24 | 尺有所短，寸有所长：CAP和数据存储技术选择

2019-11-04 四火

全栈工程师修炼指南

进入课程 >

讲述：四火

时长 21:18　大小 14.64M

你好，我是四火。

在上一讲中我们着重讲了持久层的一致性，其实，它是分布式系统的一个基础理论。你可能会问，学习基于 Web 的全栈技能，也需要学习一些分布式系统的技术吗？是的！特别是我们在学习其持久层的时候，我们还真得学习一些分布式系统的基础理论，从而正确理解和使用我们熟悉的这些持久层技术。

CAP 理论就是分布式系统技术中一个必须要掌握的内容，也是在项目早期和设计阶段实实在在地影响我们技术选型、技术决策的内容。

理解概念

我想，你已经很熟悉一致性了。今天，在一致性之后，我们也要涉及到 CAP 的另外的两个方面——可用性和分区容忍性。

1. CAP 的概念

CAP 理论，又叫做布鲁尔理论（Brewer’ s Theorem），指的是在一个共享数据的分布式存储系统中，下面三者最多只能同时保证二者，对这三者简单描述如下：

一致性（Consistency）：读操作得到最近一次写入的数据（其实就是上一讲我们讲的强一致性）；

可用性（Availability）：请求在限定时间内从非失败的节点得到非失败的响应；

分区容忍性（Partition Tolerance）：系统允许节点间网络消息的丢失或延迟（出现分区）。

下面，请让我进一步说明，从而帮助你理解。

一致性，这里体现了这个存储系统对统一数据提供的读写操作是线性化的。如果客户端写入数据，并且写操作返回成功给客户端，那么在下一次读取的时候（下一次写入以前），如果系统返回了“非失败”的响应，就一定是读出了完整、正确（最新）的那份数据，而不会读取到过期数据，也不会读取到中间数据。

可用性，体现的是存储系统持续提供服务的能力，这里表现在两个方面：

返回“非失败”的响应，就是说，不是光有响应就可以了，系统得是在实实在在地提供服务，而不是在报错；

在限定时间内返回，就是说，这个响应是预期时间内返回的，而不出现请求超时。

请注意，这里说的是“非失败”响应，而并没有说“正确”的响应。也就是说，返回了数据，但可以是过期的，可以是中间数据，因为数据是否“正确”并非由可用性来保证，而是由一致性来保证的。系统的单个节点可能会在任意时间内故障、出错，但是系统总能够靠处于非失败（non-failing）状态的其它节点来继续提供服务，保证可用性。

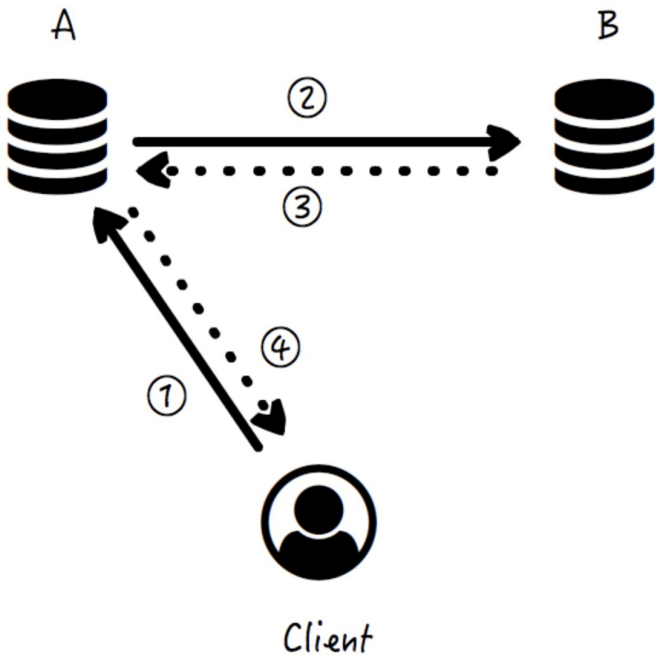
分区容忍性，体现了系统是否能够接纳基于数据的网络分区。只要出现了网络故障，无论什么原因导致某个节点和系统的其它节点失去了联系，节点间的数据同步操作无法被“及

