一种用于区块链的拜占庭容错算法

张铮文

erik@vcage.com

摘要

本文提出了一种改进的拜占庭容错算法 ,使其能够适用于区块链系统。我们假设在此网络中，消息可能会丢失、损坏、延迟、重复发送，并且接受的顺序与发送的顺序不一致。此外，节点的行为可以是任意的 :可以随时加入、退出网络 ,可以丢弃消息、伪造消息、停止工作等，还可能发生各种人为或非人为的故障。我们的算法对由n个共识节点组成的共识系统，提供$f = \left\lfloor \frac{n-1}{3} \right\rfloor$的容错能力，这种容错能力同时包含安全性和可用性，并适用于任何网络环境。

1. 概述

区块链是一种去中心化的分布式账本系统，它可以用于登记和发行数字化资产、产权凭证、积分等，并以点对点的方式进行转账、支付和交易。区块链技术最早是由中本聪在一个密码学的邮件列表中提出的[1]，也就是比特币。此后，基于区块链技术的各种应用纷纷出现，比如基于区块链的电子现金系统、基于区块链的股权交易系统、基于区块链的智能合约系统等。区块链系统与传统的中心化账本系统相比 ,具有完全公开、不可篡改、防止多重支付等优点，并且不依赖于任何的可信第三方。

然而，和任何分布式系统一样，区块链系统会面临网络延迟、传输错误、软件错误、安全漏洞、黑客入侵等问题。此外 ,去中心化的特点决定了此系统的任何一个参与者都不能被信任，可能会出现恶意节点，以及因各方利益不一致导致的数据分歧等问题。

为了防范这些潜在的错误 ,区块链系统需要一个高效的共识机制来确保每一个节点都有一个

唯一公认的全局账本。传统的针对某些特定问题的容错方法，并不能完全解决分布式系统以及区块链系统的容错问题，人们需要一种能够容忍任何种类错误的容错方案。

比特币采用工作量证明机制[1]，非常巧妙地解决了这个问题。但是代价也很明显，那就是巨额的电力成本和资源浪费。此外，新的区块链必须寻找到一种与之不同的散列算法，用于避免来自比特币的算力攻击，如莱特币采用了与比特币的 SHA256 不同的 SCRYPT 算法。

拜占庭容错技术是一种解决分布式系统容错问题的通用方案[5]。本文在 Castro 和 Liskov 于 1999 年提出的 Practical Byzantine Fault Tolerance(PBFT)[3]的基础上，提出了一种改进的拜占庭容错算法，使其能够适用于区块链系统。

2. 系统模型

区块链是一个分布式账本系统，参与者通过点对点网络连接，所有消息都通过广播的形式来发送。系统中存在两种角色：普通节点和记账节点。普通节点使用系统来进行转账、交易等操作，并接受账本中的数据；记账节点负责向全网提供记账服务，并维护全局账本。

我们假设在此网络中，消息可能会丢失、损坏、延迟、重复发送，并且接受的顺序与发送的顺序不一致。此外，节点的行为可以是任意的：可以随时加入、退出网络，可以丢弃消息、伪造消息、停止工作等，还可能发生各种人为或非人为的故障。

我们采用密码学技术来保证消息传递的完整性和真实性，消息的发送者要对消息的散列值进行签名。我们定义$\langle m\rangle_{\sigma_i}$是节点$i$对消息$m$的电子签名，$\mathrm{D}(m)$是消息$m$的散列值。如果没有特殊说明，本文所规定的签名都是对消息散列值的签名。

3. 算法

我们的算法同时提供了安全性和可用性，只要参与共识的错误节点不超过$\left\lfloor\frac{n-1}{3}\right\rfloor$，就能保证整

个系统正常运作，其中$n = |R|$表示参与共识的节点总数，R是共识节点的集合。令$f = \left\lfloor \frac{n-1}{3} \right\rfloor$，则$f$就表示系统所容许的错误节点的最大数量。由于实际上全局账本仅由记账节点来维护，因此系统中的普通节点不参与共识算法，但可以看到完整的共识过程。

参与共识的节点，需要维护一个状态表，用于记录当前的共识状态。一次共识从开始到结束所使用的数据集合，称为视图。如果在当前视图内无法达成共识，则需要更换视图。我们为每一个视图分配一个编号v，编号从 0 开始，并逐渐递增，直到达成共识为止。

我们为每一个参与共识的节点分配一个编号，从 0 开始，最后一个节点的编号为$n-1$。每一轮共识都需要有一个节点来充当议长，其它节点则为议员。议长的编号p由如下的算法来决定 :假设当前共识的区块高度为h ,则$p = (h - v) \ mod \ n$,其中p的取值范围为$0 \leq p < n$。

