

Trivia about Distributed

准备比较仓促，接着上次 Procyon 介绍 Paxos 的东风，聊点分布式周边的基础知识，算是锦上添花，也为之后介绍 Raft 打点铺垫。

内容不一定前后关联，可能比较乱，反正就是瞎聊聊😅



什么是分布式系统？

多个程序在不同的机器上独立运行。

互相通过网络交互

机器不可靠：可能宕机、重启

网络不可靠：可能延迟、丢包、分区

整个系统中没有全局共享的内存或共享锁

为什么要分布式？

多活、更多可用资源、用户侧低延迟

分布式有什么局限性？

- 独立的系统各自崩溃
- 光速上限

崩溃不可避免，任何结点或网络都可能崩溃。而光速有限导致了其他节点无法即时获取到集群的状态，网络不可靠也让结点难以区分延迟和故障。

一致性问题

更重要的是，事件在分布式系统中独立分散地发生，对系统一致性提出了挑战。

我们该如何给这些事件排序，并达成最终一致性？

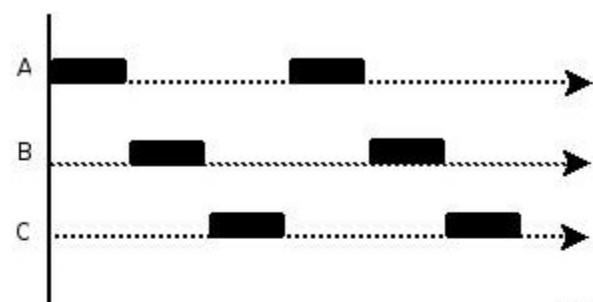
有哪些一致性？

- 强一致性
 - 线性一致性 (linearizable consistency) : 单核串行执行
 - 序列一致性 (sequential consistency) : 并行执行，但是结果等效线性一致性
- 弱一致性
 - 客户端中心一致性 (client-centric consistency model) : 客户端写缓存，过滤服务端的过期读响应
 - causal consistency : 保证因果续一致性
 - 最终一致性 (eventual consistency model) : 当写停止，系统保证最终能够达成一致

我个人感觉啊，可以用 ACID 里的 durability 的视角来理解一致性。

就是你提交了一个更改后，这个更改对自己可见吗？对其他人可见吗？会最终留存吗？

【并发】 / 【并行】



Concurrency :
1. Single Processor
2. logically simultaneous processing



Parallelism :
1. Multiprocessors, Multicore
2. Physically simultaneous processing

一个单核 CPU，跑多进程分时系统，就是并发。

分布式系统的模型

- 同步系统 (synchronous system model) : 每个节点都有准确的时钟，网络有已知的延迟上限。
- 异步系统 (asynchronous system model) : 没有全局时钟

分布式虽然是个学术领域，但是最终还是要解决业务问题。所以根本上还是看业务方对一致性的实际需求和容忍性问题。一致性很大程度上是一个 tradeoff 或者概率问题。

