FLP impossibility result

FLP不可能性定理。这个定理的核心内容是：

在异步分布式系统中，即使只有一个进程可能发生故障（崩溃故障），也不可能实现确定性的共识算法。

FLP不可能性定理揭示了在异步分布式系统中，达成一致共识的困难。它表明，在不做额外假设（如部分同步、故障检测器的准确性等）的情况下，我们不能保证所有正确的进程总是能够在有限时间内就某个值达成共识。

FLP不可能性结果的总结

1. **双价性初始化引理（Bivalent Initialization Lemma）**：
   - 存在一个双价性的（无故障）初始配置。双值初始化引理表明，在没有故障的初始配置中，存在一个所谓的"双值"配置。在双值配置中，系统可以进入两种可能的最终状态（比如在共识问题中，这两种状态可以是决定值0或1）。这意味着系统的最初状态是不确定的，即无法预先确定系统将最终进入哪种状态。
2. **扩展引理（Extension Lemma）**：
   - 从任何双价性的无故障配置`C`出发，总能到达另一个双价性的无故障配置`C'`。这意味着，无论系统执行了多少步操作，总有可能保持在一个双值状态，即系统可以继续保持这种不确定性。
3. **因此，共识永远不会终止**：
   - 结合这两个引理，我们可以得出结论：在一个完全异步的系统中，如果允许进程故障，就无法保证总是能够达成共识。因为总是存在一条路径，沿着这条路径系统保持在双值状态，即使经过任意多的步骤，系统也无法决定一个特定的最终状态。

FLP不适用的情况

1. **时间假设**：包括故障检测器和同步性。在部分同步或完全同步的系统中，可以规避FLP的限制。
2. **没有可能的故障**：如果系统中不允许任何进程失败，FLP的限制不适用。
3. **随机化**：在某些情况下，使用随机化技术可以提供概率上的共识保证。

规避FLP的方法

1. **故障检测器**：
   - 例如P、→P、Ω等故障检测器。通过引入关于进程故障和恢复的额外信息，可以帮助系统在特定条件下达成共识。
2. **概率保证**：
   - 通过随机化算法，共识的属性可以以一定概率得到保证。
3. **确定性算法的局限性**：
   - FLP不可能性结果适用于确定性算法。对于概率性或随机化算法，FLP的限制可能不适用。通过引入随机性，算法可以在期望意义上绕过FLP限制，即使在一个异步系统中也能以高概率达成共识。

FLP不可能性结果是对分布式系统共识能力的一个重要限制，但在实际应用中，通过引入时间假设、故障检测器、随机化等技术，可以在一定程度上规避这些限制。这些方法在分布式系统的设计和实现中非常关键，帮助系统在不完美的网络环境和存在进程故障的情况下达成共识。

Randomized Consensus

Common Coin

在解决共识问题时。共同硬币通常用于提供随机化决策的机制，特别是在那些需要规避FLP不可能性结果的分布式算法中。这个机制可以帮助保证算法的活性，即使在异步环境下也能够达成共识。

属性

1. **终止（Termination）**：每个正确的进程最终都会输出一个硬币值。
2. **不可预测性（Unpredictability）**：在至少一个正确的进程释放硬币之前，没有任何进程能够获得关于由正确进程输出的硬币值的任何信息。
3. **匹配（Matching）**：至少有δ的概率，每个正确的进程输出相同的硬币值。
4. **无偏差（No Bias）**：如果所有正确的进程输出相同的硬币值，那么硬币在B上的分布是均匀的（即匹配的硬币以1/#(B)的概率输出B中的任何值）。

可能的共同硬币实现

1. **本地抛硬币（Local Toss）算法**：
   - 当释放硬币时，每个进程从D中随机选择一个值`c`，并按均匀分布输出`c`。如果域是一个比特，那么这实现了一个δ=2-N+1匹配的共同硬币。选择相同的`c∈{0,1}`的概率是2^(-N)。
2. **信标算法（Beacon Algorithm）**：
   - 稍微作弊：使用外部进程，不完全分布式。可信第三方
   - 一个外部可信进程（信标）定期选择一个不可预测的随机值。
   - 当算法需要一系列共同硬币时，对于第k个硬币，每个进程从信标接收第k个随机值并输出它。
   - 这种硬币总是匹配（δ=1）。
3. **门限签名方案（Threshold Signature Scheme）**：
   - 也可以实现δ=1的匹配硬币。

