ZAB 协议：如何实现操作的顺序性

jask

2024-08-11

ZAB 协议

为什么 Multi-Paxos 无法实现操作顺序性

兰伯特的 Multi-Paxos 解决的是一系列值如何达成共识的问题，它关心的是，对于指定序号的位置，最多只有一个指令（Command）会被选定，但它不关心选定的是哪个指令，也就是说，它不关心指令的顺序性（也就是操作的顺序性）。

ZAB 如何保证顺序性

与兰伯特的 Multi-Paxos 不同，ZAB 不是共识算法，不基于状态机，而是基于主备模式的原子广播协议，最终实现了操作的顺序性。

这里我说的主备，就是 Master-Slave 模型，一个主节点和多个备份节点，所有副本的数据都以主节点为准，主节点采用二阶段提交，向备份节点同步数据，如果主节点发生故障，数据最完备的节点将当选主节点。而原子广播协议，你可以理解成广播一组消息，消息的顺序是固定的。

需要你注意的是，ZAB 在这里做了个优化，为了实现分区容错能力，将数据复制到大多数节点后（也就是如果大多数节点准备好了），领导者就会进入提交执行阶段，通知备份节点执行提交操作。在这一点上，Raft 和 ZAB 是类似的，我建议你可以对比着 Raft 算法来理解 ZAB。

什么是状态机

本质上来说，状态机指的是有限状态机，它是一个数学模型。你可以这么理解：状态机是一个功能模块，用来处理一系列请求，最大的特点就是确定性，也就是说，对于相同的输入，不管重复运行多少次，最终的内部状态和输出都是相同的。

为什么爱 Multi-Paxos, Raft 中需要状态机呢？ Multi-Paxos、Raft 都是共识算法，而共识算法是就一系列值达成共识的，达成共识后，这个值就不能改了。但有时候我们是需要更改数据的值的，比如 KV 存储，我们肯定需要更改指定 key（比如 X）对应的值，这时我们就可以通过状态机来解决这个问题。

实现操作的顺序性

首先，ZAB 实现了主备模式，也就是所有的数据都以主节点为准：

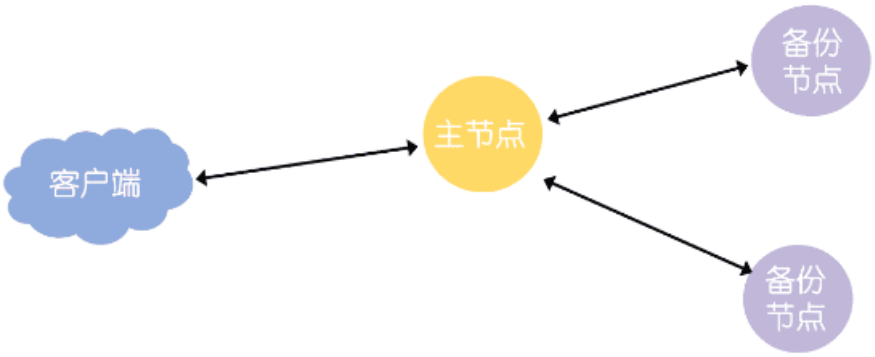

Figure 1: 架构

其次，ZAB 实现了 FIFO 队列，保证消息处理的顺序性。

另外，ZAB 还实现了当主节点崩溃后，只有日志最完备的节点才能当选主节点，因为日志最完备的节点包含了所有已经提交的日志，所以这样就能保证提交的日志不会再改变。

与 Raft 的相似：

所有日志以领导者的为准；

领导者接收到客户端请求后，会基于请求中的指令，创建日志项，并将日志项缓存在本地，然后按照顺序，复制到其他节点和提交；

在 Raft 中，也是日志最完备的节点才能当选领导者。