同步系统 的典范就是 Google 基于 TrueTime 的 Spanner。或者各种试图根据 timestamp 来处理数据变更的架构设计。

本质上就是基于实际需求容忍一定范围内的时间误差。我们可以将其称为“延迟存在上限（时钟误差存在上限）的系统”。

时钟

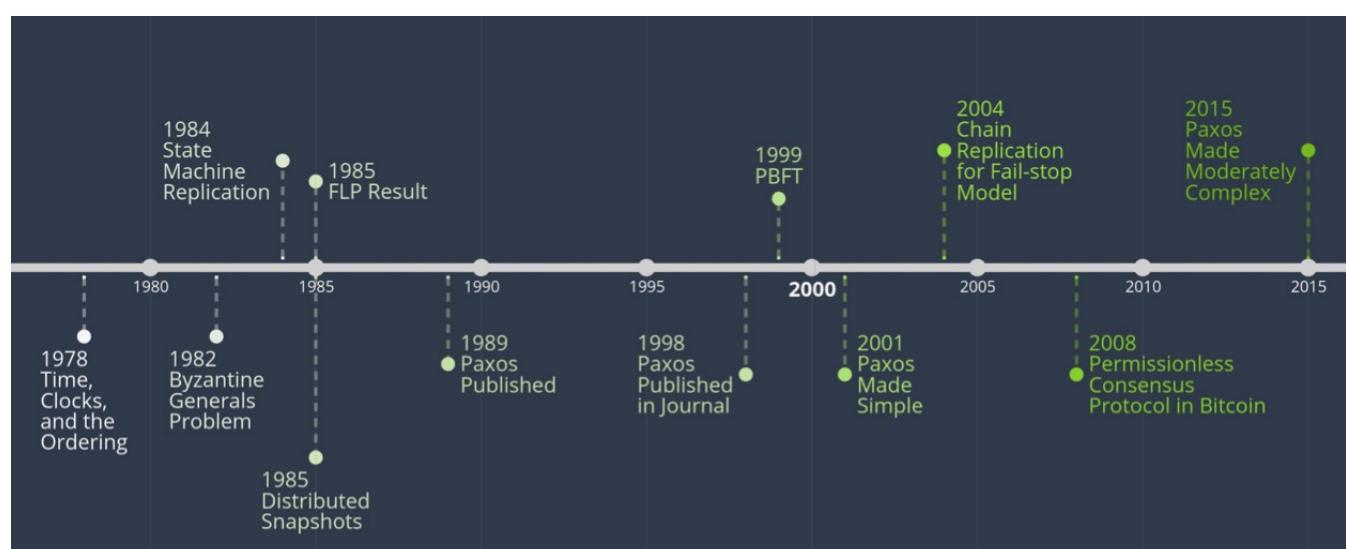
本次话题主要关心的是 异步系统，这是一个更逻辑自治的设计。光速有限和网络延迟让所有节点不可能协商出一个【真正的】全局时钟。

但是分布式的结点不停地在各自发生各种事件，要想达到最终一致性，我们必须有个顺序来应用这些事件导致的变更，该如何确认顺序呢？

这个问题，实际上构成了整个异步分布式系统领域的基石。

1978 年 Lamport 发表的《Time Clocks and the Ordering of Events in a Distributed System》正式提出和定义了这个问题。