时”完成，那么，即便它依然可以对外（客户端）提供服务，网络分区也已经出现了。

当然，如果数据只有一份，不存在其它节点保存的副本，或不需要跨节点的数据共享，那么，这就不存在“分区”，这样的分布式存储系统也就不是 CAP 关心的对象。

2. 进一步理解

如果你觉得模糊，没关系，让我使用一个简单的图示来帮你理解。



有这样一个存储系统，存在两个节点 A 和 B，各自存放一份数据拷贝。那么在正常情况下，客户端无论写数据到 A 还是 B，都需要将数据同步到另一个节点，再返回成功。比如图示中带序号的四个箭头：

箭头 ①，客户端写数据到节点 A；

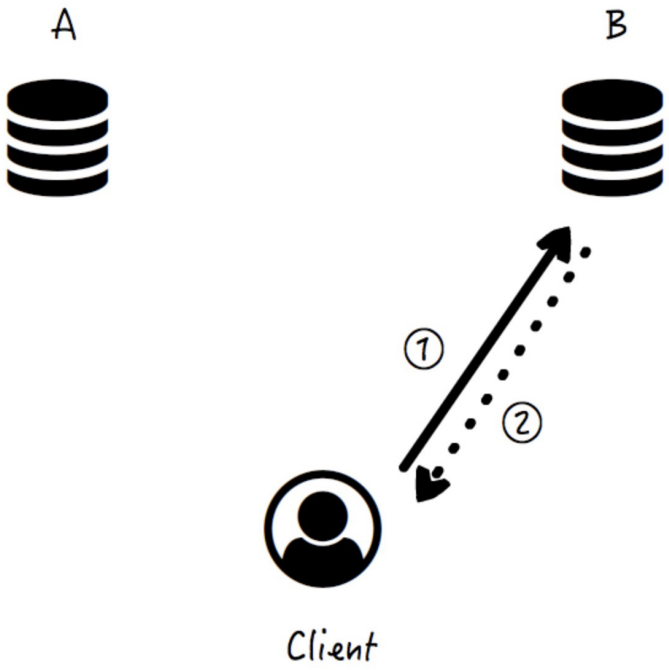
箭头 ②，节点 A 同步数据变更到节点 B；

箭头 ③，节点 B 返回成功响应到节点 A；

箭头 ④，节点 A 返回成功响应给客户端。

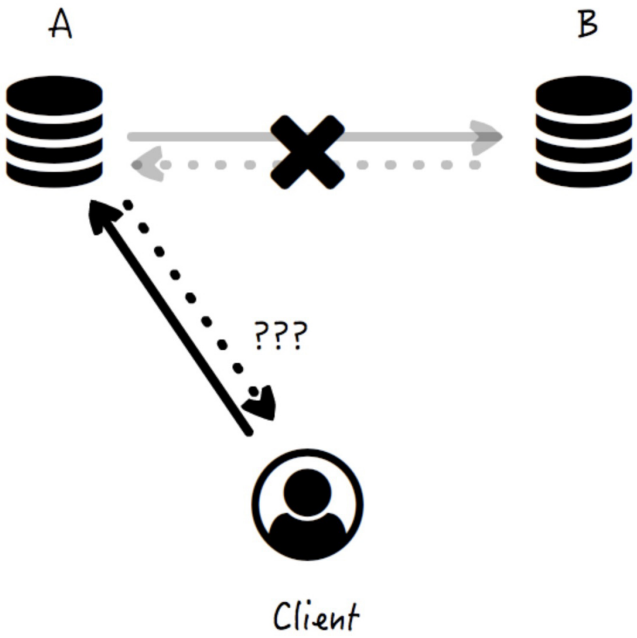
不知道你有没有回想起 [第 23 讲] 中的 Multi-Master 架构，对，但唯一需要特别指出的不同是，节点间数据拷贝是同步进行的，需要完成拷贝以后再返回响应，因为我们需要保证一致性。

