

分布式一致性协议

杨其臻

Oct 19, 2026

在单机时代，数据的正确性通常由本地事务和文件系统保证。当服务演进到分布式集群后，同一份数据会被复制到多台机器上，这就产生了「副本一致性」问题：如何让所有副本在任意时刻都呈现出相同且正确的状态。网络分区、节点宕机、消息乱序或丢失都会打破这种假设，因此分布式系统必须在「正确性」与「可用性」之间做出取舍。接下来的内容将从定理、协议到工程实践逐层展开。

1 理论基石

1.1 CAP 定理与 PACELC 模型

CAP 定理指出，在网络分区（Partition）发生时，系统只能在一致性（Consistency）和可用性（Availability）之间二选一。该定理并不意味着三者永远不可兼得，而是在特定故障场景下必须做出权衡。PACELC 模型进一步细化了这一思想：若发生分区（P），则在可用性与一致性之间选择；否则（Else），则在延迟（Latency）和一致性之间选择。工程上，系统设计者会根据业务 SLA 决定在不同场景下的取舍策略。

1.2 一致性模型谱系

分布式系统的一致性模型可视为一个连续谱系。强一致性要求任意时刻所有副本对外呈现相同视图；顺序一致性放宽到「看起来像存在全局时钟」；因果一致性仅保证因果相关的操作有序；最终一致性则允许在无并发写入时，副本最终收敛。模型越强，对延迟和可用性的影响越大，需根据业务语义选择。

1.3 Quorum 机制

Quorum 通过配置三个参数 N 、 W 、 R 来量化一致性： N 表示副本总数， W 表示写操作必须成功返回的副本数， R 表示读操作必须读取的副本数。若满足 $W+R>N$ ，则任意一次读都能至少覆盖上一次写，实现强一致性。例如 $N=3$ 、 $W=2$ 、 $R=2$ 时，任意两个副本的交集保证新值可见。

2 经典协议

2.1 2PC 与 3PC

两阶段提交（2PC）将事务分为「准备」和「提交/回滚」两阶段。协调者首先向所有参与者发送准备请求，参与者本地执行事务但不提交，仅记录 undo/redo 日志并返回「同意」或「拒绝」。若所有参与者同意，协调者发送提交指令；否则发送回滚。2PC 的主要缺陷是协调者宕机导致参与者阻塞，资源长期锁定。

三阶段提交 (3PC) 在准备前增加「CanCommit」阶段，并引入超时机制。若协调者或参与者在超时时间内未收到后续消息，可单方面释放锁，减少阻塞窗口。但 3PC 仍无法完全避免网络分区下的不一致，工程代价更高。

2.2 Paxos 算法

Paxos 将角色分为提议者 (Proposer)、接受者 (Acceptor) 和学习者 (Learner)。在「准备」阶段，提议者选择一个全局唯一提案编号 n ，向多数派接受者发送 $\text{Prepare}(n)$ 。接受者若尚未响应更高编号的 Prepare ，则承诺不再接受编号小于 n 的提案，并返回已接受的最高编号提案及其值。在「接受」阶段，提议者若收到多数派响应，则发送 $\text{Accept}(n, v)$ ， v 取响应中编号最高的提案值或自选值。接受者仅在未承诺更高编号时接受该提案。Multi-Paxos 通过选举稳定领导者，省去重复 Prepare 阶段，提升吞吐。

2.3 Raft 协议

Raft 将节点状态分为领导者 (Leader)、跟随者 (Follower) 和候选人 (Candidate)。领导者通过心跳维持权威，负责处理客户端请求并将日志条目复制到跟随者。日志条目包含任期号 (Term) 和索引，领导者只有在「安全」条件下 (即新日志已被复制到多数派) 才提交。任期号单调递增，用于检测陈旧领导者；日志匹配特性保证相同索引和任期的日志条目内容一致，从而实现状态机安全。