实际上也正是这篇论文开创了异步分布式系统这个研究领域。



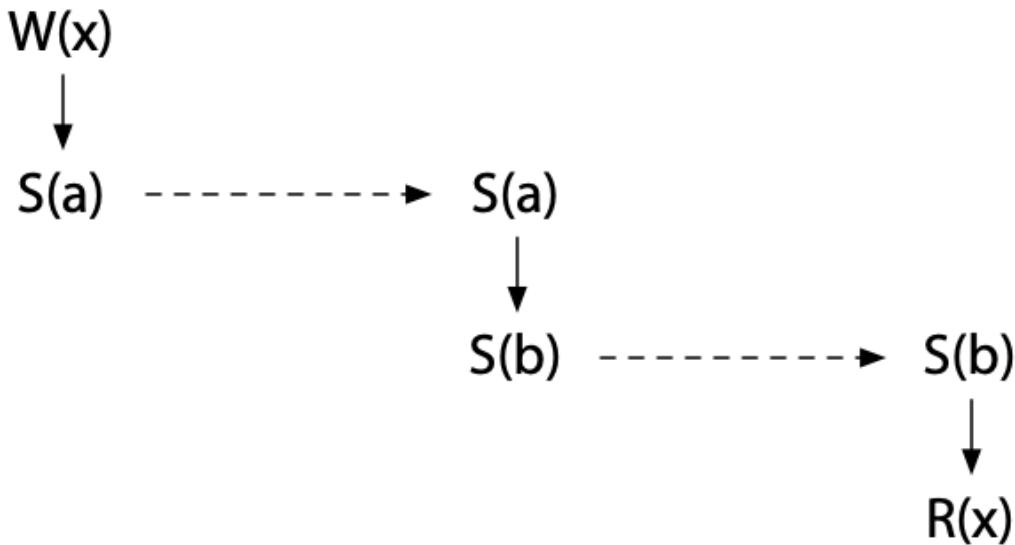
文章中首先阐述了分布式事件在不同结点上所产生的排序问题。

然后提出了一个根据事件因果关系构建偏序关系的模型，这一模型后来也被称为 Lamport Clock

Thread 1

Thread 2

Thread 3



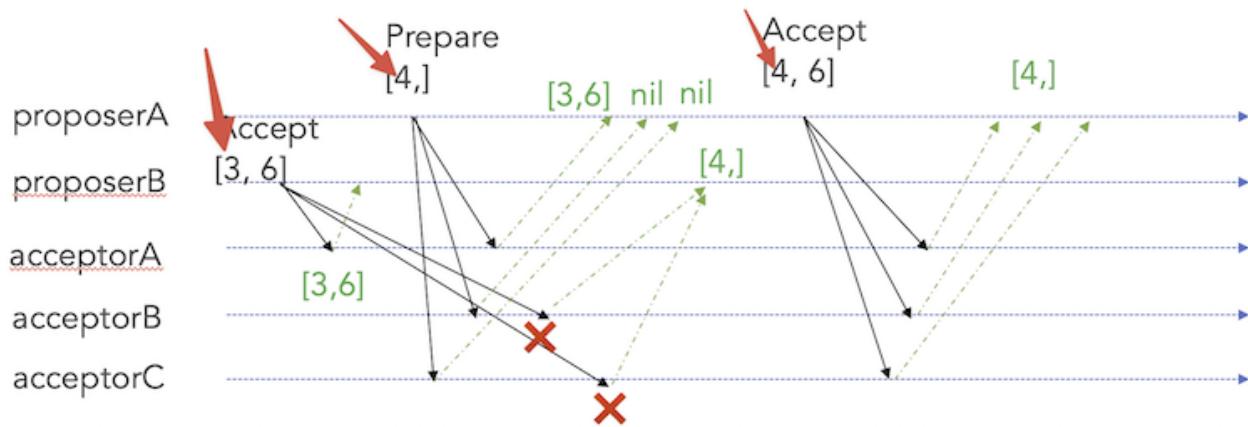
以跨结点事件为锚点，我们就可以构建出不同事件的因果关系，也被称为 **HAPPEN BEFORE** 联系。

不是所有的事件都可以被排序，所以这种方式只能为一部分事件排序，因此也被称为 **偏序 (partial order)**，与此对应的是，在有全局时钟的系统中，进行的完全排序被称为 **全序 (total order)**。