之后，客户端尝试读取刚写入的数据，无论是从节点 A 还是 B，都可以得到准确的数据：



好，这种情况下数据在 A、B 上都是一致的，并且系统也是可用的。

但是，现在网络突然出现故障，A 和 B 之间的数据拷贝通道被打断了，也就是说，分区发生了。这时候客户端再写入 A 就会出现以下情况：



你看，这时候节点 A 已经无法将数据“及时”同步到节点 B 了， 那么，节点 A 是否应该将数据写入自己，并返回“成功”给客户端呢？它陷入了两难：

如果写入并返回成功，满足系统的可用性，就意味着丢失了数据一致性。因为节点 A 的数据是最新的，而节点 B 的数据是过期的。

如果不写入数据，而直接返回失败，即节点 A 拒绝写操作，那么 A 和 B 节点上的**数据依然满足一致性（写入失败，但依然都是相互一致的老数据），但是整个系统失去了可用性。**

你看，我们怎么也无法同时保证一致性、可用性和分区容忍性这三者。

3. 三选二？

紧接着我要谈一谈对于 CAP 理论一个很大的误解——三选二。从上面对于 CAP 的描述来看， CAP 的应用似乎就是一个三选二的选择题，但事实上，完全不是这样的。

开门见山地说，**在讨论 CAP 定理的时候，P，也就是分区容忍性，是必选项。**具体来说，跨区域的系统，分区容忍性往往是不可以拿掉的，因为无论是硬件损坏、机房断电，还是地震海啸，都是无法预料、无法避免的，任何时间都可能出现网络故障而发生分区，因此工程师能做的，就是从 CP 和 AP 中选择合适的那一个。

你可以想想上面我拿图示举的那个例子，在分区发生的时候，最多只能保证一致性和可用性一个。也就是说，CAP 理论不是三选二的，而是二选一，当然，具体选哪个，我们需要“权衡”。

需要特别说明的是，**这里说的是只能“保证”一致性和可用性二者之一，而不是说，在系统正常运行时，二者不可能“同时满足”。**在系统运行正常的时候，网络分区没有出现，那么技术上我们是可能同时满足一致性和可用性两者的。

这时你可能会问，难道没有 CA，即同时“保证”一致性和可用性，而牺牲掉分区容忍性的系统吗?

有！**但请注意，那其实已经不是 CAP 理论关心的对象了，因为 CAP 要求的是节点间的数据交换和数据共享。**任何时候都不会有分区发生，这种系统基本上有这样两种形式：

单节点系统，这很好理解，没有节点间数据的交换，那么无论网络出不出故障，系统始终只包含一个节点。比方说，传统的关系型数据库，像 MySQL 或者 OracleDB，在单节点的配置下。

虽然是多节点，但是节点间没有数据共享和数据交换——即节点上的数据不需要拷贝到其它节点上。比方说，无数据副本（Replica）配置的集群 Elasticsearch 或 Memcached，在经过 hash 以后，每个节点都存放着单份不同的数据。这种情况看起来也算分布式存储，但是节点之间是互相独立的。