共同硬币在解决共识问题，特别是在概率共识算法中非常有用。它提供了一种方式，以使所有正确的进程以高概率做出相同的随机决策，即使在存在不确定性和潜在故障的分布式系统中。通过使用共同硬币，分布式算法能够在不完美的环境中达到共识，特别是在考虑FLP不可能性结果时。

Randomized Binary Consensus（随机二元共识）

属性

随机化二元共识算法，这是一种在分布式系统中达成共识的方法，特别是在异步环境中，它利用随机化来克服FLP不可能性结果带来的挑战。在这种算法中，提案/决策的值限定在{0,1}，并且基于正确的多数假设。

1. **轮次基算法（Round-based Algorithm）**：
   - 每一轮中，进程试图确保大多数进程提出相同的值。
   - 如果没有这样的值，进程将依赖于共同硬币来为下一轮选择一个提案。
2. **算法终止**：

- 算法最终以概率1终止。
- 每轮包括两个阶段。

3. **每轮操作**：
   - **第一阶段**：每个正确的进程通过尽力而为广播向所有进程提出一个值，然后从大多数进程接收提案。
   - **第二阶段**：如果从多数进程接收到的所有提案都是相同的，则传播这个提案；否则，传播⊥。
4. **不变性（Invariant）**：
   - 如果两个进程在第二阶段传播`v1`和`v2`，并且`v1`、`v2`不等于⊥，那么`v1 = v2 = v`。
5. **第二阶段的操作**：
   - 在接收到大多数第二阶段消息后，进程检查是否所有消息都等于`v`。
   - 如果是：进程决定`v`并可靠地广播决定。
   - 否则，如果存在不等于⊥的`v`：在下一轮中采用`v`作为提案。
   - 否则：采用共同硬币的结果作为下一轮的提案。
6. **依赖于共同硬币的随机性**：
   - 成功的概率可能取决于共同硬币的匹配概率δ。
   - 除非所有（初始）提案都是相同的，在这种情况下，算法可能会更快地达成共识。

随机化二元共识算法提供了一种在分布式系统中达成共识的有效方式，尤其是在处理FLP不可能性结果带来的挑战时。通过在每一轮中结合确定性行为（如从多数进程接收相同提案）和随机性行为（如共同硬币的使用），这种算法能够在不完美的网络环境和可能存在进程故障的情况下，以高概率最终达成共识。

