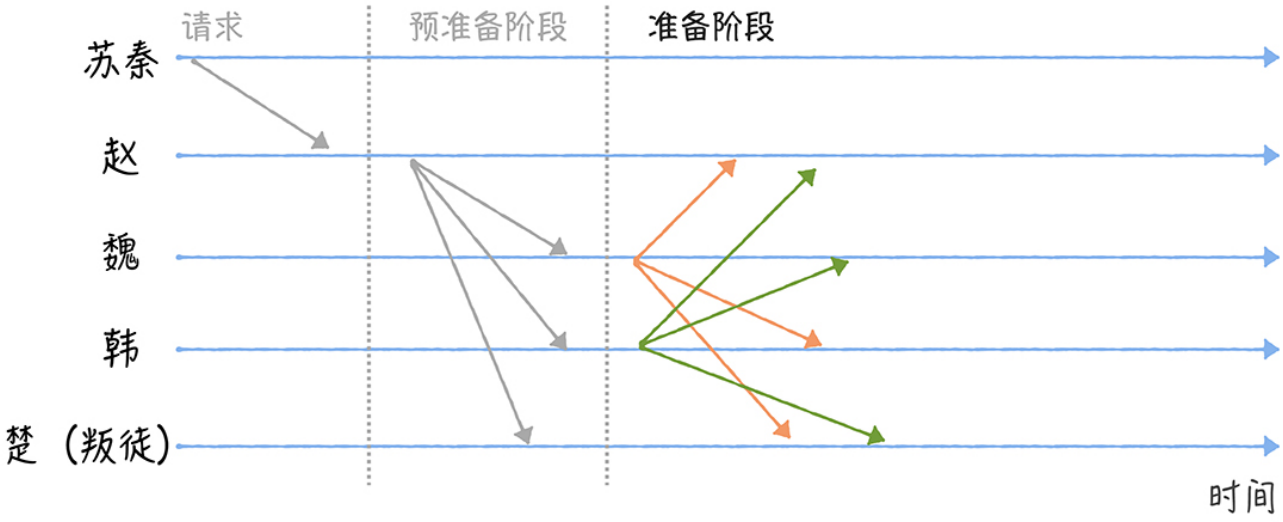

图4

然后，当某个将军收到 2f 个一致的包含作战指令的准备消息后，会进入提交（Commit）阶段（这里的 2f 包括自己，其中 f 为叛徒数，在我的演示中是 1）。在这里，我也给你提一个问题：这个时候该将军（比如魏）可以直接执行指令吗？

答案还是不能，因为魏不能确认赵、韩、楚是否收到了 2f 个一致的包含作战指令的准备消息。也就是说，魏这时无法确认赵、韩、楚是否准备好了执行作战指令。那么怎么办呢？别着急，咱们继续往下看。