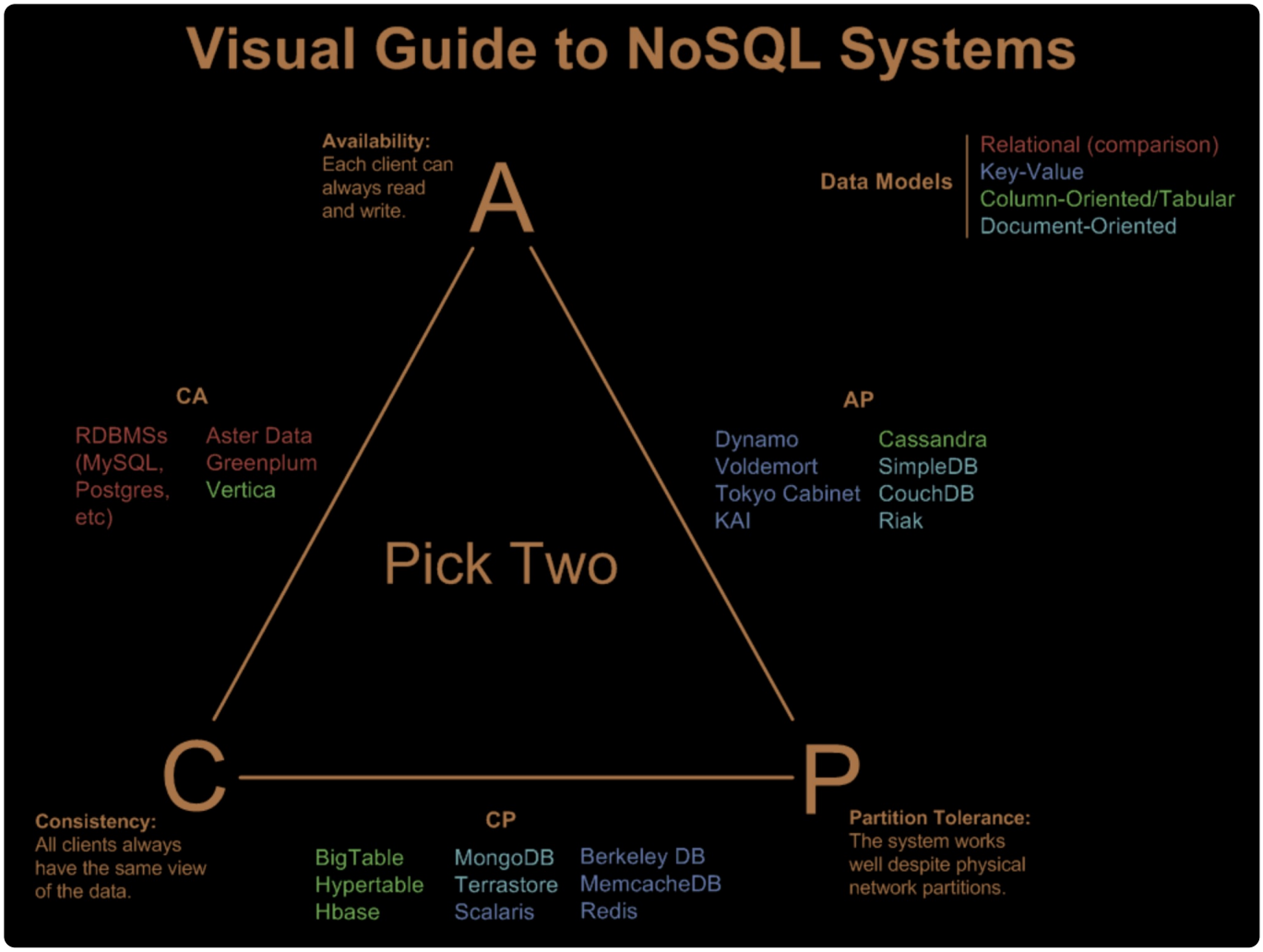
存储技术的选择：NoSQL 三角形

在谈到根据 CAP 来选择技术的时候，我想先来介绍一下 NoSQL，你将会看到它们大量地分布在下面“NoSQL 三角形”的 CP 和 AP 两条边上。

那么，到底什么是 NoSQL 呢？我们可以简单地认为，NoSQL 是“非关系数据库”，和它相对应的是传统的“关系数据库”。它被设计出来的目的，并非要取代关系数据库，而是成为关系数据库的补充，即“Not Only SQL”。

也就是说，它放弃了对于“关系”的支持，**损失了强结构定义和关系查询等能力，但是它往往可以具备比关系数据库高得多的性能和横向扩展性（scalability）等优势。**这在 Web 2.0 时代对于一些关系数据库不擅长的场景，例如数据量巨大，数据之间的关联关系较弱，数据结构（schema）多变，强可用性要求和低一致性要求等等，NoSQL 可以发挥其最大的价值。

在实际业务中，我们可以利用 CAP 定理来权衡和帮助选择合适的存储技术，且看下面这张 NoSQL 系统的 CAP 三角形（来自 Visual Guide to NoSQL Systems）。尺有所短，寸有所长，我们可以从 CAP 的角度来理解这些技术的优劣。



从图中可以发现，关系数据几乎都落在了 CA 一侧，但是请注意，技术也在不断更新，许多关系数据库如今也可以通过配置而形成其它节点的数据冗余；有时，我们则是在其上方自己实现数据冗余，比如配置数据库的数据同步到备份数据库。