实现

```
Implements: RandomizedConsensus, instance rc, with domain {0, 1}.
Uses: BestEffortBroadcast, instance beb; ReliableBroadcast, instance rb; CommonCoin.

# 当随机化共识实例初始化时触发的事件
upon event < rc, Init > do
    round := 0;  # 初始化轮次
    phase := 0;  # 初始化阶段
    proposal := ⊥;  # 初始化提案值为未定义
    decision := ⊥;  # 初始化决定值为未定义
    val := [⊥]N;  # 初始化所有进程的值数组为未定义

# 当有共识提案时触发的事件
upon event < rc, Propose | v > do
    proposal := v;  # 设置提案值为v
    round := 1;  # 设置轮次为1
    phase := 1;  # 设置阶段为1
    trigger < beb, Broadcast | [PHASE-1, round, proposal] >;  # 广播[PHASE-1, round,
proposal]消息，每个都广播

# 当通过最佳努力广播接收到第一阶段消息时触发的事件
upon event < beb, Deliver | p, [PHASE-1, r, v] > such that phase = 1 ∧ r = round do
    val[p] := v;  # 记录收到的值
```

```
# 当收到多数第一阶段消息且尚未决定时触发的事件
upon #(val) > N/2 ∧ phase = 1 ∧ decision = ⊥ do
    if 存在非⊥的v使得多数进程的val[p]为v then
        proposal := v;  # 设置提案为v
    else
        proposal := ⊥;  # 设置提案为⊥
    val := [⊥]N;  # 重置val数组
    phase := 2;  # 进入第二阶段
    trigger < beb, Broadcast | [PHASE-2, round, proposal] >;  # 广播[PHASE-2, round,
proposal]消息

# 当通过最佳努力广播接收到第二阶段消息时触发的事件
upon event < beb, Deliver | p, [PHASE-2, r, v] > such that phase = 2 ∧ r = round do
    val[p] := v;  # 记录收到的值

# 当收到大多数第二阶段消息且尚未决定时触发的事件
upon #(val) ≥ N - f ∧ phase = 2 ∧ decision = ⊥ do
    phase := 0;  # 重置阶段
    初始化一个新的共同硬币实例coin.round;
    trigger < coin.round, Release >;  # 释放共同硬币

# 当共同硬币产生输出时触发的事件
upon event < coin.round, Output | c > do
    if 存在非⊥的v使得多数进程的val[p]为v then
        decision := v;  # 决定为v
        trigger < rb, Broadcast | [DECIDED, decision] >;  # 可靠广播决定
    else
        if 存在非⊥的w使得某个进程的val[p]为w then
            proposal := w;  # 设置下一轮提案为w
        else
            proposal := c;  # 设置下一轮提案为共同硬币的结果
        val := [⊥]N;  # 重置val数组
        round := round + 1;  # 进入下一轮
        phase := 1;  # 设置阶段为1
        trigger < beb, Broadcast | [PHASE-1, round, proposal] >;  # 广播[PHASE-1,
round, proposal]消息

# 当通过可靠广播接收到决定消息时触发的事件
upon event < rb, Deliver | p, [DECIDED, v] > do
    decision := v;  # 设置决定为v
    trigger < rc, Decide | decision >;  # 触发决定事件
```

每个进程首先尝试通过广播自己的提案来达成一致，如果没有达成一致，则依赖共同硬币的输出来决定下一轮的提案。这个过程不断重复，直到所有正确的进程能够达成共识为止。

正确性

总结了随机化二元共识算法的正确性属性，包括有效性（Validity）、一致性（Agreement）、终止性（Termination）以及性能指标。

有效性（Validity）

- 如果所有的提案都是相同的，则没有进程会改变其提案为共同硬币的结果，提案值将是唯一可能的决策值。
- 否则（如果提出了0和1），决策值无论如何都是0或1。

一致性（Agreement）

- 通过多数交集的原则：如果两个进程在同一轮中发送了包含非⊥提案的第二阶段（PHASE-2）消息，则这两个提案必定等于某个值`v`。

终止性（Termination）

- 当所有正确的进程以相同的提案开始新一轮时，算法将终止。
- 由于共同硬币的匹配，所有进程以相同的值输出硬币的概率为δ，且该值是均匀选择的随机位。
- 如果所有正确的进程以相同的提案开始一轮，那么每个正确的进程只会收到包含`v`的第二阶段消息，随后在同一轮中广播带有`v`的决定消息。

性能

- **每轮性能**：
  - 需要2次消息延迟。（一个消息延迟包括消息的消息的发送（广播）和接收）
  - $O(n^2)$条消息。
- **轮数**：
  - 以概率0，轮数可能是无限的。
  - 预期的轮数与$1/\delta$成正比。

这个随机化二元共识算法利用了共同硬币的随机性来保证算法最终能够终止，并达成一致。虽然在最坏情况下（极低概率事件），算法可能需要无限轮才能结束，但在大多数情况下，预期的轮数相对较少，特别是当共同硬币输出一致的结果时（即δ值较高时）。这种算法在异步分布式系统中特别有用，因为它能在没有时间假设的情况下解决共识问题。