因为这种方法是根据因果关系进行排序，所以也被称为 **因果序 (casual order)**

有点抽象？

想想上次介绍的 Paxos 中每次数据变更都有的一个全局唯一的 number，那其实就是 casual order。



或者在 Raft 中的 Term，那其实也是 casaul order

Synchronization 同步原语

跑个题

在分布式领域的 Casual Order，实际上在编程语言里也有应用

任何一门语言，当你想进行并行编程的时候，都需要去看一下它的同步原语是什么。

或者用俗一点的话来说，你要去看一下它的锁有哪些，它的操作的 thread-safe 怎么样？

拿 Golang 来说，首先它有一篇文档介绍 GMM <https://go.dev/ref/mem>，然后还提供了 `-race` 命令来检查 race。

`-race` 实际上就是在你的代码中以同步原语（跨 goroutine 事件）为锚点，构建所有对内存操作的 casual order。

理论上所有的内存操作都应该最终构建出一个 total order，一旦出现无法被排序的事件，那么其实就说明出现了 race。

从 lamport clock 和内存模型的同步原语可以看出，这类时钟算法（或者叫分布式排序算法）的核心目的实际上是冲突检测。

它提供了一个工具，让你可以找出分布式系统中的不一致，而怎么解决这些不一致，就是一致性模型要做的事情。

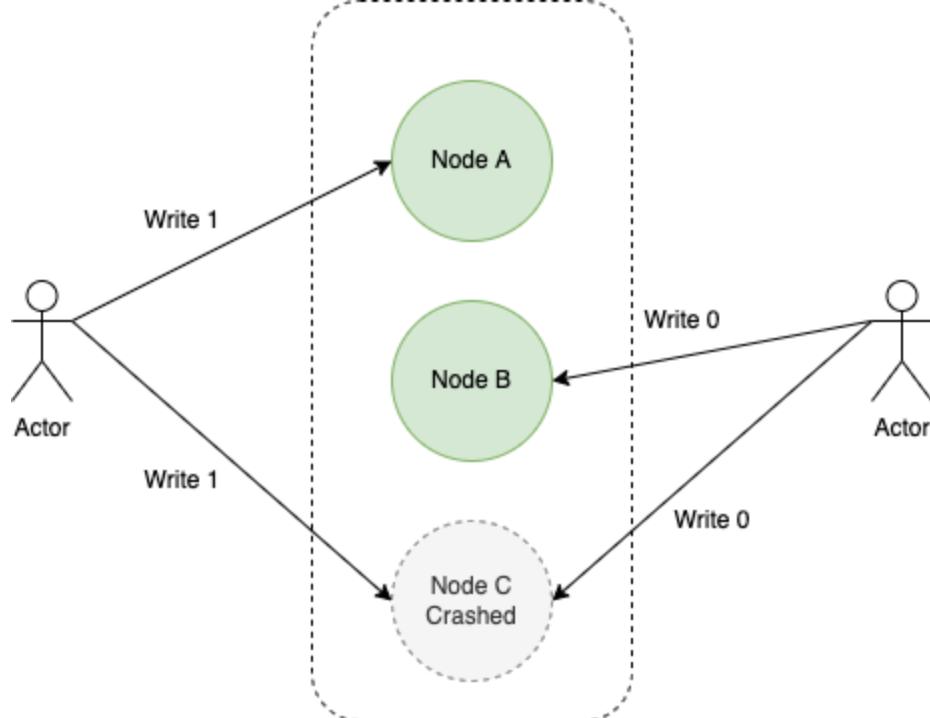
一致性模型

FLP 不可能性

No completely asynchronous consensus protocol can tolerate even a single unannounced process death

FLP 不可能原理：在网络可靠、但允许节点失效（即便只有一个）的最小化异步模型系统中，不存在一个可以解决一致性问题的确定性共识算法。

FLP 提出一个算法不可能在保证一致性的情况下，还能在有限时间内结束。



这个 C 恰到好处地偶尔崩溃，导致 client 一直无法完成多写，
可能性是有的，但一直维持这状态的概率太低了

简而言之，一轮不行就多来几轮，人不能一直倒霉

感觉就像薛定谔不确定性或哥德尔不完备性一样，虽然我们这个世界的物质基础和逻辑基础都是支离破碎的，

但是缺口不是很大，不妨碍我们继续假装不受影响的生活 😊

一致性模型

既然 FLP 有解，人们就提出了很多一致性模型来试图解决分布式问题。

分布式问题分类

有个术语大家可能经常会听到：

拜占庭将军问题

一组拜占庭将军分别各率领一支军队共同围困一座城市。为了简化问题，将各支军队的行动策略限定为进攻或撤离两种。因为部分军队进攻部分军队撤离可能会造成灾难性后果，因此各位将军必须通过投票来达成一致策略，即所有军队一起进攻或所有军队一起撤离。因为各位将军分处城市不同方向，他们只能通过信使互相联系。在投票过程中每位将军都将自己投票给进攻还是撤退的信息通过信使分别通知其他所有将军，这样一来每位将军根据自己的投票和其他所有将军送来的信息就可以知道共同的投票结果而决定行动策略。

简而言之，分布式的结点，如何达成共识？

虽然都是共识问题，但是还有细分：

- 拜占庭将军问题：结点不可信，可能有人伪造消息
- 非拜占庭问题：结点可信，只是不可靠（宕机、延迟）

我们此次只关注于非拜共识问题，旨在可信结点群中实现共识，常见算法有 Paxos、Zab、Raft 等。

拜占庭问题更多的是区块链等关注的领域，相关的算法有 PoW、PoS 等。

我们关注什么样的分布式系统？

综合前面的信息，我们关注的实际上是【非拜占庭异步分布式系统】。

一致性模型一览

1. 主备 M/S：单点写，从只读
2. Gossip：
3. 两部提交 2PC：多主、脑裂、要求所有节点 approve
4. 选主 Quorum：单主

	M/S	Gossip	2PC	Quorum
Consistency	Eventual		Strong	
Transactions	Full	Local	Full	
Latency	Low		High	
Throughput	High		Low	Medium
Data loss	Some		None	
Failover	Read only	Read/write		