每一次共识产生一个区块，并附有至少$n - f$个记账节点的签名。一旦有新的区块产生，则立即开始新一轮的共识，同时重置$v = 0$。

3.1.一般流程

假设系统要求每次产生区块的时间间隔为t，则在一切正常的情况下，算法按照以下流程执行：

1) 任意节点向全网广播交易数据，并附上发送者的签名；

2) 所有记账节点均独立监听全网的交易数据，并记录在内存；

3) 议长在经过时间t后，发送$\langle PerpareRequest, h, v, p, block, \langle block \rangle_{\sigma_p} \rangle$；

4) 议员i在收到提案后，发送$\langle PerpareResponse, h, v, i, \langle block \rangle_{\sigma_i} \rangle$；

5) 任意节点在收到至少$n - f$个$\langle block \rangle_{\sigma_i}$后，共识达成并发布完整的区块；

6) 任意节点在收到完整区块后，将包含的交易从内存中删除，并开始下一轮共识；

该算法要求参与共识的节点中，至少有$n - f$个节点具有相同的初始状态：即对于所有的节

点i，具有相同的区块高度h和视图编号v。而这个要求很容易达成：通过区块同步来达到h的一致性，通过视图更换来达到v的一致性。区块同步不在本文讨论范畴，不再赘述。视图更换的规则见下文 3.2。

节点在监听全网交易以及在收到提案后，需要对交易进行合法性验证。如果发现非法交易，则不能将其写入内存池；如果非法交易包含在提案中，则放弃本次共识并立即开始视图更换。交易的验证流程如下：

1) 交易的数据格式是否符合系统规则，如果不符合则判定为非法；

2) 交易在区块链中是否已经存在，如果存在则判定为非法；

3) 交易的所有合约脚本是否都正确执行，如果没有则判定为非法；

4) 交易中有没有多重支付行为，如果有则判定为非法；

5) 如果以上判定都不符合，则为合法交易；

3.2. 视图更换

当节点i在经过$2^{v+1} \cdot t$的时间间隔后仍未达成共识，或接收到包含非法交易的提案后，开始进入视图更换流程：

1) 令$k = 1$，$v_k = v + k$；

2) 节点i发出视图更换请求$\langle ChangeView, h, v, i, v_k \rangle$；

3) 任意节点收到至少$n - f$个来自不同i的相同v_k后，视图更换达成，令$v = v_k$并开始共识；

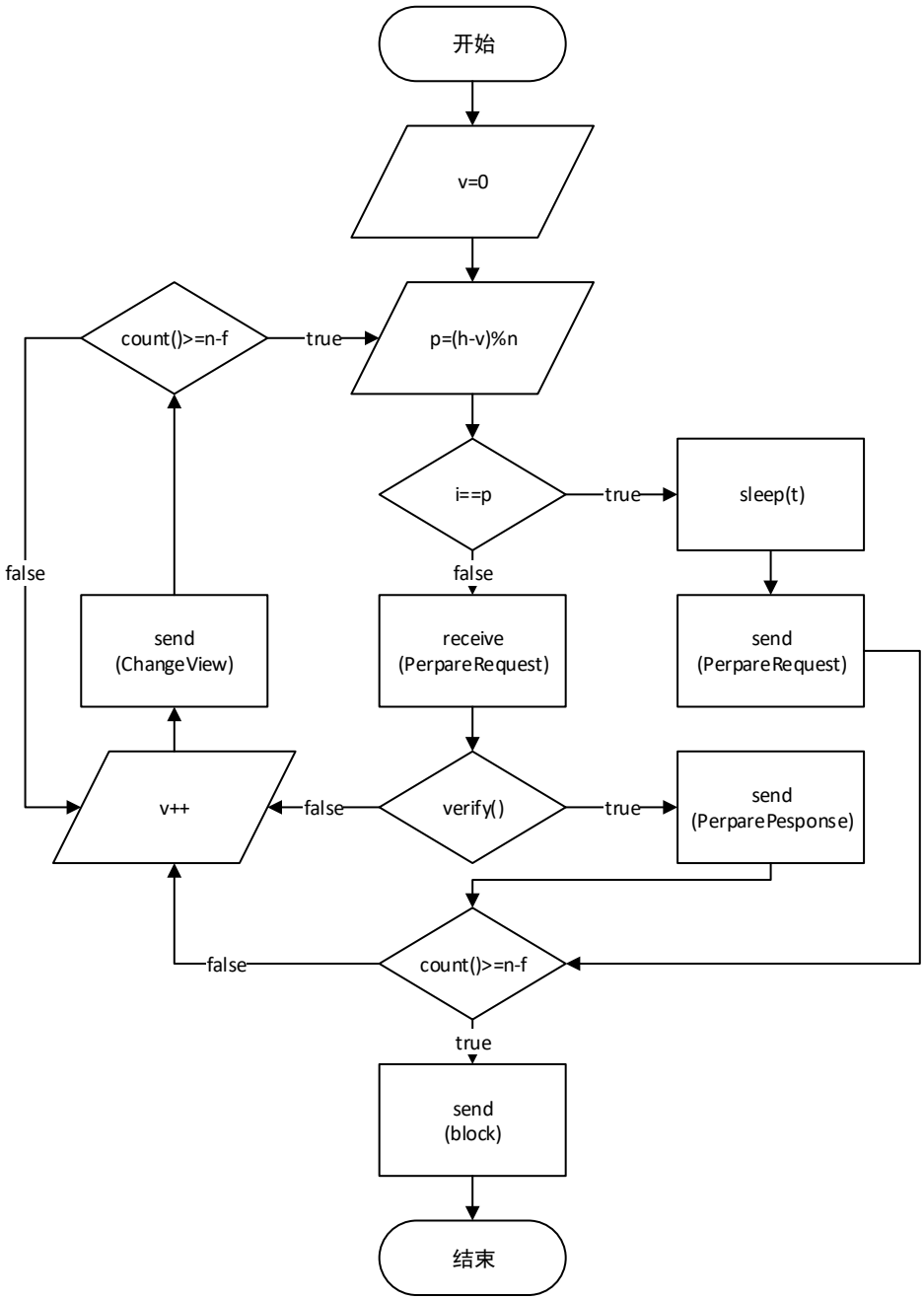
4) 如果在经过$2^{v_k+1} \cdot t$的时间间隔后，视图更换仍未达成，则k递增并回到第 2)步；

