Gossip 协议

jask

2024-08-23

Gossip 协议

Gossip 的三板斧分别是：直接邮寄（Direct Mail）、反熵（Anti-entropy）和谣言传播（Rumor mongering）。

直接邮寄：就是直接发送更新数据，当数据发送失败时，将数据缓存下来，然后重传。从图中你可以看到，节点 A 直接将更新数据发送给了节点 B、D。

只采用直接邮寄是无法实现最终一致性的，要是先最终一致性需要反熵。

本质上，反熵是一种通过异步修复实现最终一致性的方法。常见的最终一致性系统（比如 Cassandra），都实现了反熵功能。

反熵指的是集群中的节点，每隔段时间就随机选择某个其他节点，然后通过互相交换自己的所有数据来消除两者之间的差异，实现数据的最终一致性：

在实现反熵的时候，主要有推、拉和推拉三种方式。

因为反熵需要节点两两交换和比对自己所有的数据，执行反熵时通讯成本会很高，所以我不建议你在实际场景中频繁执行反熵，并且可以通过引入校验和（Checksum）等机制，降低需要对比的数据量和通讯消息等。

虽然反熵很实用，但是执行反熵时，相关的节点都是已知的，而且节点数量不能太多，如果是一个动态变化或节点数比较多的分布式环境（比如在 DevOps 环境中检测节点故障，并动态维护集群节点状态），这时反熵就不适用了。那么当你面临这个情况要怎样实现最终一致性呢？答案就是谣言传播。

谣言传播，广泛地散播谣言，它指的是当一个节点有了新数据后，这个节点变成活跃状态，并周期性地联系其他节点向其发送新数据，直到所有的节点都存储了该新数据：

从图中你可以看到，节点 A 向节点 B、D 发送新数据，节点 B 收到新数据后，变成活跃节点，然后节点 B 向节点 C、D 发送新数据。其实，谣言传播非常具有传染性，它适合动态变化的分布式系统。

如何使用 Anti-entropy 实现最终一致性

反熵的目标是确保每个 DATA 节点拥有元信息指定的分片，而且不同节点上，同一分片组中的分片都没有差异。比如说，节点 A 要拥有分片 Shard1 和 Shard2，而且，节点 A 的