2.4 Zab 协议

Zab (ZooKeeper Atomic Broadcast) 面向 ZooKeeper 的主从架构设计。它通过广播协议确保更新操作按 FIFO 顺序送达所有副本。崩溃恢复时，新领导者首先完成历史同步，再接受新请求，快速恢复对外服务能力。

3 现代演进

3.1 拜占庭容错协议

PBFT 通过三阶段广播实现拜占庭容错，但通信复杂度为 $O(n^2)$ 。HotStuff 将通信复杂度降至 $O(n)$ ，并利用链式结构实现流水线共识，适用于大规模区块链场景。Tendermint 将 BFT 与 PoS 结合，提供即时确定性。区块链中，PoW 通过算力竞争隐式达成一致，PoS 通过权益投票，DAG 则允许并行出块以提升吞吐。

3.2 混合与分层模型

TiDB 采用 Multi-Raft 分区数据，每个 Region 独立选举领导者，由 Placement Driver 统一调度跨地域副本，实现地理分布与强一致。Google Spanner 使用 TrueTime API 获取全球时钟偏差上界，结合 Paxos 实现外部一致性与线性化。

3.3 新硬件加速

RDMA 允许零拷贝、内核旁路的数据传输。协议如 Derecho 利用 RDMA 单边操作实现高吞吐、低延迟共识；HERD 通过内存哈希表加速请求处理，充分发挥硬件特性。

4 工程实践要点

4.1 性能权衡

批量 (Batching) 与流水线 (Pipeline) 可显著提升吞吐, 但会增加尾延迟。并行复制在多副本场景下能隐藏网络抖动, 但需控制并发度以避免带宽过载。系统需在延迟与吞吐之间寻找平衡点。

4.2 存储与网络

预写日志 (WAL) 保证崩溃恢复; 快照 (Snapshot) 与日志压缩 (Log Compaction) 控制存储增长; 背压机制在生产者过快时减缓写入, 保护下游节点。

4.3 监控与运维

关键指标包括当前任期 (Term)、已提交索引 (CommitIndex)、选举超时 (ElectionTimeout)。混沌工程可模拟网络分区、节点宕机与时钟漂移, 验证系统在异常场景下的表现。

4.4 选型 Checklist

评估数据规模、延迟 SLA、运维人力及 CAP 偏好: 若追求强一致且运维资源充足, 可选 Raft 或 Multi-Paxos; 若接受最终一致以换取高可用, 则可考虑 Gossip 或 Dynamo 风格协议。

5 案例研究

5.1 etcd 的 Raft 实现

etcd 使用 Raft 维护键值存储。领导者将客户端写请求封装为日志条目, 复制到多数派后提交并应用到状态机。Follower 通过 gRPC 接收心跳与日志, 异常时触发预投票 (Pre-Vote) 机制, 避免网络抖动导致频繁选举。

5.2 Kafka 的 Leader/Follower 机制

Kafka 的分区在 ISR (In-Sync Replicas) 集合内选举领导者。只有 ISR 内的副本才能被选为领导者, 保证已提交消息不丢失。生产者可配置 acks=all, 要求 ISR 中所有副本确认, 提升持久性。

5.3 TiDB 的 Percolator + Raft 两层一致性

TiDB 使用 Percolator 模型实现分布式事务, 底层由 TiKV 的 Raft 保证单 Region 强一致。事务提交时, 先写入锁和数据, 再异步清理锁; Raft 确保锁和数据原子落盘, 从而实现快照隔离。

一致性协议从 2PC 的阻塞式提交, 演进到 Paxos、Raft 等多数派表决机制, 再到 BFT 与混合模型, 持续在正确性、性能与运维之间寻找最优解。云原生与 Serverless 场景对低延迟、弹性伸缩提出更高要求。未来的突破可能来自形式化验证工具、硬件加速协议以及基于强化学习的智能调度策略, 进一步降低运维门槛并提升系统鲁棒性。