进入提交阶段后，各将军分别广播提交消息给其他将军，也就是告诉其他将军，我已经准备好了，可以执行指令了。

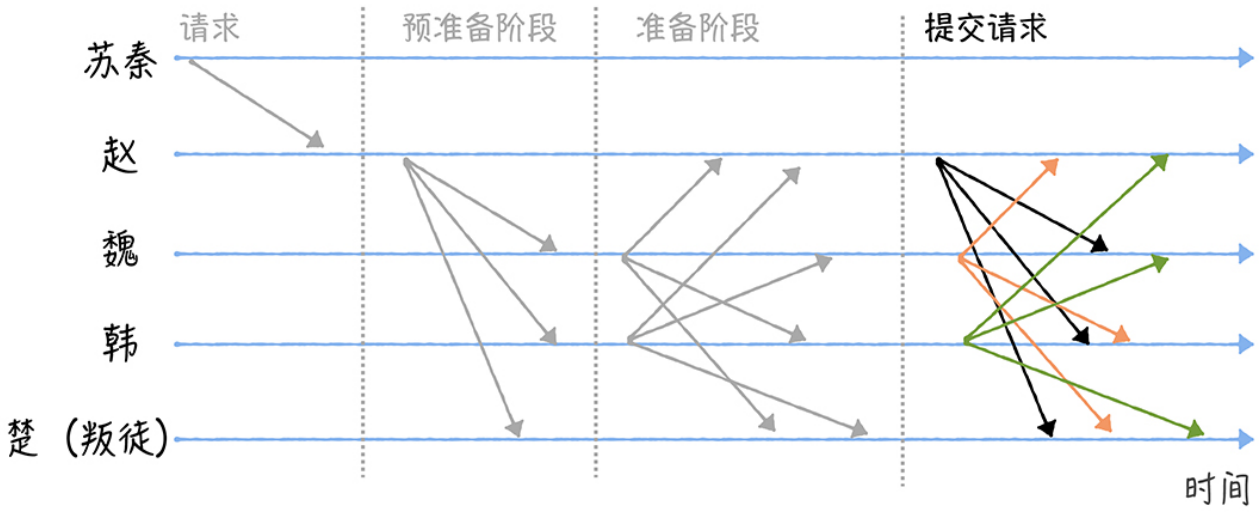

图5

最后，当某个将军收到 2f + 1 个验证通过的提交消息后（包括自己，其中 f 为叛徒数，在我的演示中为 1），也就是说，大部分的将军们已经达成共识，这时可以执行作战指令了，那么该将军将执行苏秦的作战指令，执行完毕后发送执行成功的消息给苏秦。

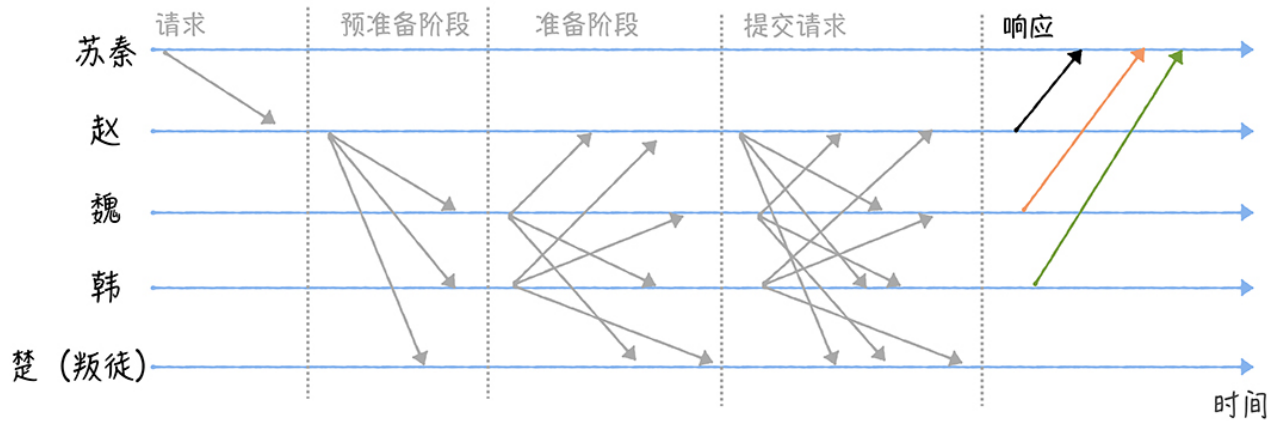

图6

最后，当苏秦收到 f+1 个相同的响应（Reply）消息时，说明各位将军们已经就作战指令达成了共识，并执行了作战指令（其中 f 为叛徒数，在我的演示中为 1）。

你看，经过了三轮协商，是不是就指定的作战指令达成了共识，并执行了作战指令了呢？

在这里，苏秦采用的就是**简化版的 PBFT 算法**。在这个算法中：

- 你可以将赵、魏、韩、楚理解为分布式系统的四个节点，其中赵是主节点（Primary node），魏、韩、楚是从节点（Secondary node）；
- 将苏秦理解为业务，也就是客户端；
- 将消息理解为网络消息；
- 将作战指令“进攻”，理解成客户端提议的值，也就是希望被各节点达成共识，并提交给状态机的值。

