拜占庭共识

在拜占庭容错（Byzantine fault tolerance）的环境中，共识算法的有效性（Validity）变体特别重要，因为它们需要考虑可能存在恶意或故障的进程。 **（不考大题！！！）**

有效性变体

1. **强有效性（Strong Validity）**：
    - 如果所有正确的进程都提出了相同的值`v`，那么没有一个正确的进程会决定一个不同于`v`的值。这意味着算法需要保证即使在面对拜占庭故障时，也不会做出与所有正确进程的一致提案不同的决定。
2. **弱有效性（Weak Validity）**：
    - 如果所有进程都是正确的并且提出了相同的值`v`，那么没有一个正确的进程会决定一个不同于`v`的值；此外，如果所有进程都是正确的，并且某个进程决定了`v`，那么`v`是由某个进程提出的。这种情况下，算法可以容忍在所有进程都正确的情况下做出一致的决定。

拜占庭错误：二义性（Equivocation）

- 在拜占庭容错模型中，最具挑战性的错误类型之一是“二义性”，其中拜占庭领导者（即可能是恶意的进程）向不同的正确进程发送不一致的请求。

`f < N/3`的下界证明（简述）

- 在拜占庭容错模型中，存在一个经典的下界：`f < N/3`，即为了容忍`f`个拜占庭错误，系统中需要至少`3f + 1`个进程。这个下界基于以下假设和论证：
    - 假设系统中有`N = 3f`个进程。显示在没有故障检测器（FD）的情况下，拜占庭一致性广播是不可能的。
    - 这是因为在最坏情况下，`f`个拜占庭进程可能会向一部分正确的进程发送一个值，而向另一部分正确的进程发送另一个不同的值。在这种情况下，正确的进程无法区分哪些信息是可信的，因此无法达成一致。

PBFT

PBFT（实用拜占庭容错，Practical Byzantine Fault Tolerance）是一种在分布式计算中应对拜占庭故障的算法。

PBFT的工作原理

1. **数字签名**：
    - PBFT使用数字签名来确保消息的完整性和来源的可靠性，这对于在拜占庭环境中防止恶意行为至关重要。
    - 与Paxos类似，PBFT也包含两个关键阶段：读取阶段（READ）和强加阶段（IMPOSE）。在读取阶段，领导者向其他节点询问当前的状态和值。在强加阶段，领导者基于收集到的信息决定下一步的提案，并尝试让其他节点接受这个提案。

Problems in the Byzantine world

1. 进入新一轮的时机

在拜占庭世界中，心跳机制可能无法正常工作，因为恶意进程可能会伪造或干扰心跳信号。因此，PBFT使用了超时机制来判断是否需要进入新一轮：

- 当客户端发送请求到进程时，该进程会将请求转发给当前轮次的领导者，并设置一个超时计时器。
- 如果在超时期限内没有决定被做出（即没有达成共识），进程会增加轮次计数器，并将请求再次转发给新一轮的领导者，然后重新设置超时计时器。

2. 确保新领导者选择最高轮次的估计值

为了确保新的领导者（如S1）选择了最高轮次的估计值，PBFT设计了以下机制：

- 新领导者需要广播至少`2f+1`个`GATHER`消息。这些消息包含关于之前轮次的信息和提案。
- 其他进程通过验证这些`GATHER`消息中的信息来确认新领导者的提案是基于最高轮次的估计值。这种验证机制类似于重放之前的选择过程，确保了新领导者的提案是合理的。

3. 将估计值传递到下一轮

在PBFT中，将一个轮次的估计值传递到下一个轮次是通过全体参与的方式完成的：

- **第一轮**：在第一轮中，所有进程都参与交换消息，以确保估计值的唯一性。这意味着在第一轮结束时，所有进程都对估计值有共同的理解。
- **第二轮及之后**：在第二轮以及随后的轮次中，同样采用全体进程参与的方式，以确保估计值可以被正确地传递到下一轮。这样，每一轮的领导者都能基于之前轮次的结果来提出新的提案。