$R+W>N$

在 2PC、Quorum 类型的模型中，常常会看到一个说法是 $R+W > N$ 的一致性约束。

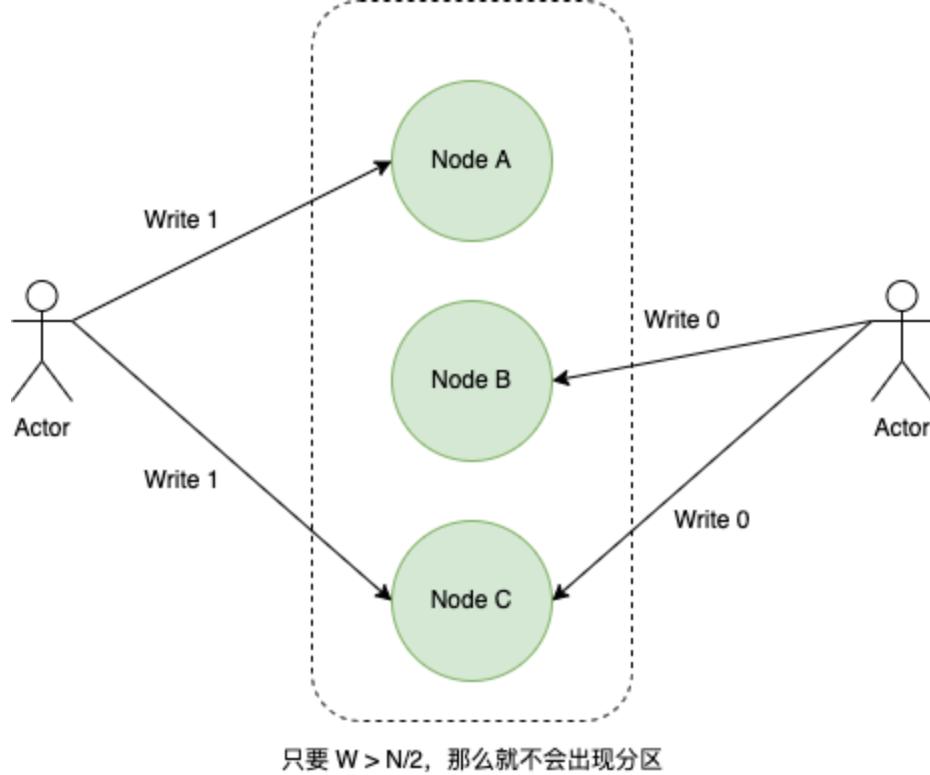
简而言之就是：假设有 N 个集群，每次写 W 个结点，每次读 R 个结点，只要 $R + W > N$ ，就能保证数据一致性。

拿 ElasticSearch 来说，它就像是一个多主集群，你完全可以把它当成一个分布式共识系统来用 😅。

Index API 中有一个参数是 `wait_for_active_shards`，就是用来指定 W 的数量。

Search API 和 GET API 中可以返回每一个 document 的 `_seq_no` 和 `_primary_term`。你可以读取 R 个 ES 结点，然后取其中 seq & term 最高的值为最终结果。

