04 | BASE理论：CAP的碱，追求可用性

2020-02-19 韩健

讲述：于航
时长 12:52 大小 10.32M

你好，我是韩健。

很多同学可能喜欢使用事务型的分布式系统，或者是强一致性的分布式系统，因为使用起来很方便，不需要考虑太多，就像使用单机系统一样。但是学了 CAP 理论后，你肯定知道在分布式系统中要实现强一致性必然会影响可用性。比如，在采用两阶段提交协议的集群系统中，因为执行提交操作，需要所有节点确认和投票。

所以，集群的可用性是每个节点可用性的乘积，比如，假设 3 个节点的集群，每个节点可用性为 99.9%，那么整个集群的可用性为 99.7%，也就是说，每个月约宕机 129.6 分钟，**这是非常严重的问题。** 而解决可用性低的关键在于，根据实际场景，尽量采用可用性优先的 AP 模型。

讲到这儿，可能会有一些同学“举手提问”：这也太难了，难道没有现成的库或者方案，来实现合适的 AP 模型？是的，的确没有。因为它是一个动态模型，是基于业务场景特点妥协折中后设计实现的。不过，你可以借助 BASE 理论帮助你达成目的。

在我看来，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。几乎所有的互联网后台分布式系统都有 BASE 的支持，这个理论很重要，地位也很高。一旦掌握它，你就能掌握绝大部分场景的分布式系统的架构技巧，设计出适合业务场景特点的、高可用性的分布式系统。

而它的核心就是基本可用（Basically Available）和最终一致性（Eventually consistent）。也有人会提到软状态（Soft state），在我看来，软状态描述的是实现服务可用性的时候系统数据的一种过渡状态，也就是说不同节点间，数据副本存在短暂的不一致。你只需要知道软状态是一种过渡状态就可以了，我们不多说。

那么基本可用以及最终一致性到底是什么呢？你又如何在实践中使用 BASE 理论提升系统的可用性呢？这些就是本节课的重点了，而我建议你集中注意力，认真学习本节课的内容，学以致用，将 BASE 理论应用到日常工作中。

实现基本可用的 4 板斧

在我看来，基本可用是说，当分布式系统在出现不可预知的故障时，允许损失部分功能的可用性，保障核心功能的可用性。就像弹簧一样，遇到外界的压迫，它不是折断，而是变形伸缩，不断适应外力，实现基本的可用。

具体说的话，你可以把基本可用理解成，当系统节点出现大规模故障的时候，比如专线的光纤被挖断、突发流量导致系统过载（出现了突发事件，服务被大量访问），这个时候可以通过服务降级，牺牲部分功能的可用性，保障系统的核心功能可用。

就拿 12306 订票系统基本可用的设计为例，这个订票系统在春运期间，因为开始售票后先到先得的缘故，会出现极其海量的请求峰值，如何处理这个问题呢？

咱们可以在不同的时间，出售不同区域的票，将访问请求错开，削弱请求峰值。比如，在春运期间，深圳出发的火车票在 8 点开售，北京出发的火车票在 9 点开售。**这就是我们常说的流量削峰。**

另外，你可能已经发现了，在春运期间，自己提交的购票请求，往往会在队列中排队等待处理，可能几分钟或十几分钟后，系统才开始处理，然后响应处理结果，**这就是你熟悉的延迟响应。** 你看，12306 订票系统在出现超出系统处理能力的突发流量的情况下，会通过牺牲响应时间的可用性，保障核心功能的运行。

而 12306 通过流量削峰和延迟响应，是不是就实现了基本的可用呢？现在它不会再像最初的时候那样，常常 404 了吧？

再比如，你正负责一个互联网系统，突然出现了网络热点事件，好多用户涌进来，产生了海量的突发流量，系统过载了，大量图片因为网络超时无法显示。那么这个时候你可以通过哪些方法，保障系统的基本可用呢？

相信你马上就能想到体验降级， 比如用小图片来替代原始图片，通过降低图片的清晰度和大小，提升系统的处理能力。

然后你还能想到过载保护， 比如把接收到的请求放在指定的队列中排队处理，如果请求等待时间超时了（假设是 100ms），这个时候直接拒绝超时请求；再比如队列满了之后，就清除队列中一定数量的排队请求，保护系统不过载，实现系统的基本可用。

你看，和 12306 的设计类似，只不过你负责的互联网系统是通过牺牲部分功能的可用性，保障核心功能的运行。

我说了这么多，主要是想强调：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。

我希望你能在后续的分布式系统的开发中，**不仅掌握流量削峰、延迟响应、体验降级、过载保护这 4 板斧**，更能理解这 4 板斧背后的妥协折中，从而灵活地处理不可预知的突发问题。

