分布式理论

CAP

一致性（Consistency）

含义：一致性说的是客户端的每次读操作，不管访问哪个节点，要么读到的都是同一份最新写入的数据，要么读取失败。

一致性强调的是数据正确。

可用性（Availability）

含义：可用性说的是任何来自客户端的请求，不管访问哪个非故障节点，都能得到响应数据，但不保证是同一份最新数据

强调的是服务可用，但不保证数据正确

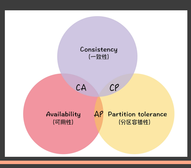
分区容错性（Partition Tolerance）

含义：当节点间出现任意数量的消息丢失或高延迟的时候，系统仍然在继续工作。也就是说，分布式系统在告诉访问本系统的客户端：不管我的内部出现什么样的数据同步问题，我会一直运行

强调的是集群对分区故障的容错能力

CAP 不可能三角

对于一个分布式系统而言，一致性（Consistency）、可用性（Availability）、分区容错性（Partition Tolerance）3 个指标不可兼得，只能在 3 个指标中选择 2 个。



只要有网络交互就一定会有延迟和数据丢失，而这种状况我们必须接受，还必须保证系统不能挂掉，所以分区容错性（Partition Tolerance）是必须的，那么就只能在CP、AP中选择

如果读操作会读到旧数据，影响到了系统运行或业务运行（也就是说会有负面的影响），推荐选择 C，否则选 A。

模型

CP

采用 CP 模型的分布式系统，舍弃了可用性，一定会读到最新数据，不会读到旧数据

一旦因为消息丢失、延迟过高发生了网络分区，就影响用户的体验和业务的可用性（比如基于 Raft 的强一致性系统，此时可能无法执行读操作和写操作）

典型的应用是 Etcd，Consul 和 Hbase

AP

采用 AP 模型的分布式系统，舍弃了一致性，实现了服务的高可用

用户访问系统的时候，都能得到响应数据，不会出现响应错误，但会读到旧数据

典型应用就比如 Cassandra 和 DynamoDB

ACID

概念

- 原子性(Automicity)
- 一致性(Consistency)
- 隔离性(isolation)
- 持久性(durability)

分布式事务实现基础理论

二阶段提交协议

通过二阶段的协商来完成一个提交操作

协商需要一个协调者（Coordinator）

大致流程

- 提交请求阶段（又称投票阶段）
  - 在第一个阶段，每个参与者投票表决事务是放弃还是提交，一旦参与者投票要求提交事务，那么就不允许放弃事务
  - 在一个参与者投票要求提交事务之前，它必须保证能够执行提交协议中它自己那一部分，即便参与者出现故障或者中途被替换掉。
- 提交执行阶段（又称完成阶段）
  - 要么全部执行，要么全部放弃
  - 事务的每个参与者执行最终统一的决定，提交事务或者放弃事务。这个约定，是为了实现 ACID 中的原子性。

二阶段提交协议最早是用来实现数据库的分布式事务的，不过现在最常用的协议是 XA 协议。这个协议是 X/Open 国际联盟基于二阶段提交协议提出的，也叫作 X/Open Distributed Transaction Processing（DTP）模型，比如 MySQL 就是通过 MySQL XA 实现了分布式事务。

存在的问题

- 在提交请求阶段，需要预留资源，在资源预留期间，其他人不能操作（比如，XA 在第一阶段会将相关资源锁定）；
- 数据库是独立的系统。

因为这两点，我们无法根据业务特点弹性地调整锁的粒度，而这都会影响数据库的并发性能

TCC（Try-Confirm-Cancel）

TCC 是 Try（预留）、Confirm（确认）、Cancel（撤销）3 个操作的简称，它包含了预留、确认或撤销这 2 个阶段

大致流程

- 先进入预留阶段
- 预留阶段执行没问题，进入确认阶段
- 预留阶段执行出错，进入撤销阶段

TCC 本质上是补偿事务，它的核心思想是针对每个操作都要注册一个与其对应的确认操作和补偿操作（也就是撤销操作）

它是一个业务层面的协议，你也可以将 TCC 理解为编程模型，TCC 的 3 个操作是需要在业务代码中编码实现的，为了实现一致性，确认操作和补偿操作必须是等幂的，因为这 2 个操作可能会失败重试

TCC 不依赖于数据库的事务，而是在业务中实现了分布式事务，这样能减轻数据库的压力，但对业务代码的入侵性也更强，实现的复杂度也更高

ACID 特性理解为 CAP 中一致性的边界，最强的一致性也就是 CAP 的服（Acid）

BASE

BASE 理论是 CAP 理论中的 AP 的延伸，是对互联网大规模分布式系统的实践总结，强调可用性

核心就是基本可用（Basically Available）和最终一致性（Eventually consistent）

基本可用

基本可用在本质上是一种妥协，也就是在出现节点故障或系统过载的时候，通过牺牲非核心功能的可用性，保障核心功能的稳定运行

实现基本可用的四板斧

- 流量削峰：不同地区售票时间错峰出售
- 延时响应：买票排队，基于队列先收到用户买票请求，排队异步处理，延迟响应
- 体验降级：看到非实时数据，可以采用缓存数据提供服务
- 过载保护：熔断/限流，直接拒绝掉一部分请求，或者当请求队列满了，移除一部分请求，保证整体系统可用

最终一致性

最终一致性是说，系统中所有的数据副本在经过一段时间的同步后，最终能够达到一个一致的状态。也就是说，在数据一致性上，存在一个短暂的延迟。

实现方式

- 读时修复
  - 在读取数据时，检测数据的不一致，进行修复
  - 比如 Cassandra 的 Read Repair 实现，具体来说，在向 Cassandra 系统查询数据的时候，如果检测到不同节点的副本数据不一致，系统就自动修复数据。
- 写时修复
  - 在写入数据，检测数据的不一致时，进行修复
  - 比如 Cassandra 的 Hinted Handoff 实现。具体来说，Cassandra 集群的节点之间远程写数据的时候，如果写失败就将数据缓存下来，然后定时重传，修复数据的不一致性。
- 异步修复
  - 这个是最常用的方式，通过定时对账检测副本数据的一致性，并修复

ACID 理论和BASE理论的对比

ACID 理论是传统数据库常用的设计理念，追求强一致性模型

BASE 理论支持的是大型分布式系统，通过牺牲强一致性获得高可用性