随着k的增加，超时的等待时间也会呈指数级增加，可以避免频繁的视图更换操作，并使各节点尽快对v达成一致。

而在视图更换达成之前，原来的视图v依然有效，由此避免了因偶然性的网络延迟超时而导

致不必要的视图更换。

3.3.流程图



4. 容错能力

我们的算法对由n个共识节点组成的共识系统，提供$f = \left\lfloor \frac{n-1}{3} \right\rfloor$的容错能力，这种容错能力同

时包含安全性和可用性，并适用于任何网络环境。

由于节点的请求数据包含发送者的签名，恶意的记账节点无法伪造请求，它只能试图将系统的状态回退到过去，从而使系统发生“分叉”。

我们假设系统所在的网络环境，恰好将所有共识节点分割成 3 个部分，即：$R = R_1 \cup R_2 \cup F$，且$R_1 \cap R_2 = \emptyset$，$R_1 \cap F = \emptyset$，$R_2 \cap F = \emptyset$。假设R_1和R_2都由诚实的记账节点组成，且已形成网络孤岛，各自只能与自己所在集合内的节点通讯；F全部都是恶意记账节点且已经合谋，可以统一行动；此外，F的网络条件允许它们和任意节点进行通讯，包括R_1和R_2。

如果F想要使系统发生“分叉”，只需与R_1达成共识并发布区块，并在不通知R_2的情况下与之达成第二次共识，“撤销”与R_1的共识。

若想满足这个条件，需满足：$|R_1| + |F| \geq n - f$，且$|R_2| + |F| \geq n - f$。在最坏的情况下，$|F| = f$，即恶意节点的数量达到系统所能容忍的最大值，则上述关系变为：$|R_1| \geq n - 2f$，且$|R_2| \geq n - 2f$。两式相加：$|R_1| + |R_2| \geq 2n - 4f$，化简后：$n \leq 3f$。已知$f = \left\lfloor \frac{n-1}{3} \right\rfloor$，与上式矛盾，故可证明系统在容错范围内无法被分叉。

5. 参考文献

[1] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J]. 2008.

[2] Lamport L, Shostak R, Pease M. The Byzantine generals problem[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1982, 4(3): 382-401.

[3] Castro M, Liskov B. Practical Byzantine fault tolerance[C]//OSDI. 1999, 99: 173-186.

[4] Bracha G, Toueg S. Asynchronous consensus and broadcast protocols[J]. Journal

of the ACM (JACM), 1985, 32(4): 824-840.

[5] 范捷, 易乐天, 舒继武. 拜占庭系统技术研究综述[J]. 软件学报, 2013, 6: 012.