这样就实现了对一个分布式集群的读写一致性。

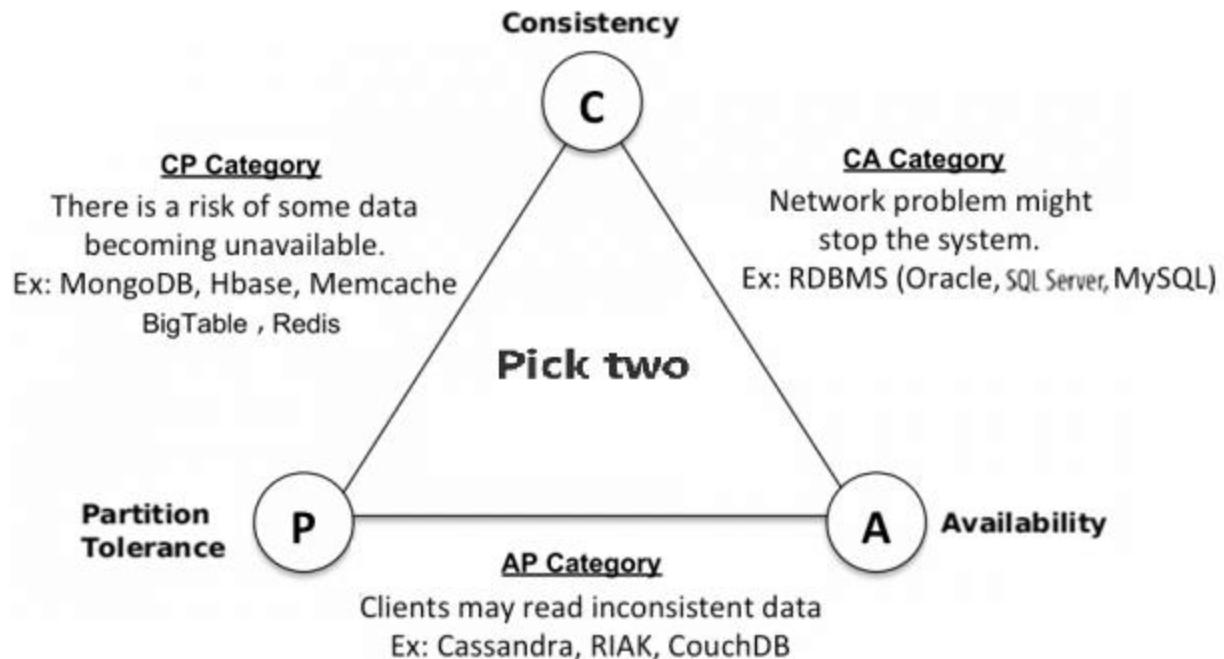


如果 $W < N/2$, 那么这就是个最终一致性, 比如 Gossip, 可能存在数据分区和丢失。

如果 $w > N/2$, 那么这就是个强一致性, 如 Raft。

CAP Theorem

CAP 不可能三角

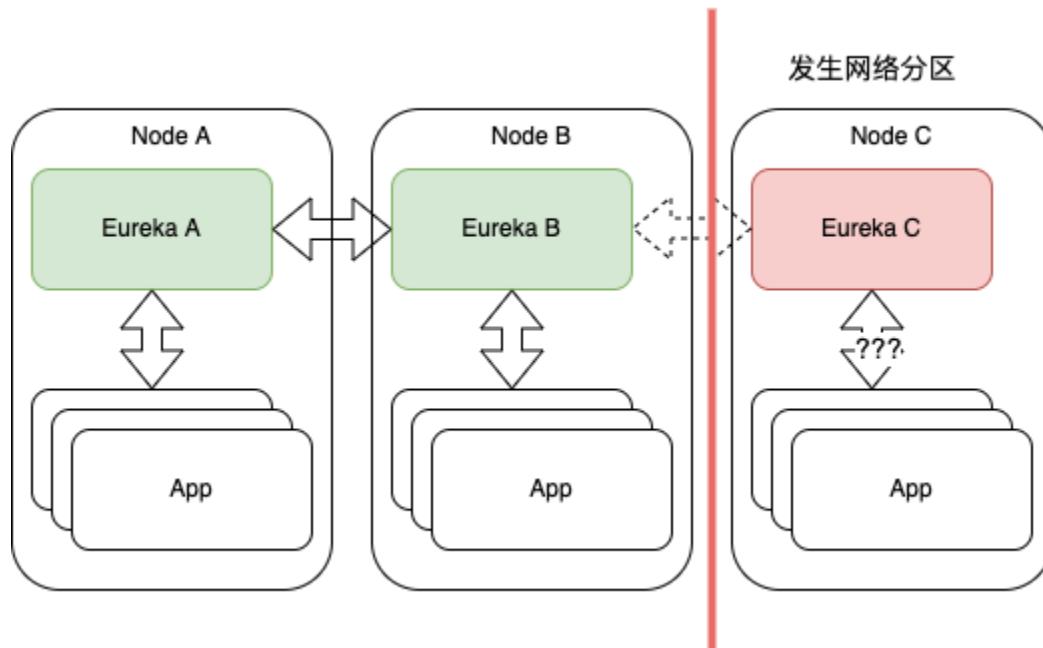


既然讨论分布式系统, 那么 P 是一定要的。问题就是 C 和 A 选哪个?