带你了解了基本可用之后，我再来说说 BASE 理论中，另一个非常核心的内容：最终一致性。

最终的一致

在我看来，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。

几乎所有的互联网系统采用的都是最终一致性，只有在实在无法使用最终一致性，才使用强一致性或事务，比如，对于决定系统运行的敏感元数据，需要考虑采用强一致性，对于与钱有关的支付系统或金融系统的数据，需要考虑采用事务。

你可以将强一致性理解为最终一致性的特例，也就是说，你可以把强一致性看作是不存在延迟的一致性。**在实践中，你也可以这样思考：** 如果业务的某功能无法容忍一致性的延迟（比如分布式锁对应的数据），需要实现的是强一致性；如果能容忍短暂的一致性的延迟（比如 QQ 状态数据），就可以考虑最终一致性。

那么如何实现最终一致性呢？你首先要知道它以什么为准，因为这是实现最终一致性的关键。一般来说，在实际工程实践中有这样几种方式：

以最新写入的数据为准，比如 AP 模型的 KV 存储采用的就是这种方式；

以第一次写入的数据为准，如果你不希望存储的数据被更改，可以以它为准。

那实现最终一致性的具体方式是什么呢？常用的有这样几种。

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。

在这里，我想强调的是因为写时修复不需要做数据一致性对比，性能消耗比较低，对系统运行影响也不大，所以我推荐你在实现最终一致性时优先实现这种方式。而读时修复和异步修复因为需要做数据的一致性对比，性能消耗比较多，在开发实际系统时，你要尽量优化一致性对比的算法，降低性能消耗，避免对系统运行造成影响。

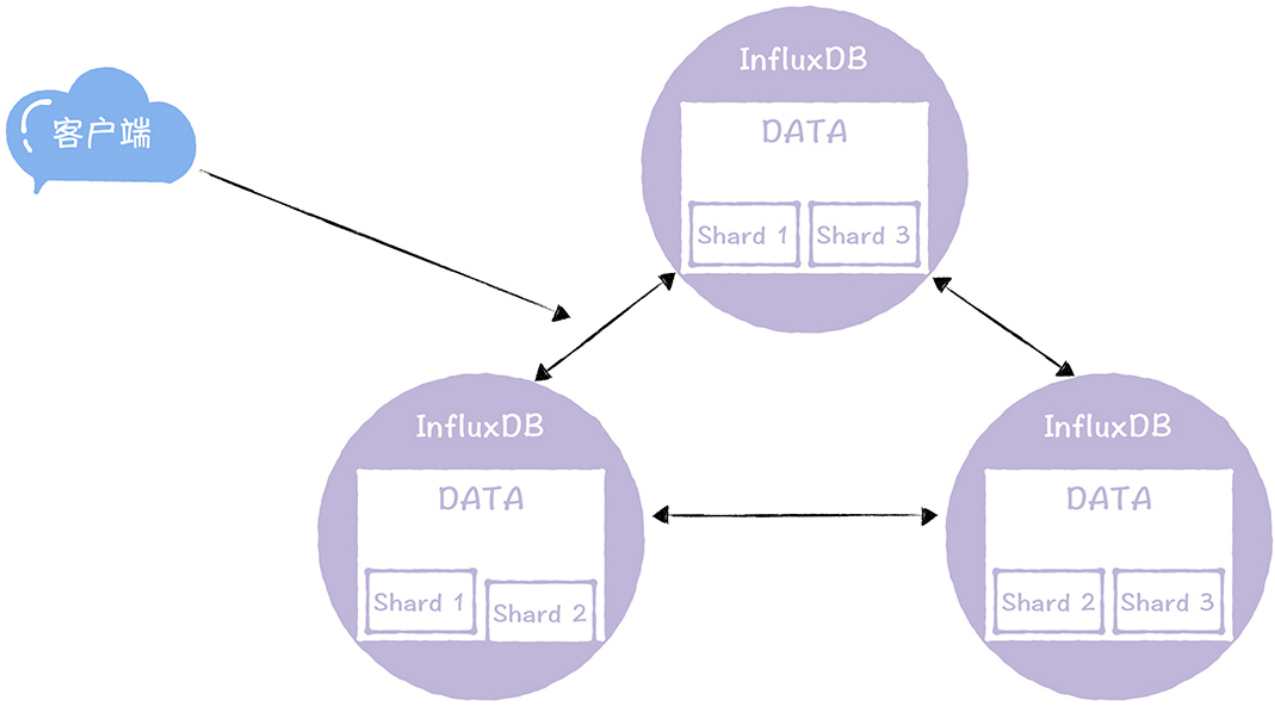
另外，我还想补充一点，在实现最终一致性的时候，**我推荐同时实现自定义写一致性级别（All、Quorum、One、Any），** 让用户可以自主选择相应的一致性级别，比如可以通过设置一致性级别为 All，来实现强一致性。