Randomized Multivalued consensus （随机多元共识）

在分布式系统中，将随机化二元共识算法（针对的是仅有两个可能值的情况）扩展到多值共识（Multivalued Consensus）是一个有趣的挑战。多值共识允许进程从一个更大的值域中提出值，而不仅仅是{0, 1}。这个扩展需要考虑几个关键问题，包括如何使用具有更大值域的共同硬币以及这对有效性（Validity）属性的影响。

使用更大值域的共同硬币

- **可行性**：理论上，可以使用具有更大值域的共同硬币来实现随机化多值共识。这样的硬币可以从一个较大的集合中随机选择一个值。

- **挑战**：在多值场景中，保证共同硬币的输出能够帮助进程最终达成一致会更加复杂。在二元共识中，共同硬币的作用是在进程无法达成一致时打破僵局。但在多值共识中，选择的机制和时机需要更加精细的控制。

对有效性的影响

- **有效性定义**：在共识算法中，有效性通常要求如果一个进程决定了一个值，那么这个值必须是由某个进程提出的。
- **挑战**：在多值共识中，当使用具有更大值域的共同硬币时，需要确保算法仍然遵守有效性的要求。特别是，在决定一个非二元值时，算法需要确保这个值是由某个进程提出的，而不是仅由共同硬币随机生成的。

实现方法

- **间接方法**：一种方法是首先使用随机化二元共识来减少值的候选集，然后在这个更小的集合上进行多值共识。
- **直接方法**：另一种方法是修改共识算法，使其直接在多值域上运行，但这通常需要更复杂的协调和决策逻辑。

实现

```
Randomized Multivalued consensus •(Asynchronous Common Subset, ACS)
Implements: Randomized Multivalued Consensus, instance rmc.
Uses: ReliableBroadcast (rb), n instances; RandomizedBinaryConsensus (rc), n
instances.

# 当随机化多值共识实例初始化时触发的事件
upon event < rmc, init > do
    proposal := ⊥;  # 初始化提案值为未定义
    delivered := [⊥]N;  # 初始化所有进程的接收值数组为未定义
    proposed := [FALSE]N;  # 初始化所有进程的提议状态为假

# 当有共识提案时触发的事件
upon event < rmc, Propose | v > such that proposal = ⊥ do
    proposal := v;  # 设置提案值为v
    trigger < rb, Broadcast | v >;  # 可靠广播提案值

# 当通过可靠广播接收到消息时触发的事件
upon event < rb, Deliver | p, m > do
    delivered[p] := m;  # 记录p进程的提案值
    if proposed[p] = FALSE then  # 如果p进程尚未提议
        proposed[p] := TRUE;  # 设置p进程的提议状态为真
        trigger < rc.p, Propose | 1 >;  # 在rc.p实例上提议1

# 当随机化二元共识实例做出决定时触发的事件
upon event < rc.p, Decide | v > do
    if v = 1 then  # 如果决定值为1
        decidedone := decidedone ∪ {p};  # 将p进程加入到决定为1的进程集合中
    if #decidedone ≥ f+1 then  # 如果决定为1的进程数至少为f+1
        for ∀ p : proposed[p] = FALSE do  # 对于所有尚未提议的进程
            proposed[p] := TRUE;  # 设置提议状态为真
```

```
            trigger < rc.p, Propose | 0 >;  # 在rc.p实例上提议0
        decided := decided ∪ {p};  # 将p进程加入到已决定的进程集合中
        if #decided = n then  # 如果所有进程都已决定
            s := maxrank(decidedone);  # 选择具有最高排名的进程
            trigger < rmc, Decide | delivered[s] >;  # 触发决定事件，决定值为该进程的接收值

            # 或者，可以决定所有decidedone集合中进程的接收值的并集
            # trigger < rmc, Decide | {∪delivered[s] | ∀ s ∈ decidedone } >;
```

这个算法的目的是在一个异步环境中实现多值共识。它首先通过可靠广播发送提案值，然后利用随机化二元共识算法在每个进程中独立地决定是否接受这个提案。通过这种方式，算法能够在不确定的网络条件下有效地达成共识。