在这里我想说的是， PBFT 算法是通过签名（或消息认证码 MAC）约束恶意节点的行为，也就是说，每个节点都可以通过验证消息签名确认消息的发送来源，一个节点无法伪造另外一个节点的消息。最终，基于大多数原则（2f + 1）实现共识的。

需要你注意的是，最终的共识是否达成，客户端是会做判断的，如果客户端在指定时间内未收到请求对应的 f + 1 相同响应，就认为集群出故障了，共识未达成，客户端会重新发送请求。

另外需要你注意的是，PBFT 算法通过视图变更（View Change）的方式，来处理主节点作恶，当发现主节点在作恶时，会以“轮流上岗”方式，推举新的主节点。

最后我想说的是，尽管 PBFT 算法相比口信消息型拜占庭之解已经有了很大的优化，将消息复杂度从 O(n ^ (f + 1)) 降低为 O(n ^ 2)，能在实际场景中落地，并解决实际的共识问题。但 PBFT 还是需要比较多的消息。比如在 13 节点集群中（f 为 4）。

- 请求消息：1
- 预准备消息：3f = 12
- 准备消息：3f * (3f - f) = 96

提交消息：(3f - f + 1) * (3f + 1)= 117

回复消息：3f - 1 = 11

也就是说，一次共识协商需要 237 个消息，你看，消息数还是蛮多的，所以我推荐你，在中小型分布式系统中使用 PBFT 算法。

内容小结

以上就是本节课的全部内容了，本节课我主要带你了解了口信消息型拜占庭问题之解的局限和 PBFT 的原理，我希望你明确这样几个重点。

1. 不管口信消息型拜占庭问题之解，还是签名消息型拜占庭问题之解，都是非常理论化的，未考虑实际场景的需求，而且协商成本非常高，指数级的消息复杂度是很难在实际场景中落地，和解决实际场景问题的。
2. PBFT 算法是通过签名（或消息认证码 MAC）约束恶意节点的行为，采用三阶段协议，基于大多数原则达成共识的。另外，与口信消息型拜占庭问题之解（以及签名消息型拜占庭问题之解）不同的是，PBFT 算法实现的是一系列值的共识，而不是单值的共识。

最后，我想说的是，相比 Raft 算法完全不适应有人作恶的场景，PBFT 算法能容忍 (n - 1)/3 个恶意节点 (也可以是故障节点)。另外，相比 PoW 算法，PBFT 的优点是不消耗算力，所以在日常实践中，PBFT 比较适用于相对“可信”的场景中，比如联盟链。

需要你注意的是，PBFT 算法与 Raft 算法类似，也存在一个“领导者”（就是主节点），同样，集群的性能也受限于“领导者”。另外，O(n ^ 2) 的消息复杂度，以及随着消息数的增加，网络时延对系统运行的影响也会越大，这些都限制了运行 PBFT 算法的分布式系统的规模，也决定了 PBFT 算法适用于中小型分布式系统。

课堂思考

当客户端在收到了 f + 1 个结果，就认为共识达成了，那么为什么这个值不能小于 f + 1 呢？欢迎在留言区分享你的看法，与我一同讨论。

最后，感谢你的阅读，如果这篇文章让你有所收获，也欢迎你将它分享给更多的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　12 | Quorum NWR算法：想要灵活地自定义一致性，没问题！

下一篇　14 | PoW算法：有办法黑比特币吗？

精选留言 (15)

写留言

笑若海
2020-03-11

如果接收到小于f+1个消息就认可服务返回结果，可能都是来自f个恶意节点的消息，导致客户端接受恶意结果。f+1保证至少一个正确结果，如果其中存在恶意消息，客户端会发现不一致性，认为请求处理失败。
这又引发一个新问题，客户端怎么确定f值?

