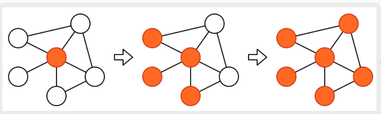
Gossip

前言

- Problem
  - 不管是前面提到的 Basic Paxos 还是 Raft 共识算法，它们都有一个非常显著的特征: 就是必须要保证集群中多数节点存活，算法才能够正常工作。从 CAP 的角度来讲，它们满足了 CP，并且能够容忍少数节点出现故障
  - 但是，如果系统对可用性要求非常高，集群中如果只剩下一个节点，仍然要求能够正常工作的话，那么 Basic Paxos 或者是 Raft 就不是一个好的选择，因为它们必须在大多数节点存活的情况下才能正常工作
- 那么，当我们对数据一致性要求并没有那么高，最终一致性也能够接受，并且需要集群保持极高的可用性的话，Gossip 协议就是其中一个选择
- Gossip 算法最早由施乐（Xerox）公司提出，也就是那个最早发明 GUI 而被乔布斯借鉴开发出了 Apple Lisa 的那家公司，现在在做打印机，不得不说命运是一个很神奇的东西
- Gossip 算法被翻译成 “流言算法”、“流行病算法”、“瘟疫算法”，虽然听着不是很好，但准确地描述了 Gossip 算法的特点: 像流言一样漫天飞，像病毒一样迅速扩散

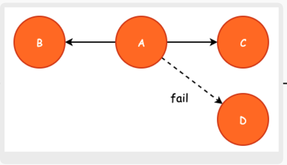


如左图所示，Gossip 就是利用了一种随机地且带有传染性性的方式将信息传播到整个网络中，并且在一段时间以后，所有的节点均被 “感染”，从而达到最终一致性

Gossip 运转机理

❶ 直接邮递（Direct Mail）

- 直接邮递是 Gossip 中最简单的一环，直接将更新的数据发送到其它节点。如果发送失败的话，那么就暂存到发送队列中，然后寻找合适的时机进行重试
- 但是，发送队列可能会达到最大容量导致数据丢失，所以，还需要其它的方式来保证数据的一致性



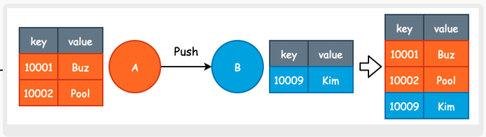
Node-A 向 Node-D 发送消息失败

❷ 反熵（Anti-Entropy）

- 直接邮递存在数据丢失的可能，因此 Gossip 通过反熵（Anti-Entropy）的手段来异步修正数据
- 熵（Entropy）在计算机中描述是系统的混乱程度，熵越高，系统无序程度就越高。而反熵（Anti-Entropy）就是主动地降低无序程度，这和我们整理桌面、笔记没有什么区别
- 反熵是指集群中的节点每隔一段时间就会随机地选择某个其它的节点，通过交换双方的数据来消除差异的过程

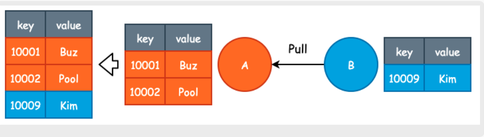
Push

- Push 其实就是将自己的数据推送给另一个随机的节点，比如 Node-A 将数据发送给 Node-B，来修复 Node-B 的数据



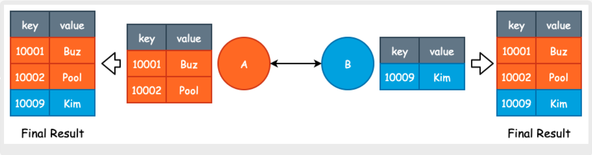
Pull

- Pull 其实就是拉取另一个随机节点的数据，比如 Node-A 主动拉取 Node-B 的数据，来修复 Node-A 的数据



Push & Pull

- 推拉模式，用于修复双方的数据。上面两个过程结束后 Node-A 和 Node-B 的数据达到一致



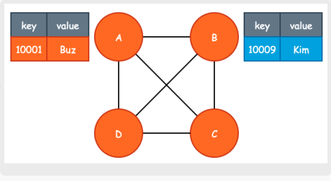
- 反熵的成本很高，不仅需要 2 次网络通信，还需要将全部数据通过网络传输给其它节点。因此，反熵的动作不应频繁执行，并且必须引入校验和（Check Sum）来快速判断节点间数据差异情况

❸ 谣言传播（Rumor Mongering）

- 谣言传播其实就是直接邮递的集群过程。但某一个节点收到信息以后，那么它将会选择一个固定周期（比如2S），随机地向它所连接的 K 个节点传播消息
- 当这 K 个节点收到消息以后，如果这个消息它从来没有收到过的话，那么也会周期性的选择 K 个相邻的节点发送该消息
- 最终，集群中所有的节点都会收到这条消息，从而达成一致。这个过程酷似 BFS 或者是 Flood Fill 的过程，只不过谣言传播的过程是节点主动发起，并且是并行进行的

再议反熵

反熵顺序



假设现有集群的拓扑结果如左图所示，共有 4 个节点，并且组成了一副稠密图，也就是任意一个节点都和其它节点直接相连。那么此时，反熵的顺序会对集群对某个值达成共识的所需时间产生影响吗？

假设反熵的顺序为 A-D-B-C-A，也就是组成了一个汉密尔顿回路

可以看到，在走过上述路径后，所有的节点均达成一致。事实上，对于任何一个汉密尔顿路径而言，在没有新增数据的情况下，总是能够使得大约一半的节点数据完全一致

因此，在实现反熵功能时，如果可能的话，可以按照某一个固定的顺序使用汉密尔顿回路的方式来进行，在最少的网络通信下达到更多节点保持一致的状态。这样一来，集群达成共识的时间我们可以预估，从而能够更好的进行 Trouble Shooting，而不是将决定权交给 “随机化”

数据对比

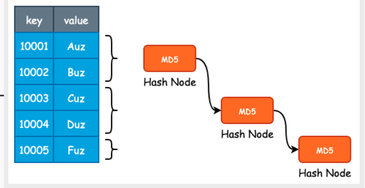
反熵的另外一个问题就是两个节点间的数据对比，如果直接使用全量数据进行对比的话，那么这个过程在数据量较大时将会产生非常多的消耗。因此我们需要有一个 Check Sum 来降低一致性检测的性能消耗

全量数据 MD5

- 一种方式是直接对比全量数据的 MD5，若两者不一致，则开始进行反熵
- 这种方式可以以 O(1) 的时间复杂度和网络传输复杂度完成数据对比，但是这种方式在有数据更新时，又需要重新计算所有数据的 MD5，效率较差

哈希链

- 将数据分成一段一段的，然后分别计算每一段数据的 MD5，将其组成一个链表或者是树，前者称之为哈希链，后者则称之为哈希树，在区块链中应用较多



- 这种方式在更新数据时，只需要重新计算一小部分的 MD5 值即可，但是对比成本和网络传输成本较高

小结

- Gossip 是一种去中心化的具有高度容错的最终一致性算法，就算集群中只剩下一个节点，仍然能够正常工作
- Gossip 使用直接邮递、反熵和谣言传播来实现数据的最终一致性。直接邮递将数据直接发送给其它节点，若发送失败，则进行重传；反熵通过一种随机化的方式来修正两个节点间的数据差异；谣言传播则是以 “病毒扩散” 的方式去中心化地传播信息
- Gossip 并不适用于集群中节点数量较多的情况，因为此时节点间的反熵和谣言传播将会成为系统瓶颈