推方式，就是将自己的所有副本数据，推给对方，修复对方副本中的熵：

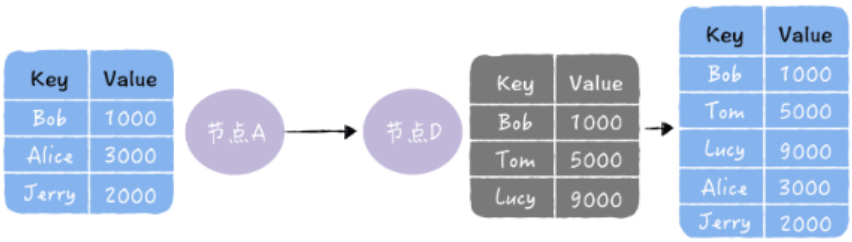

图4

拉方式，就是拉取对方的所有副本数据，修复自己副本中的熵：

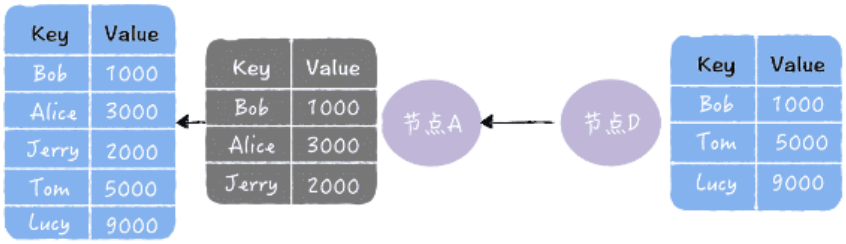

图5

理解了推和拉之后，推拉这个方式就很好理解了，这个方式就是同时修复自己副本和对方副本中的熵：

Figure 1: 三种

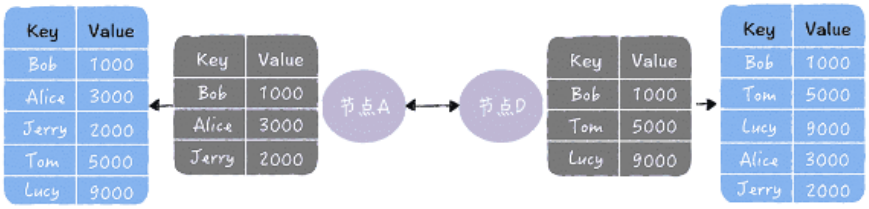

Figure 2: 推拉

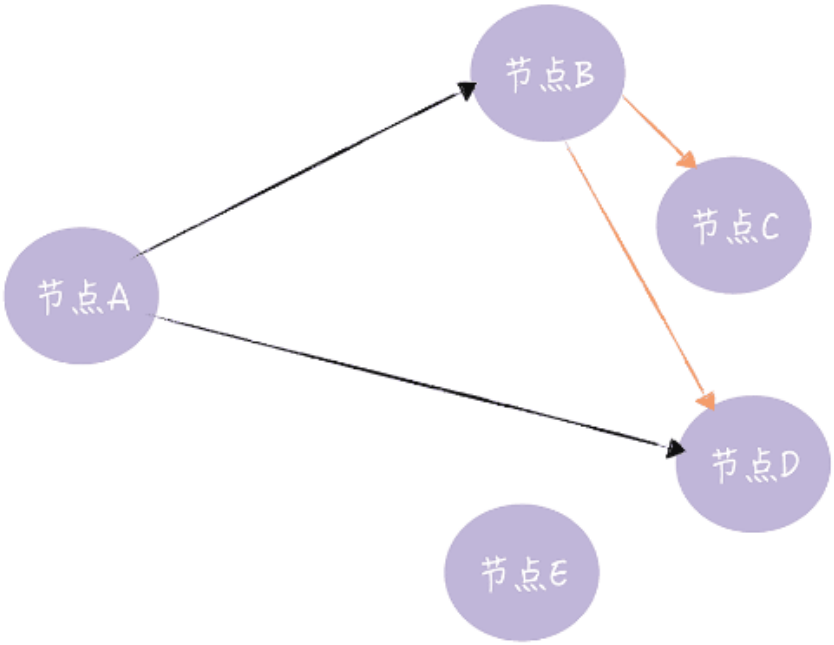

Figure 3: 状态

Shard1 和 Shard2，与节点 B、C 中的 Shard1 和 Shard2，是一样的。

那么，在 DATA 节点上，存在哪些数据缺失的情况呢?

1. 缺失分片：也就是说，在某个节点上整个分片都丢失了。

2. 节点之间的分片不一致：也就是说，节点上分片都存在，但里面的数据不一样，有数据丢失的情况发生。

第一种情况修复起来不复杂，我们只需要将分片数据，通过 RPC 通讯，从其他节点上拷贝过来就可以了。

二种情况修复起来要复杂一些，按照一定顺序来修复节点的数据差异，先随机选择一个节点，然后循环修复，每个节点生成自己节点有、下一个节点没有的差异数据，发送给下一个节点，进行修复。

实现细节与算法细节并不一样

并不是随机的选择节点，而是一次修复所有节点的数据不一致。

> 这样做能减少数据不一致对监控视图影响的时长。而我希望你能注意到，技术是要活学活用的，要能根据场景特点权衡妥协，设计出最适合这个场景的系统功能。最后需要你注意的是，因为反熵需要做一致性对比，很消耗系统性能，所以建议你将是否启用反熵功能、执行一致性检测的时间间隔等，做成可配置的，能在不同场景中按需使用。

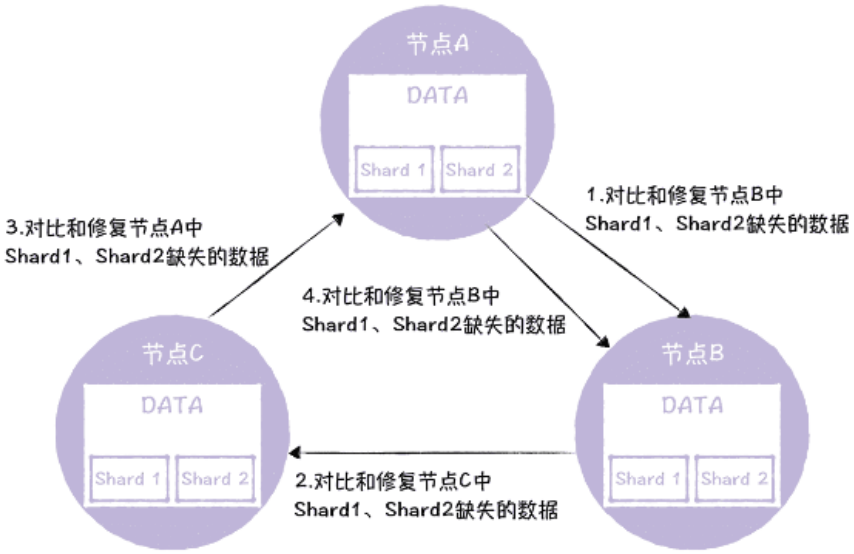

Figure 4: 流程

总结

作为一种异步修复、实现最终一致性的协议，反熵在存储组件中应用广泛，比如 Dynamo、InfluxDB、Cassandra，我希望你能彻底掌握反熵的实现方法，在后续工作中，需要实现最终一致性时，优先考虑反熵。

因为谣言传播具有传染性，一个节点传给了另一个节点，另一个节点又将充当传播者，传染给其他节点，所以非常适合动态变化的分布式系统，比如 Cassandra 采用这种方式动态管理集群节点状态。

实现数据副本的最终一致性时，一般而言，直接邮寄的方式是一定要实现的，因为不需要做一致性对比，只是通过发送更新数据或缓存重传，来修复数据的不一致，性能损耗低。在存储组件中，节点都是已知的，一般采用反熵修复数据副本的一致性。当集群节点是变化的，或者集群节点数比较多时，这时要采用谣言传播的方式，同步更新数据，实现最终一致。