实现

```
# 每个进程pi的初始化
Init: round := 1; estimate := nil; proposal := nil; estround := 0;
      proofvc[] := ø; proofpre[] := ø; states[][] := [nil,0]

# 客户端向所有进程发送请求
Client: send <REQUEST,o,t,c>_σ_c to all

# 收到来自客户端c的请求时
upon receiving req:= <REQUEST,o,t,c>_σ_c from client c do
    if proposal = nil then
        proposal := req;  # 如果还没有提案，则设置提案为请求

# 当轮次更新时
upon round updated do
    send <VIEWCHANGE,round, estimate,estround,proof_est>_σ_p_i to leader(round)

# 收到来自pj的视图更改消息时
upon receiving m:=<VIEWCHANGE,rnd, est,estrnd,proof>_σ_p_j from pj
    if [est,estrnd] is matching with proof then
        states[rnd][pj] := [est,estrnd]
        insert m into proofvc[rnd]
```

```
# 如果达到了足够多的状态并且自己是领导者时
upon #states[round] ≥ 2f+1 and leader(round) = self do
    if exists states[round][pk] ≠ [nil,0] then
        select states[rnd][pk]=[est,estrnd] with the highest estrnd
        proposal := est
    broadcast <NEWVIEW,round,proposal,proof_vc[round]>_σ_p_i     # NEWVIEW = PRE-
PREPARE

# 收到来自领导者的新视图消息时
upon receiving <NEWVIEW,round,est,proof>_σ_p_j from pj and pj = leader(round) do
    if proof contains 2f+1 VIEWCHANGE msgs and est is with the highest estrnd in
proof then
        broadcast <PREPARE,round,est>_σ_p_i

# 收到准备消息时
upon receiving m := <PREPARE,round,est>_σ_p_j from pj
    insert m into proofpre[round]

# 如果达到足够多的准备消息并且消息匹配时
upon #proofpre[round] ≥ 2f+1 and proofpre[round] contains 2f+1 matching est do
    estimate := est;
    estround := round;
    broadcast <COMMIT,round,estimate>_σ_p_j

# 收到足够多的提交消息时，做出决定
upon receiving <COMMIT,round,v>_σ_* from 2f+1 processes do
    decide(v)

# PBFT共识的结束

# 做出决定时的操作
upon decide(v) do
    r := execute v.o            # v.o是操作
    send <REPLY,v.t,r>_σ_pi to v.c  # v.t是时间戳；v.c是客户端

# 收到足够多匹配的回复消息时，接受结果
upon receiving <REPLY,v.t,r>_σ_* with matching r from f+1 processes do
    accept r

# 收到来自客户端c的请求时
upon receiving req:= <REQUEST,o,t,c>_σ_c from client c do
    forward req to leader(round)
    set timeout

# 如果超时并且没有请求被决定
upon timeout expires and no request decided do
    round++
    forward req to leader(round)
    set timeout
```

表现

消息复杂度

1. **最佳情况**：
   - 在最佳情况下，即不发生任何拜占庭故障且每个阶段都顺利完成的情况下，PBFT的消息复杂度为`O(n^2)`，其中`n`是系统中的进程数。这是因为在PBFT的每个阶段中，每个进程都需要与其他所有进程进行通信。
2. **通信复杂度**：
   - 然而，在一些情况下，尤其是在需要进行视图更改（即选举新的领导者）的情况下，PBFT的通信复杂度可以达到`O(n^3)`。具体原因如下：
     - **新领导者广播`NEWVIEW`消息**：新领导者需要将`NEWVIEW`消息广播给`O(n)`个进程。
     - `NEWVIEW`**消息包含**`VIEWCHANGE`**消息**：每个`NEWVIEW`消息包含`O(n)`个`VIEWCHANGE`消息，因为每个进程都可能发送一个`VIEWCHANGE`消息。
     - `VIEWCHANGE`**消息包含**`PREPARE`**消息**：每个`VIEWCHANGE`消息又包含`O(n)`个`PREPARE`消息，因为每个进程在之前的视图中都可能发送了`PREPARE`消息。

HotStuff

HotStuff 是一种在部分同步模型中的领导者为中心的拜占庭容错（BFT）共识协议。HotStuff 与传统的 BFT 协议（如 PBFT）相比，具有若干显著特点：

特点

1. **主动轮换领导者**：
   - 为了避免依赖单一领导者可能带来的风险和性能瓶颈，HotStuff 采用主动轮换领导者的策略。
2. **线性视图变更**：
   - HotStuff 实现了`O(n)`的视图变更复杂度，相比于 PBFT 的`O(n^3)`，大大降低了在领导者变更时的通信成本。
3. **乐观响应性**：
   - HotStuff 的新领导者只需要等待首批`n - f`个响应即可保证其提案能够取得进展，这增强了系统的响应能力和效率。

一般思想

- HotStuff 中的领导者负责消息的传播和聚合。它使用`(n, 2f+1)`阈值签名来结合`2f+1`个签名份额，从而将`O(n^2)`的通信复杂度减少到`O(n)`。
- 类似于 Tendermint，HotStuff 使用锁机制来防止转发视图变更消息（在 PBFT 中）。

锁定机制

- 在 PBFT 中，新的领导者需要广播`2f+1`个`VIEWCHANGE`消息，以便其他进程验证被强加的提案。这导致`O(n^3)`的通信复杂度。
- 在 HotStuff 中，进程锁定其估计值，使得每个进程都能独立地验证提案是否有效。
- 这种方法的权衡是需要额外一轮消息交换（与 PBFT 相比）。如果没有这一轮，系统可能会出现活锁问题。