无论哪一种方法，一旦其它节点用于数据冗余的数据副本出现，这个存储系统就落到上述三角形的另外两边去了。

云上的 NoSQL 存储服务，多数落在了 AP 一侧，这也和 NoSQL 运动可用性优先保证而降级一致性的主题符合。比如 Amazon 的 DynamoDB，但是这个也是可以通过不同的设置选项来改变的，比如 DynamoDB 默认采用最终一致性，但也允许配置为强一致性，那时它就落到了 CP 上面。

实际场景

接着我们考虑几个实际应用场景，看看该采用哪一条边的技术呢？既然是基于 Web 的全栈工程师的技术学习，我就来举两个基于网站应用的例子。

还记得我们在 [第 09 讲] 中谈到的页面聚合吗？对于门户网站来说，无论是显示的数据，还是图片、样式等等静态资源，通过 CDN 的方式，都可以把副本存放在离用户较近的节点，这样它们的获取可以减少延迟，提高用户体验。因此，这些系统联合起来，就形成了一个可以使用 CAP 讨论的分布式系统。

那么，很容易理解的是，且不用说网络故障而发生分区的情况，即便在正常情况下，这些信息并不需要具备那么严格的“即时性”，新闻早显示、晚显示几秒钟，乃至几分钟，都不是什么问题，上海的读者比北京的读者晚看到一会儿，也不是什么问题。但是，大型网站页面打不开，就是一个问题了，这显然会影响用户的体验。因此，从这个角度说，我们可以牺牲一致性，但需要尽量保证可用性，因此这是一个选择 AP 的例子。

事实上，对于大型的系统而言，我们往往不需要严格的一致性，但是我们希望保证可用性，因此在大多数情况下我们都会选择 AP。但是，有时情况却未必如此。

再举一个例子，航空公司卖机票，在不考虑超售的情况下，一座一票，航空公司的网站当然可以采用上面类似的做法；有时，甚至在正常的情况下，余票的显示都可以不是非常准确的（比如显示“有票”可以避免显示这个具体数字）。但是，当客户真正在选座售票的时候，即扣款和出票的时候就不是这样的了，一致性必须优先保证。因为如果可用性保证不了，即有时候订票失败，用户最多也就是牢骚几句，这还可以接受，但要是出现一致性问题，即两个人订了同一个座位的票，那就是很严重的问题了。

最后，我想说的是，这里的选择是一个带有灰度的过程，并非只有 0 和 1 这两个绝对的答案，我们还是需要具体问题具体分析，不要一刀切。

从特性上说，甚至可以部分特性做到 CP，部分做到 AP，这都是有可能的。比如说，涉及钱的问题一定是 CP 吗？不一定，ATM 机就是一个很经典的例子，在网络故障发生时，ATM 会处于 stand-alone 模式，在这种模式下，用户依然可以执行查询余额等操作（很可能数额不准确），甚至还可以取款，但是这时的取款会有所限制，例如限制一个额度（银行承担风险），或者是限制只能给某些银行的卡取款，毕竟可用性和一致性的丢失会带来不同的风险和后果，两害相权取其轻。

总结思考

今天我们学习和理解了 CAP 理论，并且了解了一些实际应用的例子。希望你能够通过今天的内容，彻底掌握其原理，并能够逐渐在设计中应用起来，特别是在技术选型做“权衡”的

时候。

现在，我们来看一下今天的思考题吧：

> 你是否了解或是接触过分布式系统，特别是分布式存储系统，它是否能归类到 NoSQL 三角形中的某一条边上呢？
>
> 互联网上的绝大多数系统都是可以牺牲一致性，而优先保证可用性的，但也有一些例外。你能举出几个即便牺牲可用性，也要保证数据一致性的例子来吗？

今天的主要内容就到这里，欢迎你在留言区进行讨论，也欢迎你继续学习下面的选修课堂。

选修课堂：从 ACID 到 BASE

ACID 和 BASE，正好是英文里“酸”和“碱”的意思。有意思的是，关系数据库和非关系数据库，它们各自的重要特性，也恰恰可以用酸和碱来体现。下面我来简单做个比较，你可以从中感受一下二者的差异和对立性，为我们后两讲介绍技术选型打下基础。

