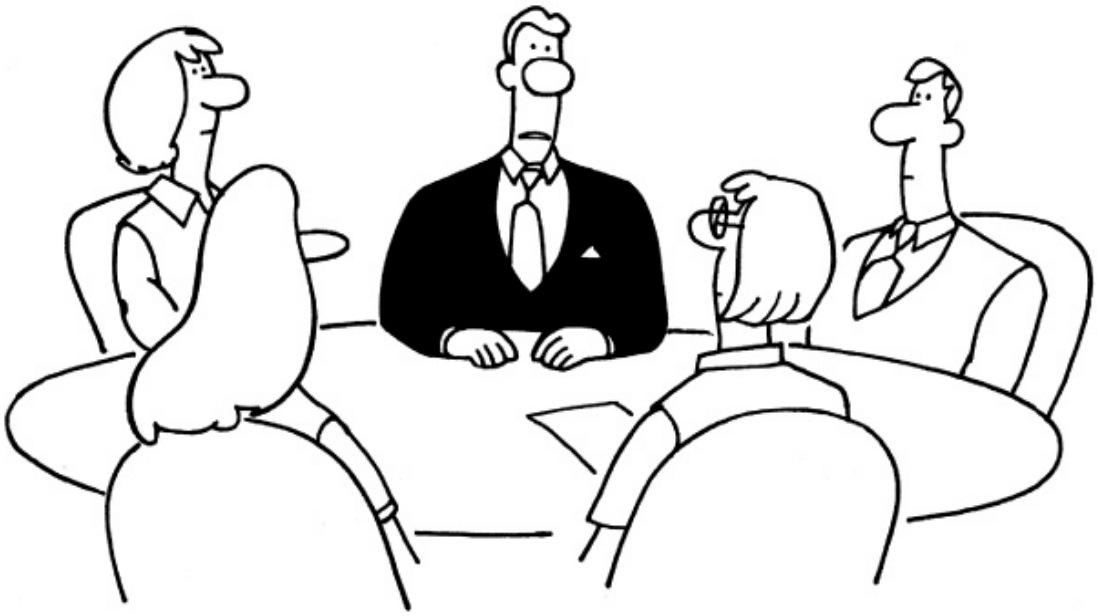
分布式系统一致性问题、CAP定律以及 BASE 理论

一致性问题

在介绍分布式系统一致性问题之前，我们先来了解一下副本概念。分布式系统会存在许多异常问题，比如机器宕机；为了提供高可用服务，一般会将数据或者服务部署到很多机器上，这些机器中的数据或服务可以称为副本。如果其中任何一台节点出现故障，用户可以访问其他机器上的数据或服务。由于副本的存在，如何保证这些节点上的副本数据或服务一致性，是整个分布式系统需要解决的核心问题，这也就是本文提到的一致性问题。



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

一致性按照不同的角度可以分为客户端以及系统
。从客户端角度看，就是客户端读写操作是否符合某种特性；从系统角度看，就是系统更新如何复制分布到整个系统，以保证数据最终一致。