实现

```
Hotstuff （omit signature operations）
# 每个进程pi的初始化
Init: round := 1; estimate := nil; proposal := nil; estround := 0;
      proofest = ∅; states[][] := [nil,0]; lockedest := nil; lockedrnd := 0

# 客户端向所有进程发送请求
Client: send <REQUEST,o,t,c>_σ_c to all

# 收到来自客户端c的请求时
upon receiving req:= <REQUEST,o,t,c>_σ_c from client c do
    if proposal = nil then proposal := req  # 如果还没有提案，则设置提案为请求

# 当轮次更新时
upon round updated do
    send <NEW-VIEW,round, estimate,estround,proof_est>_σ_p_i to leader(round)

# 收到来自pj的新视图消息时
upon receiving m:=<NEW-VIEW,rnd,est,estrnd,proof>_σ_p_j from pj
    if [est,estrnd] is matching with proof then
        states[rnd][pj] := [est, estrnd, proof]

# 如果达到了足够多的状态并且自己是领导者时
upon #states[round] ≥ 2f+1 and leader(round) = self do
    if exists states[round][pk]≠[nil,0] then
        select states[rnd][pk]=[est, estrnd, proof] with highest estrnd
        proposal := est
    broadcast <<PREPARE,round,proposal>_σ_p_i, estrnd, proof>

# 收到来自领导者的准备消息时
upon receiving <<PREPARE,round,est>_σ_p_j,estrnd,proof> from pj and pj =
leader(round) do
    if [est,estrnd] is matching with proof and (est = lockedest or estrnd >
lockedrnd) then
        send <PREPARE,round,est>_σ_p_i to leader(round)

# 收到足够多的准备消息时
upon receiving <PREPARE,round,est>_σ_* from 2f+1 processes do
    combine 2f+1 PREPARE signatures into sig
    estimate := est; estround := round; proofest := sig
    broadcast <<PRE-COMMIT, round, est>_σ_p_i, sig>

# 收到来自领导者的预提交消息时
upon receiving <<PRE-COMMIT,round,est>_σ_p_i,sig> from pj and pj = leader(round) do
    if [round,est] is matching with sig then
        estimate := est; estround := round; proofest := sig
        send <PRE-COMMIT,round,est>_σ_p_i to leader(round)

# 收到足够多的预提交消息并且自己是领导者时
```

```
upon receiving <PRE-COMMIT,round,est>_σ_* from 2f+1 processes and leader(round) =
self do
    combine 2f+1 PRE-COMMIT signatures into sig
    lockedest := est; lockedrnd := round
    broadcast <<COMMIT,round,est>_σ_p_i,sig>

# 收到来自领导者的提交消息时
upon receiving <<COMMIT,round,est>_σ_p_j,sig> from pj and pj = leader(round) do
    if [round,est] is matching with sig then
        lockedest := est; lockedrnd := round
        send <COMMIT,round,est>_σ_p_i to leader(round)

# 收到足够多的提交消息并且自己是领导者时
upon receiving <COMMIT,round,est>_σ_* from 2f+1 processes and self=leader(round) do
    combine 2f+1 COMMIT signatures into sig
    broadcast <<DECIDE,round,est>_σ_p_j,sig>
    decide(est)

# 收到来自领导者的决定消息时
upon receiving <<DECIDE,round,est>_σ_p_j,sig> from pj and pj = leader(round) do
    if [round,est] is matching with sig then
        decide(est)

# HotStuff共识的结束
```

HotStuff通过一系列消息传递和验证的步骤来达成共识。它使用了锁定估计值的机制来简化验证过程，每个进程可以独立验证提案的有效性。此外，HotStuff通过减少每轮所需的消息数目，降低了与PBFT相比的通信复杂度。该算法的每个阶段，如准备（PREPARE）、预提交（PRE-COMMIT）、提交（COMMIT）和决定（DECIDE），都需要收集足够多的签名或消息，以确保系统的一致性和安全性。通过这种方式，HotStuff为分布式系统提供了一种高效且可靠的共识机制。

表现

消息复杂度

- **最佳情况**：`O(n)`
    - 在最佳情况下，每个轮次中的每个阶段（如PREPARE, PRE-COMMIT, COMMIT）仅需要领导者向所有其他节点发送一次消息，然后每个节点响应一次。因此，消息复杂度为`O(n)`。

通信复杂度

- **最佳情况**：`O(n)`
    - 通信复杂度也是`O(n)`，因为在每个轮次的每个阶段，领导者只需要向所有其他节点广播一条消息，并且每个节点只需要回复一条消息。与PBFT协议的`O(n^3)`通信复杂度相比，HotStuff显著减少了所需的通信量。

优化

- **阈值签名**：HotStuff利用了阈值签名（threshold signature），将多个签名合并为一个，从而减少了消息的大小和处理签名所需的计算资源。

- **领导者轮换**：通过主动轮换领导者，HotStuff避免了对单一领导者的依赖，这也有助于提高协议的效率和抗拜占庭攻击的能力。