展开

1　3

myrfy
2020-03-11

老师，可以详细解释一下视图变更是什么意思吗

展开

作者回复: 好，我后面补充下，具体说说和演示下。

2

Purson
2020-03-13

如果将军数为 n、叛将数为 f，那么算法需要递归协商 f+1 轮，消息复杂度为 O(n ^ (f + 1))，是怎样算出来的，第一讲说了两轮的能看明白，但是没有说3轮的，找不到递推关系，希望老师详细说一下BFT和PBFT两者区别

展开

1

忆水寒
2020-03-11

PBFT 算法通过视图变更（View Change）的方式，来处理主节点作恶，当发现主节点在作恶时，会以“轮流上岗”方式，推举新的主节点。
老师能详细补充一下吗?

作者回复: 好，我后面做个加餐吧，具体说说和演示下。

1

EidLeung
2020-03-15

PBFT需要提前知道叛将的数量么？实际落地过程中不可能提前知道啊，这又该怎么落地?

右耳听海
2020-03-15

麻烦老师补充下pbft实现一系列共识值pbft做了些什么优化，消息数是随一系列值倍数增加吗

作者回复: 加一颗星:)，是倍数增加的，相当于一轮新的共识协商。更多细节，我后面补充说说吧。

右耳听海

2020-03-15

老师能具体说下pbft实现的是一系列值的共识而不是单值的共识具体指什么吗，一系列值的共识不也可以包装成一个值吗，不如:进攻，准备粮草，这是两个值，但是也可以是在一个消息中

作者回复: 加一颗星:)，两个值，就是两个消息，放到一个消息中，就是一个值了。单值的共识，比如Basic Paxos，它只能就提议的一个值达成共识，你再提议新值，它是无法达成共识的。一系列值的共识，比如Multi-Paxos能就多值（也就是指令）达成共识，也就是你提议了一个值，可以再提议一个值，分别会达成共识，比如PBFT，客户端可以发送一个请求，再发送一个请求，请求的内容会分别达成共识，被忠诚的节点们执行。

Purson

2020-03-13

口信型的O(n ^ (f + 1))是怎样推导出来的，我看第一章说2轮，第一轮A向 B C D 分别发一个消息，记3，第二轮剩下的3个分别向对方发2消息，记6，加起来总共9，用 4^2好像不太对。除非第一轮的苏秦不是将军，或者n就是忠诚将军数，n=3，就对。但是如果是f=2，一共有7名将军，第二轮协商到底是怎样的顺序?

展开

翠羽香凝

2020-03-12

“口信消息型拜占庭问题之解的局限我想说的是，这个算法有个非常致命的缺陷。如果将军数为 n、叛将数为 f，那么算法需要递归协商 f+1 轮，” 这里看不懂，01讲不是说算法一共是两轮吗?

Fs

2020-03-12

这就是为什么区块链的效率提升不上去？达成共识的时间效率太低

6 7 8 9 10

2020-03-11

最后消息数的算法，看不懂呢

展开

congyh
2020-03-11

有一个问题想请教韩老师: PBFT算法如何处理集群节点数的变更?


梁伦
2020-03-11

图4下面的2f应该为2f+1

展开

作者回复: 加一颗星:)，这是很容易误解的一个点，是2f个，这2f个准备消息和预准备消息是相同的，所以，加上主节点，就是2f + 1了。


沉淀的梦想
2020-03-11

客户端要收到 f+1 个结果，我理解这个是为了防止 f 个叛徒直接给客户端返回 ok。不太理解的是为什么准备阶段要收到 2f 个一致的包含作战指令的准备消息，提交阶段需要 2f+1 个验证通过呢？这两个也设置成 f+1，不可以吗？

展开


小晏子
2020-03-11

课后思考：因为有f个坏的节点，如果客户端收到的结果小于f，最坏的情况是这f个结果都是坏节点发回来的，所以这就导致了客户端判断错误。所以客户端收到的结果必须大于f个，最少就是f+1个，也就是说最少要有一个好的节点发出来的结果来做判断。

展开