现在，想必你了解了 BASE 理论的核心内容了吧？不过这是理论层面上的，那么在实践中，该如何使用 BASE 理论的呢？

如何使用 BASE 理论

我以自研 InfluxDB 系统中 DATA 节点的集群实现为例，带你来使用 BASE 理论。咱们先来看看如何保障基本可用。

DATA 节点的核心功能是读和写，所以基本可用是指读和写的基本可用。那么我们可以通过分片和多副本，实现读和写的基本可用。也就是说，将同一业务的数据先分片，然后再以多份副本的形式分布在不同的节点上。比如下面这张图，这个 3 节点 2 副本的集群，除非超过一半的节点都故障了，否则是能保障所有数据的读写的。



那么如果实现最终一致性呢？就像我上文提到的样子，我们可以通过写时修复和异步修复实现最终一致性。另外，还实现自定义写一致性级别，支持 All、Quorum、One、Any 4 种写一致性级别，用户在写数据的时候，可以根据业务数据的特点，设置不同的写一致性级别。

内容小结

本节课我主要带你了解了 BASE 理论，以及 BASE 理论的应用，我希望你明确几个重点：

1. BASE 理论是对 CAP 中一致性和可用性权衡的结果，它来源于对大规模互联网分布式系统实践的总结，是基于 CAP 定理逐步演化而来的。它的核心思想是，如果不是必须的话，不推荐实现事务或强一致性，鼓励可用性和性能优先，根据业务的场景特点，来实现非常弹性的基本可用，以及实现数据的最终一致性。
2. BASE 理论主张通过牺牲部分功能的可用性，实现整体的基本可用，也就是说，通过服务降级的方式，努力保障极端情况下的系统可用性。
3. 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。BASE 理论在很大程度上，解决了事务型系统在性能、容错、可用性等方面痛点。另外我再多说一句，BASE 理论在 NoSQL 中应用广泛，是 NoSQL 系统设计的事实上的理论支撑。

最后我强调一下，对于任何集群而言，不可预知的故障的最终后果，都是系统过载。如何设计过载保护，实现系统在过载时的基本可用，是开发和运营互联网后台的分布式系统的重中之重。那么我建议你，在开发实现分布式系统，要充分考虑如何实现基本可用。

课堂思考

我在文章中提了一些实现基本可用的方法，比如流量削峰、延迟响应、体验降级、过载保护等，那么你不妨思考一下，还有哪些方法可以用来实现基本可用呢？欢迎在留言区分享你的看法，与我一同讨论。

最后，感谢你的阅读，如果这篇文章让你有所收获，也欢迎你将它分享给更多的朋友。

上一篇　03 | ACID理论：CAP的酸，追求一致性

下一篇　05 | Paxos算法（一）：如何在多个节点间确定某变量的值?

精选留言 (17)

写留言

cricket1981
2020-02-20

All、Quorum、One、Any一致性级别中One和Any的区别是什么?

> 作者回复: one表示必须写成功一个节点，any表示所有节点都没写成功，如果请求成功保存到了失败重传的缓存队列中，也算成功。any是最弱的写一致性级别。

4

川杰
2020-02-19

请问：写时修复：在写入数据，检测数据的不一致时，进行修复；这个比对，是指和其它节点数据比对吧？那么数据以谁为准呢？极端情况下，每个节点上的数据都不一样，我该听谁的?

作者回复: 可以这么理解，写时修复，写失败时，将数据缓存到本地磁盘上，然后周期性的重传，本质上，就是失败重传。
第二个问题，所有数据，都不一样，这时现实场景肯定会出现的情况，可以通过多次执行异步修复，来实现一致性。具体的实现方法，我会在11讲，具体说说:)

2

Purson
2020-02-19

还有熔断和限流。 发现文稿里面关于最终一致性中，写的描述有矛盾： “写时修复：在写入数据，检测数据的不一致时，进行修复。” 后面 “在这里，我想强调的是因为写时修复不需要做数据一致性对比，性能消耗比较低，” 前面说写的时候有做一致性对比，后面说写的时候没有做一致性对比。其实写的时候修复主要通过失败重试的方式吧?