先说说 ACID。

关系数据库的一大优势，就是可以通过事务的支持来实现强一致性，而事务，通常可以包含这样几个特性。

> Atomicity：原子性，指的是无论事务执行的过程有多么复杂，要么提交成功改变状态，要么提交失败回滚到提交前的状态，这些过程是原子化的，不存在第三种状态。
>
> Consistency：一致性，这里的一致性和我们前面介绍的一致性含义略有不同，它指的是事务开始前、结束后，数据库的完整性都没有被破坏，所有键、数据类型、检查、触发器等等都依然有效。
>
> Isolation：隔离性，指的是多个并发事务同一时间对于数据进行读写的能力，同时执行，互不影响。事务隔离分为四大级别，不同的数据库默认实现在不同的级别，我在扩展阅读中放置了一些学习材料，感兴趣的话可以进一步学习。
>
> Durability：持久性，一旦事务成功提交，那么改变是永久性的。

接着说说 BASE。

前面已经谈到了 NoSQL，CAP、最终一致性，再加上 BASE，被称作 NoSQL 的三大基石。而 BASE，是基于 CAP 衍生出来，对于其牺牲一致性和保证可用性的这一分支，落实到具体实践中的经验总结，在大规模互联网分布式系统的设计中具有指导意义。

BA：基本可用，即 Basically Available。这就是说，为了保障核心特性的“基本可用”，无论是次要特性的功能上，还是性能上，都可以牺牲。严格说来，这都是在“可用性”方面做的妥协。例如电商网站在双十一等访问压力较大的期间，可以关闭某一些次要特性，将购物支付等核心特性保证起来。如果你还记得 [第 17 讲] 中的“优雅降级”，那么你应该知道，这里说的就是优雅降级中一个常见的应用场景。

S：软状态，即 Soft State。说的是允许系统中的数据存在中间状态，这也一样，为了可用性，而牺牲了一致性。

E：最终一致性，即 Eventually Consistent。S 和 E 两点其实说的是一个事情，一致性的牺牲是可行且有限度的，某个数据变更后的时间窗口内出现了不一致的情况，但是之后数据会恢复到一致的状态。举例来说，上文提到过的 CDN 系统便是如此，再比如社交媒体发布后的互动，像点赞、评论等功能，这些数据可以延迟一会儿显示，但是超过了一定的时间窗口还不同步到就会是问题。

扩展阅读

Towards Robust Distributed Systems，这是一个胶片，来自 Eric Brewer 最早谈及 CAP 理论的一个分享；而 Brewer’s CAP Theorem，这一篇是对 CAP 理论证明的论文，想看中文的话可以看看这篇中文译文。

文中提到了 NoSQL 的概念，CAP 的三角形一图中也有一些实现的例子，NoSQL Databases 这个网站列出了比较全面的 NoSQL 数据库列表，可供查询。

文中提到了某些存储服务能够通过配置在 CAP 的三角形上切换。比如 DynamoDB，它是一个 NoSQL 的键 / 值文档数据库，就可以配置为 CP，也可以配置为 AP，官方的读取一致性这篇文章做了简要说明；再比如 S3，它是一个云上的对象存储服务，它的一致性根据对象创建和对象修改而有所不同，你可以看一下官方的这个说明。

如果对 BASE 感兴趣，你可以看看这篇最原始的 Base: An Acid Alternative，想看中文译文的话可以看看这篇。

对于文中提到的事务隔离，感兴趣可以进一步参见维基百科，还有一篇美团技术团队写的 Innodb 中的事务隔离级别和锁的关系，也是很好的针对事务隔离的学习材料。

上一篇　23 | 知其然，知其所以然：数据的持久化和一致性

精选留言 (1)

写留言

tt
2019-11-04

我在某银行的省级分行工作。大体上，账务核心是全国集中的，标准银行业务从粗力度上看直接与核心发生交互，而对于分省的特色业务，是通过分行接入核心进行账务交易，分行系统与三方商户进行交互和资金清算。分行会记录金融交易的状态，商户也会记录状态。
从记录金融交易状态这一点来看，总行核心、分行交易系统、商户IT系统算是构成了一个...
展开

作者回复: 说得很好 :) 感谢。

即便如银行，在一些出现网络分区的场景下依然优先保证可用性，而可以牺牲一致性。