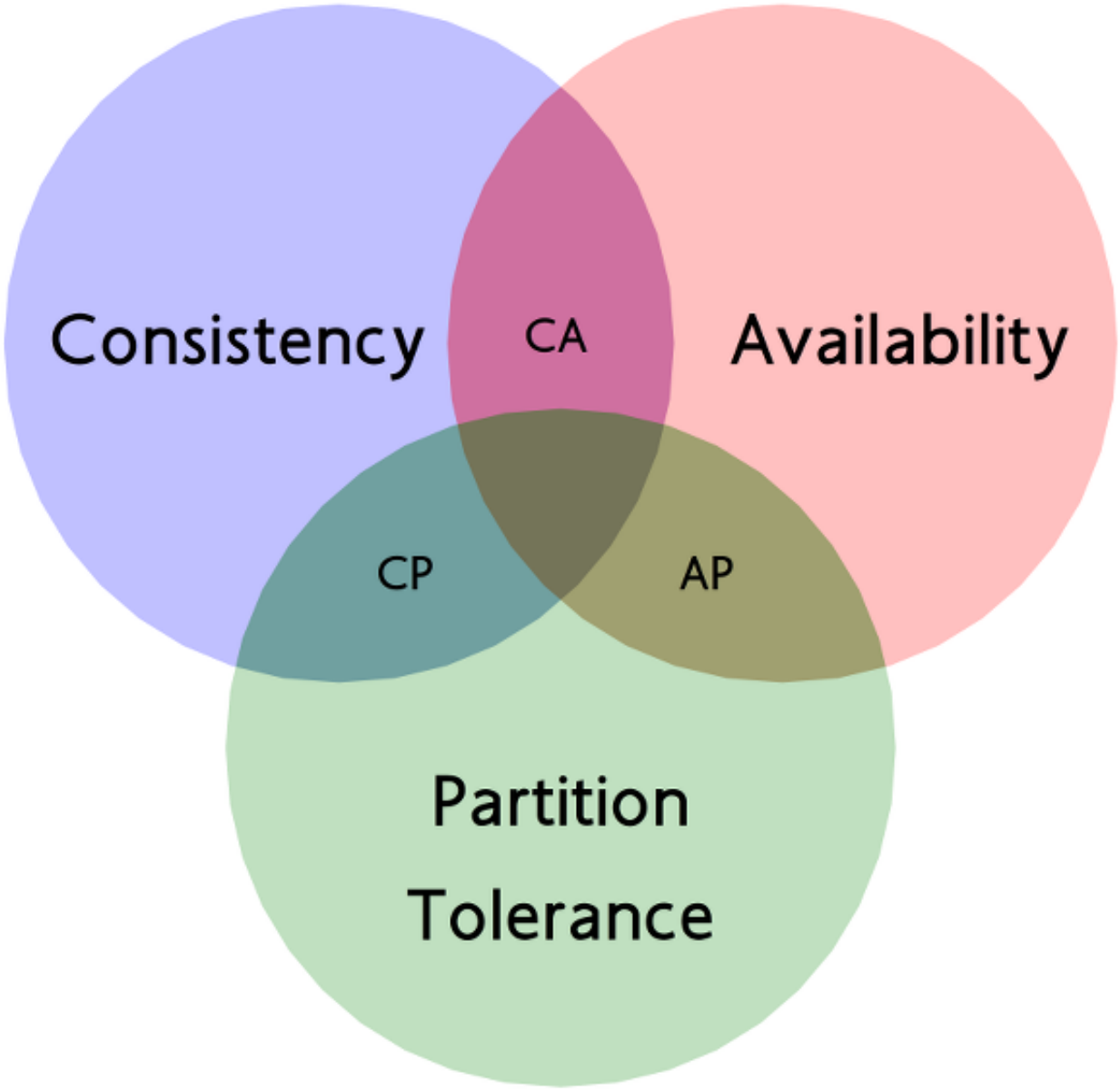
如果从客户端角度看，一致性又可以分为以下几种：

- 强一致性（Strong Consistency）：在任何时候，用户或节点都可以读到最近一次成功更新的副本数据。这种一致性肯定是我们最想要的，但是很遗憾，强一致性在实践中很难实现，而且一般都会牺牲可用性。
- 弱一致性（Weak Consistency）：某个进程更新了副本的数据，但是系统不能保证后续进程能够读取到最新的值。
- 最终一致性（Eventual Consistency）：最终一致性是弱一致性的一种特例。某个A 进程更新了副本的数据，如果没有其他进程更新这个副本的数据，系统最终一定能够保证后续进程能够读取到A进程写进的最新值。但是这个操作存在一个不一致性的窗口，也就是 A 进程写入数据，到其他进程读取 A 写进去的值所用的时间。最终一致性又可以根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为以下几种：
    1. 因果一致性（Causal Consistency）：如果进程A通知进程B它已更新了一个数据项，那么进程B的后续访问将返回更新后的值，且一次写入将保证取代前一次写入。与进程A无因果关系的进程C的访问遵守一般的最终一致性规则。
    2. 读写一致性（Read-Your-Writes Consistency）：当进程A自己更新一个数据项之后，它总是访问到更新过的值，绝不会看到旧值。这是因果一致性模型的一个特例。
    3. 会话一致性（Session Consistency）：这是上一个模型的实用版本，它把访问存储系统的进程放到会话的上下文中。只要会话还存在，系统就保证读写一致性。如果由于某些失败情形令会话终止，就要建立新的会话，而且系统的保证不会延续到新的会话。
    4. 单调读一致性（Monotonic Read Consistency）：如果进程已经看到过数据对象的某个值，那么任何后续访问都不会返回在那个值之前的值。
    5. 单调写一致性（Monotonic Write Consistency）：系统保证来自同一个进程的写操作顺序执行。

当然，还存在其他的一些一致性变种，这里就不再敖述。

CAP 定律

2000 年 7月，来自加州大学伯克利分校的 Eric Brewer 教授提出了著名的 CAP 猜想。2年后，来自麻省理工学院的 Seth Gilbert 和 Nancy Lynch 从理论上证明了 CAP 可行性，从此 CAP 定理在学术上成为了分布式计算领域公认的定理，影响着分布式计算的发展。



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

CAP原则包含如下三个元素：

- C（Consistency）：一致性。在分布式系统中的所有数据备份，在同一时刻具有同样的值，所有节点在同一时刻读取的数据都是最新的数据副本。
- A（Availability）：可用性，好的响应性能。完全的可用性指的是在任何故障模型下，服务都会在有限的时间内处理完成并进行响应。
- P（Partition tolerance）：分区容忍性。尽管网络上有部分消息丢失，但系统仍然可继续工作。

CAP 原理证明，任何分布式系统只可同时满足以上两点，无法三者兼顾。由于关系型数据库是单

节点无复制的，因此不具有分区容忍性，但是具有一致性和可用性；而分布式的服务化系统都需要满足分区容忍性，那么我们必须在一致性和可用性之间进行权衡。如果在网络上有消息丢失，也就是出现了网络分区，则复制操作可能会被延后，如果这时我们的使用方等待复制完成再返回，则可能导致在有限时间内无法返回，就失去了可用性；而如果使用方不等待复制完成，而在主分片写完后直接返回，则具有了可用性，但是失去了一致性。因此，系统架构师需要把精力放在如何根据业务在 C 和 A 之间进行选择。

BASE 理论

BASE 是 Basically Available（基本可用）、Soft state（软状态）和 Eventually consistent（最终一致性）三个短语的简写，由 eBay 架构师 Dan Pritchett 于 2008 年在《BASE: An Acid Alternative》（论文地址点 这里）论文中首次提出。BASE 思想与 ACID 原理截然不同，它满足 CAP 原理，通过牺牲强一致性获得可用性， 一般应用于服务化系统的应用层或者大数据处理系统中，通过达到最终一致性来尽量满足业务的绝大多数需求。
BASE 模型包含如下三个元素：

- BA：（Basically Available ），基本可用。
- S：（ Soft State），软状态，状态可以在一段时间内不同步。
- E：（Eventually Consistent ），最终一致，在一定的时间窗口内，最终数据达成一致即可。

关于最终一致的几种变种参见上面，在实际系统实践中，可以将若干变种结合起来，来实现各种业务需求。