展开

作者回复: 加一颗星:)，是的，通过写操作的失败错误，发现不一致，然后通过重传修复数据的不一致。

2

2020-02-20

acid是数据库系统经典之作；base是在实践中受挫后的思想松绑，提出一种重要的指导，给人以信心

作者回复: 加一颗星:)，互联网只能base，可用性和钱，低性能堆机器，都是钱:)

1

Sinclairs
2020-02-19

从微服务的角度来考虑, 有这些方式能够尽可能地保证系统的基本可用:
1. 使用消息队列, 对偶然的高并发写操作进行削峰填谷;
2. 对进程间的服务调用做好熔断保护;
3. 在系统能力无法支撑高并发访问时, 对非核心业务降级;
4. 对关键服务做好限流.

展开

作者回复: 加一颗星:)

1 1

陈
2020-02-22

我觉得还有故障和服务隔离，弹性扩容等等。

展开

作者回复: 加一颗星:)

施耀南
2020-02-21

谢谢老师课程，基本明白，为保证基本可用除了上述方案，我们游戏业务中，还有流量控制！

作者回复: 加一颗星:)

远方
2020-02-21

hadoop的hdfs文件系统和kafka消息队列，按我的理解，也是基于ap的扩展base理论开发的，是这样的吧?

fcb的鱼
2020-02-20

实现最终一致性的方案中,更侧重于存储层面的事情吧。假设有A,B,C三个节点，那么"读时修复"方案是不是意味着每个读操作，会同时访问这三个节点吗，然后比对三个节点返回的数据是否一样，一样的话返回任意一个节点的数据，不一样了再以最新的数据为准，并且修复其他不同节点的数据？在AP模型的应用中，是不是得同时校验几个节点的数据是否一致?

展开

作者回复: 每种方案，都有自己比较适合的场景的，比如读时修复，在实现了Quorum NWR时，意义比较大，因为，本来就需要读多副本，发现不一致了，就顺便修复。
在AP模型系统中，一般是需要实现异步修复的，不过具体的技术方案，还是取决于场景。
咱们这么理解哈，技术是解决场景问题的，使用哪些技术？如何使用技术？是取决于场景的，也就是需求。

fcb的鱼
2020-02-20

在使用Base理论的举例中，3节点2个副本的架构中，再各个节点中这2个副本的数据最终要是一样的吧？在一个系统整体的架构中，上游的系统可以使用Base理论，但是底层的系统如支付系统需要使用强一致性，这样的话，支付系统就称为了瓶颈。在这种场景中，整个系统的架构应该如何考虑呢？

展开

作者回复: 从描述看，这是一套复杂的系统，设计混合架构吧，根据场景特点不同，做拆分，这么说可能比较抽象。如果有更具体的信息和需求，咱们可以一起展开讨论下:)
最后补充一点，架构之道，在于权衡妥协。

Michael Tesla
2020-02-20

老师好，请教下这个 3 节点 2 副本的InfluxDB集群的工作流程：
1. 正常工作时，某个分片的数据只在节点1写入，然后节点1将数据同步到节点2。
2. 如果节点1挂了，客户端就将数据写入节点2。
3. 之后，等节点1恢复，节点2将数据同步回节点1（即写时恢复），同时客户端也重新将数据写入节点1。

展开

作者回复: 加一颗星:)

小晏子
2020-02-19

这几个“流量削峰、延迟响应、体验降级、过载保护”都是为了实现业务的基本可用的手

段。比如在秒杀场景中，会突然之间有大流量的请求涌入，而对付这些过载流量，就需要使用这几个手段来保障业务的可用性。

展开

作者回复: 加一颗星:)

盘胧
2020-02-19

在最前端链路就做好流量控制，和最后的兜底方案

展开

作者回复: 加一颗星:)

Fs
2020-02-19

流量削峰、延迟响应、体验降级、过载保护这些也不只是分布式系统问题，单机同样也有

作者回复: 加一颗星:)，可用性敏感的系统，都要考虑的。

hello
2020-02-19

重试、幂等、异步、负载均衡、故障隔离、流量切换、自动扩缩容、兜底（熔断限流降级）、容量规划

作者回复: 加一颗星:)

忆水寒
2020-02-19

限流策略、熔断机制关闭部分服务等。

展开

作者回复: 加一颗星:)

每天晒白牙

2020-02-19

1.服务调用之间做好流量限制，即限流，避免瞬时打进大量流量
2.利用MQ实现削峰填谷
3.熔断降级也是一种保证基本可用的方案，但实际工作中我还没尝试过

展开

作者回复: 加一颗星:)