如果是共识系统 (Raft、Consoul、ZK), 或者比较重要的数据系统 (MySQL), 那么 C 就是需要的, 也就是会选 CP。

如果可用性比较重要，能够容忍数据分区，那么就会选 AP，如 Eureka 等配置管理系统。

有时候你会听到有人讲：CP 不适合做配置管理，所以不要用 ZK 做配置管理器，而要用 Eureka，为什么？



当网络分区时，少数派结点是要终止服务等待一致性，还是放弃一致性继续保持可用性？

具体业务场景具体分析。

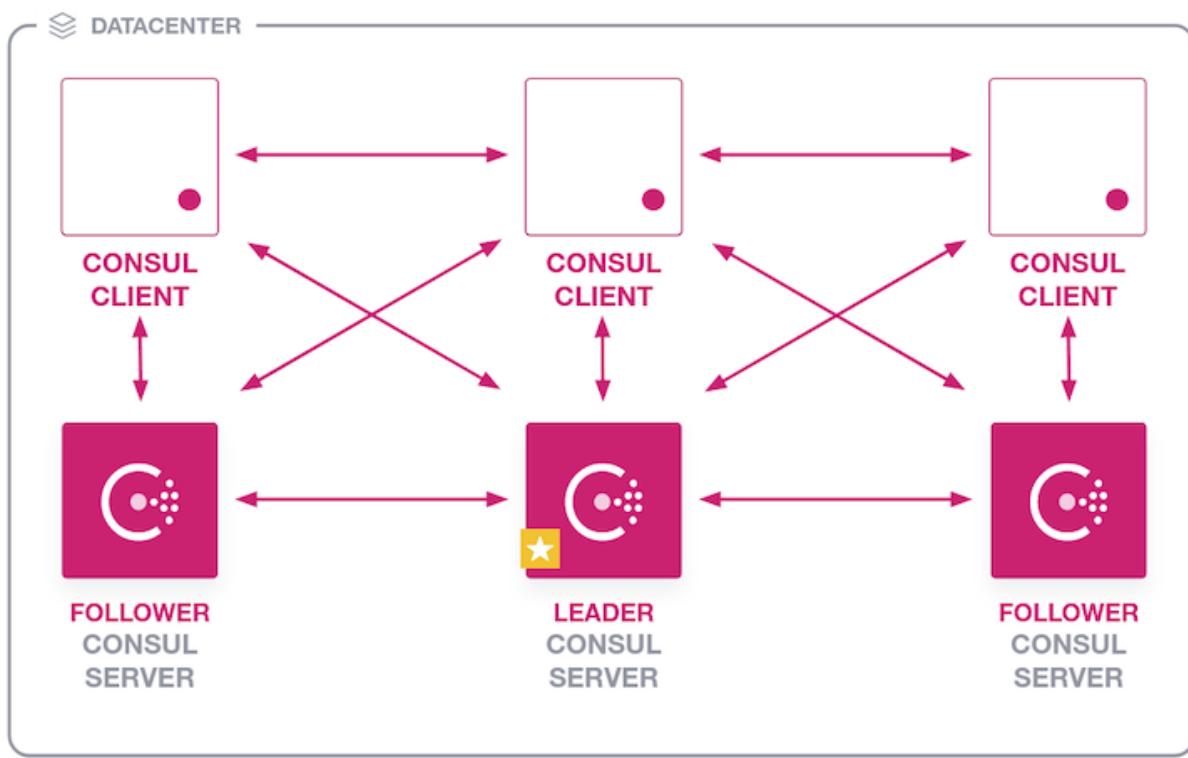
像 ConfigServer 类型的场景，变更并不频繁，可用性高于一切，所以说 ZK、ETCD 之类的 CP 不适合拿来用。

但是问题又来了，Consul 也是 CP 吧，为啥官方一直宣称自己是 ConfigServer ？

我当时也很奇怪，去观察了一下发现，consul 除了 server 外还有个 agent !

app 实际上是在和 agent 通讯，agent 定期去 server 拉取更新。

所以，虽然 consul server 是 CP，但是它挂了不会影响 A。通过 agent 的本地缓存，实际上把 CP 系统转换为了 AP。



其他一些来不及介绍的

其他一些话题或领域，今天来不及写了，简单提一下

密码学界有句话叫做“不要轻易尝试自己设计一个加密算法”。

分布式界也有句类似的话叫做“不要轻易尝试自己设计一个分布式一致性算法”。

加密算法的问题是你很难自证它是安全的。分布式一致性算法的难点在于你很难自证它在复杂的网络环境下能保证一致性。

那我们怎么保证一个分布式算法的一致性呢？

有一个叫做形式化验证的领域，借助 TLA+ 等工具，可以对其进行验证。

像最终一致性这种东西，尤其是 Gossip 算法，我怎么知道它要多长时间才能达成一致？

有一个叫做 PBS: Probabilistically Bounded Staleness 的工具，可以用来预估一致性延迟

对于一些特殊的操作类型和数据，可以采用 CRDT(convergent replicated data types) 的方式，实现自动的冲突合并和强一致性。

多被应用于在线多人协作编辑等。

除了 CRDT 外，还有种叫做 CALM (consistency as logical monotonicity) 的东西也可以保证强一致性。

这两个东西我都只熟练掌握了名字拼写，就不介绍了😅。

谢谢

瞎聊到此结束🙏

杂七杂八的讲了一些铺垫，期待同事们在后续的分享里介绍具体的实现